TRS8BIT

PRODUCED AT WWW.TRS-80.ORG.UK

SEPTEMBER 2026 - VOLUME 20 ISSUE 3

PAGE 02 — TRS80GP - LATEST UPGRADE DETAILS
GEORGE & PETER PHILLIPS

PAGE 04 — TANDY ASSEMBLY 2026
IAN MAVRIC
LATEST NEWS AND DETAILS OF SPEAKERS FOR THIS YEAR'S EVENT WHICH IS BEING HELD ON THE 2ND TO THE 4TH OCTOBER 2026 IN CINCINNATI, OH. ARE YOU LUCKY ENOUGH TO BE GOING?

PAGE 11 — FOR SALE - LARGE DONATION GIVEN TO THE SITE
I'M LOOKING FOR OFFERS AS A 'JOB LOT' PLEASE. (RE-SELL ON EBAY AND TRIPPLE YOUR MONEY! OR JUST INCREASE YOU COLLECTION!)
BUYER TO COLLECT, ESTATE OR SMALL VAN NEEDED!

PAGE 18 — FLOPPY DAYS - TRANSCRIPT, EPISODE 8, PART 1
RANDY KINDIG

PAGE 31 — KALEIDOSCOPE
E.T. FONEHUME
ANOTHER FANTASTIC BASIC ONE-LINER. I'LL MENTION THE 'FUN' GEORGE AND I HAS WITH IT, ON THE NEXT TTLIVE EVENT!

PAGE 34 — XII BASQUE TRS-80 TOURNAMENT
EGOITZ CAMPO
HERE ARE THE RESULT FROM THE LATEST COMPETITION.

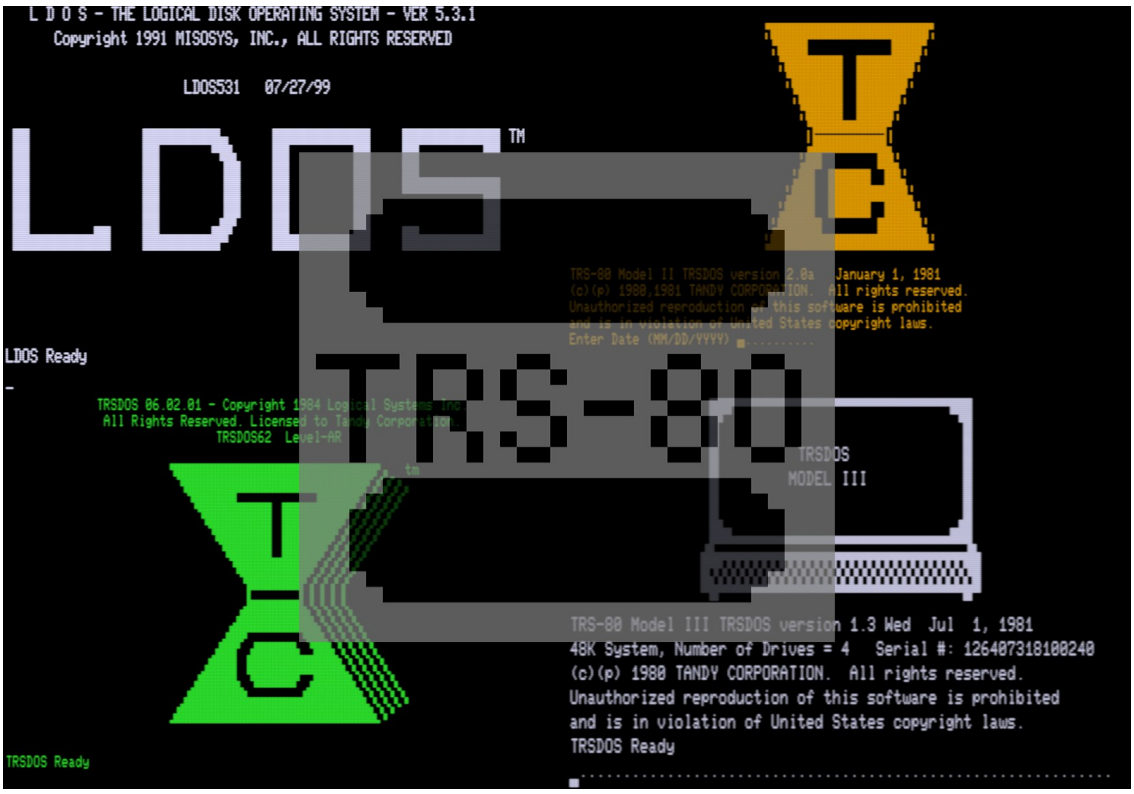
PAGE 38 — THE POWER OF THE ROM
VINCE OTTEN

PAGE 59 — THE WAY WE WERE
ANDREW HOWARD
A LOOK BACK TO 'THE DAY' WITH AN INTERESTING SOFTWARE UPGRADE FOR THE VIDEO GENIE/SYSTEM-80

PAGE 70 — 'COUNTDOWN' QUIZ
CAN YOU WORK OUT THE ANSWER IN UNDER 30 SECONDS?

PAGE 74 — TOUGH TIMES FOR RETRO COMPUTER BUYERS
IAN MAVRIC
MORE 'SAD' NEWS OF RETRO COMPUTER LOVERS

TRS80GP - LATEST UPGRADE
VERSION 2.5.8



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



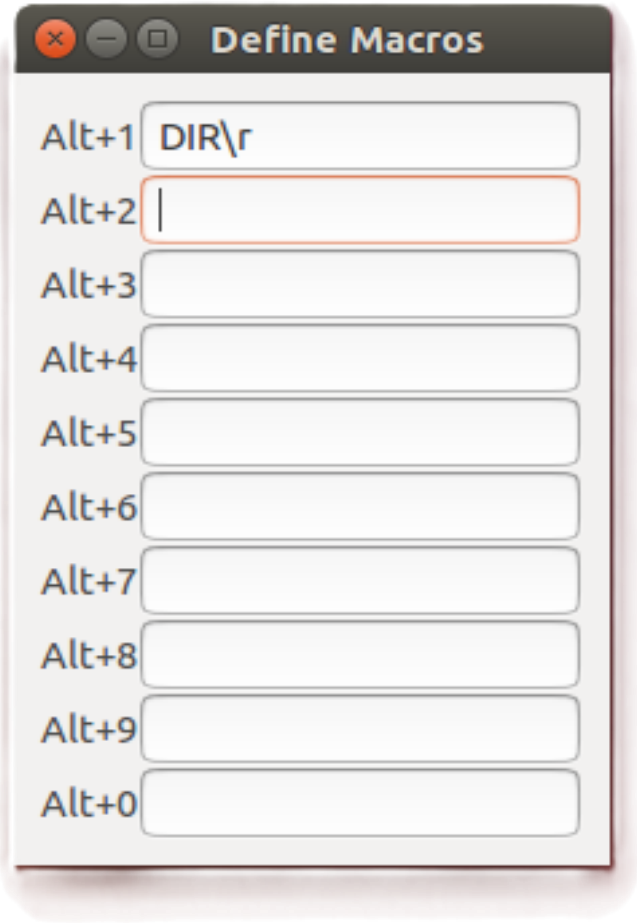
FOR WINDOWS, MACOS, LINUX & RASPBERRY PI.
EMULATES MODELS 1, 2, 3, 4, 12, 16, 6000, MC-10,
COLOR COMPUTER, DT-1, VIDEOTEX, TANDY 10 AND 'GREEN THUMB'.

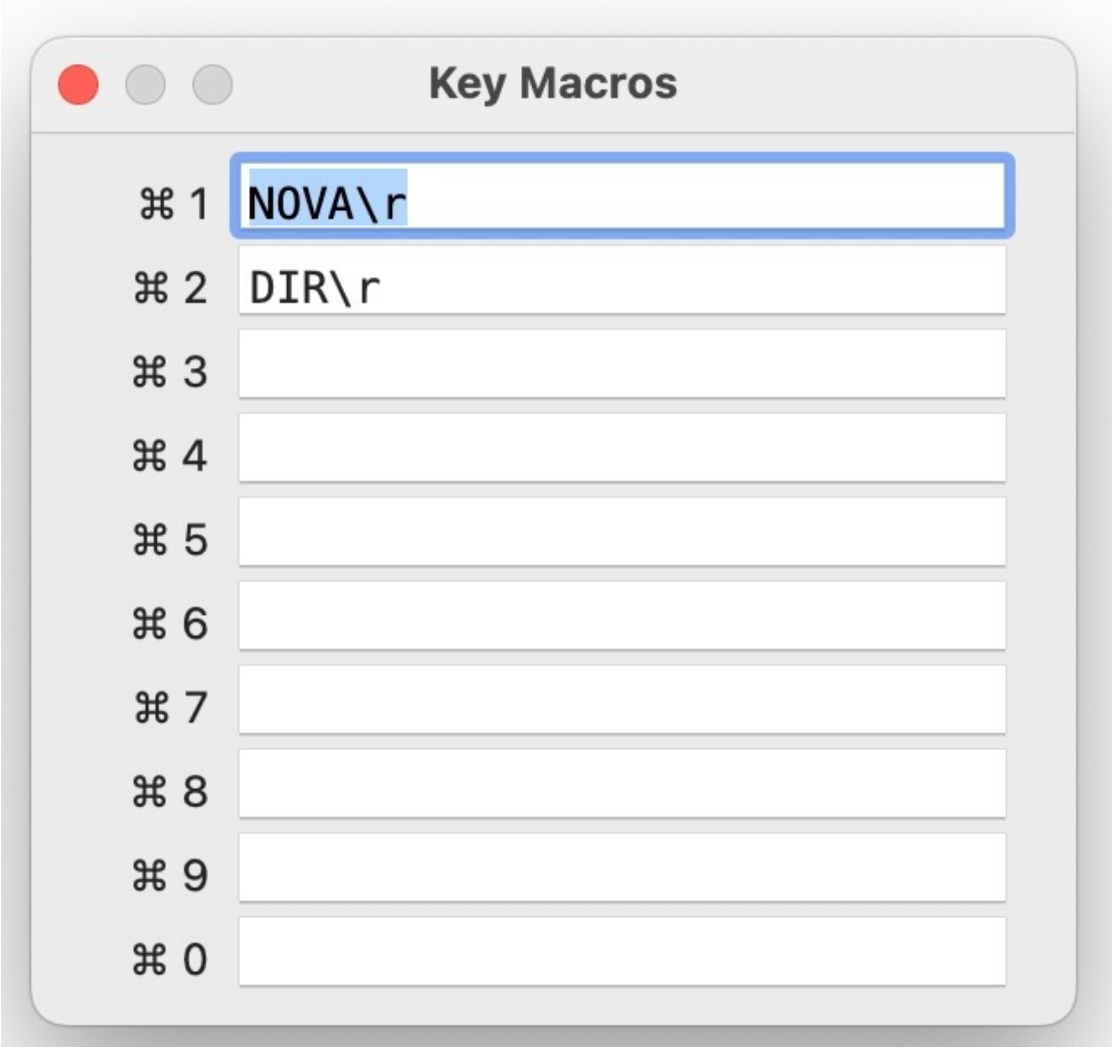
JUST LIKE A REAL TRS-80 WITH NONE OF THE HASSLE.
BUILT-IN DEBUGGER AND OTHER FEATURES, JUST GREAT FOR PROGRAMMERS.

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

UPDATED VERSION, 2.5.8, ADDITIONALLY OFFERS :

PASTE MACROS FOR MACOS AND LINUX.
PASTE MACROS CAN BE DEFINED ON THE COMMAND LINE DISK SUMMARY CAN SHOW LIKELY EMPTY SECTORS.

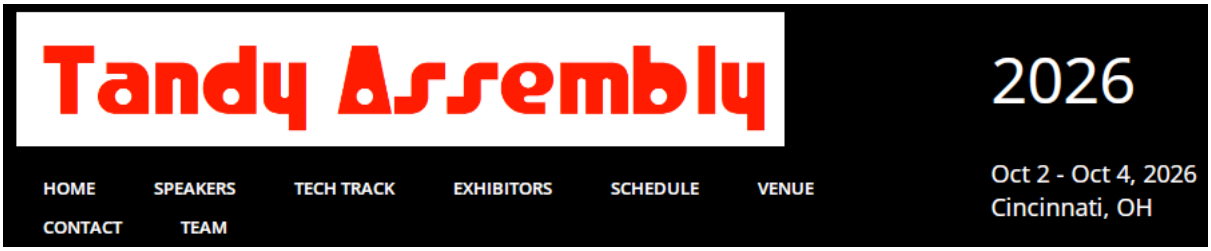

Define Macros
Alt+1
DIR\r
Alt+2
Alt+3
Alt+4
Alt+5
Alt+6
Alt+7
Alt+8
Alt+9
Alt+0


Key Macros
⌘ 1
NOVA\r
⌘ 2
DIR\r
⌘ 3
⌘ 4
⌘ 5
⌘ 6
⌘ 7
⌘ 8
⌘ 9
⌘ 0

MAV'S TANDY ASSEMBLY QUARTERLY COLUMN

TA26 NEWS

by Ian Mavric



Bringing you the latest news you need to know about the greatest Tandy TRS-80 show on the planet. With less than one month now until the show it's time to get ready for those planning the long trip to Cincinnati Ohio.

FIRSTLY, SOME SAD NEWS: VALE JERRY HEEP

It's with a heavy heart I have to be the bearer of sad news that Jerry Heep, who was going to do a keynote talk at TA26 unexpectedly passed away in late August. We were all looking forward to meeting him and hearing about his time working at Radio Shack on, among other things, his work on the Color Computer.



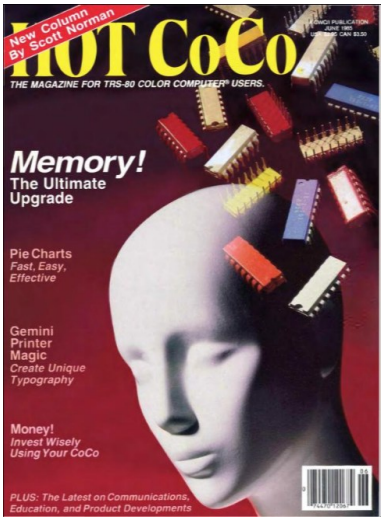
Michael Nadeau

"80 Micro, HOT CoCo, and the Early Days of Computer Magazines"

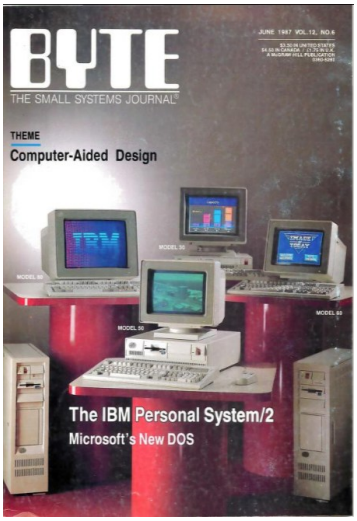


Michael Nadeau has spend more then 40 years in tech publishing, from the early days of microcomputers to the era of enterprise software and cloud computing. He joined Wayne Green Inc. at 80 Microcomputing as a junior editor in April 1981. He became editor-in-chief of HOT CoCo when that magazine launched, eventually returning to 80 Micro as editor-in-chief. At BYTE, he first managed the magazine's testing lab and then became executive editor. Nadeau will share stories of the early days of computer publishing as well as anecdotes about the early computer industry.

Bio: Michael Nadeau is the former editor-in-chief of 80 Micro and HOT CoCo magazines, and the former executive editor of BYTE. Recently retired, he has also worked for or contributed to many other computer and technology publications such as CSO, Data Informed, SAP Experts, and Family PC. Nadeau is also the author of Collectible Microcomputers and the BYTE Guide to CD-ROM. He currently volunteers at local energy-related organizations and writes articles for cybersecurity publications.







THE CLASSES

TTT101

Title: Soldering, Desoldering and Welding Electronics

Instructor: Jeff Birt (Hey Birt!)

Home: Rolla, Missouri, USA



Abstract: Invariably when working on our vintage computers, we'll all run into the need to solder, desolder or weld electrical connections. We'll first take a quick look at how our precious circuit boards were originally produced and then look at various methods for rework when replacing components and/or doing upgrades. Topics include: original production methods, avoiding unnecessary rework, reworking tools and techniques, welding electronics

Bio: Jeff grew up in central Ohio about 30 miles from Springfield where Tandy Assembly was formerly held. The past 31 years he has lived in the Missouri Ozarks. A jack of all trades, he has repaired computers, fax machines, worked in industrial automation and electronics manufacturing before finally settling in for the last 23 years at Missouri University of Science and Technology as an engineer designing and building machinery and apparatus for research projects. An interest in vintage computing sprang up about 15 years ago in an effort to recover childhood C64 disks and replace a long-lost Tandy PC-6.

TTT102

Title: The XLR8er

Instructor: Scott Toenniessen

Home: Buzzards Bay / Cape Cod, MA



Abstract: The XLR8er gave the TRS-80 Model 4 a serious performance boost, and Scott traces its full story—from its roots in Steve Ciarcia's SB-180 single-board computer (Byte Magazine), built around an 8MHz HD64180 CPU and 256K of RAM, through its takeover by Misosys and later efforts to revive the board. He covers the hardware (RAM upgrade requirements, NGA vs. GA differences, his own install in a Model 4P with a Graphyx card and clock chip), but the heart of the talk is the software: original and enhanced Misosys utilities, community LS-DOS patches exploiting the HD64180's native multiply and DMA instructions for gains well beyond the clock bump, fixes for TRSDOS 1.3 and Model III booting quirks, and high-res graphics tools that dramatically sped screen draws. Drawing on six years of daily use in his own 4P and patches he contributed via the Misosys Quarterly and CompuServe's LDOS forum, Scott offers both a history and a practical guide for anyone wanting to pursue the board.

Bio: Scott became a TRS-80 enthusiast as a small child the first time he saw a Model I sitting near the entrance of his local Radio Shack store. As an eighth grader, he had saved up enough to purchase a used Level 1 4K system and then later was happy to acquire a brand-new Model 4P. All the learning and experimenting on those systems inspired him to earn a degree in Computer Science and pursue that as a career. Along the way he has been employed by Eastman Kodak Company and Fidelity Investments, among others. He is currently employed as a site reliability engineer at Advisor360 in the greater Boston area. Now that his five kids are grown, TRS-80s have come back to the forefront taking over part of his Cape Cod home with his original two systems having been joined by a few others. He has used an XLR8er board in his TRS-80 Model 4P since 1988 and was active in that community when many advancements were being developed using the board's capabilities.

TTT103

Title: TrashBoy - TRS-80 Games in your Pocket

Instructor: Sascha Haeberling and Arno Puder

Home: San Francisco, California, USA



Abstract: We've been working on a Game Boy-like version of the TRS-80 that has been optimized for playing the catalog of TRS-80 games, plus it has a few extra features :-). We will show how we designed and built this (hardware and software) and how it ties into some of the other tech we have shown before at TA, such as TRS-IO and RetroStore.

Bio: Sascha Haeberling is a Software Engineer whose career includes 18 years at Google, a stint at an early-stage startup, and his current role at Meta. While his professional focus remains on modern software development, he occasionally dabbles in the vintage computing world. This intermittent dive into TRS-80 hardware and software is almost entirely fueled by the influence of his co-presenter Arno Puder and the Tandy Assembly community.

TTT104

Title: Getting Started with Color Computer (6809) Assembly Language

Instructor: Tim Halloran

Home: Pittsburgh, PA



Abstract: Tim is still working on this but expect an introduction to cartridge and disk versions of EDTASM+, LWASM (LWTOOLS) and more.

Bio: Tim Halloran has a Ph.D. in Software Engineering from Carnegie Mellon University, a M.S. in Computer Engineering from the U.S. Air Force Institute of Technology, and a B.S. in Computer Science from the Air Force Academy. He has 30 years of experience in the software development industry. Tim is a TRS-80 enthusiast of both the Z80 and the CoCo computers.

TTT105

Title: Adding an AM9511 Arithmetic Processing Unit (APU) to a Big Tandy

Instructor: Ruud Broers

Home: Eindhoven, The Netherlands



Abstract: The AM9511 is an interesting chip. It added fixed and floating arithmetic to microcomputers before the 8087 even existed. This made the 24 pin chip with a simple control interface a nice challenge to add to my TRS-80 Model II. In this talk I will explain the small board I designed for it, and how to use it. I use the Basic program language to interface to the chip. No need for Z80 assembler. I will bring my first board to the show.

Bio: I was fascinated by the TRS-80 Model I as soon as it hit the streets in the Netherlands. As a student of the University of Utrecht I purchased the Model I as a 16K machine. Printing listings was performed on a Teletype that was bought used from the Philips Nat Lab in Eindhoven. Very noisy but reliable. Later on my Model 1 was expanded to 64K and received a Dutch (home build) expansion system called the "Utrecht Bus", which used ECB format boards. Functions of the system were floppy disk, serial port and Eprom programmer. My collection of Tandy computers was then expanded with a Model 100 and a Model 4P which ran Newdos80 in 80x24 character mode. The bigger Tandy computers were not within my reach. However, when I was able to purchase a working Model II in 2017 I did not hesitate. Meanwhile that system has been expanded with my 512K RAM board (designed and build by me) for use by the Z80 and a Gotek drive for easy "floppy disk" use. The latest addition to my Model II is the AM9511 Arithmetic Processing Unit (APU) described here: https://github.com/GreyZ80/AM9511_for_TRS-80_M2.

SATURDAY AFTERNOON AND EVENING

THE AUCTION

Attendees and vendors donate everything, and then I auction it with No Reserve! Everything sells, and there is an eclectic mix of vintage TRS-80 computer items mixed in what whatever else people donate. In previous years that has included hampers, Tim Tams, test equipment, LCD flat-screen monitors. There is always lots of computers and accessories. Last year alone there was Cocos, Model 4s, a Model III, an MC-10, and at least 4 different printers!

THE TRIVIA SHOW!

At 8pm I will be running the trivia game. Like last year we have a short game for the ladies of 4 rounds, and then the guys have a very hard 6 rounds. At the same time the audience will be playing a special 'pub style' trivia game. Everyone gets a game sheet so bring along your pen or pencil and join in the fun!

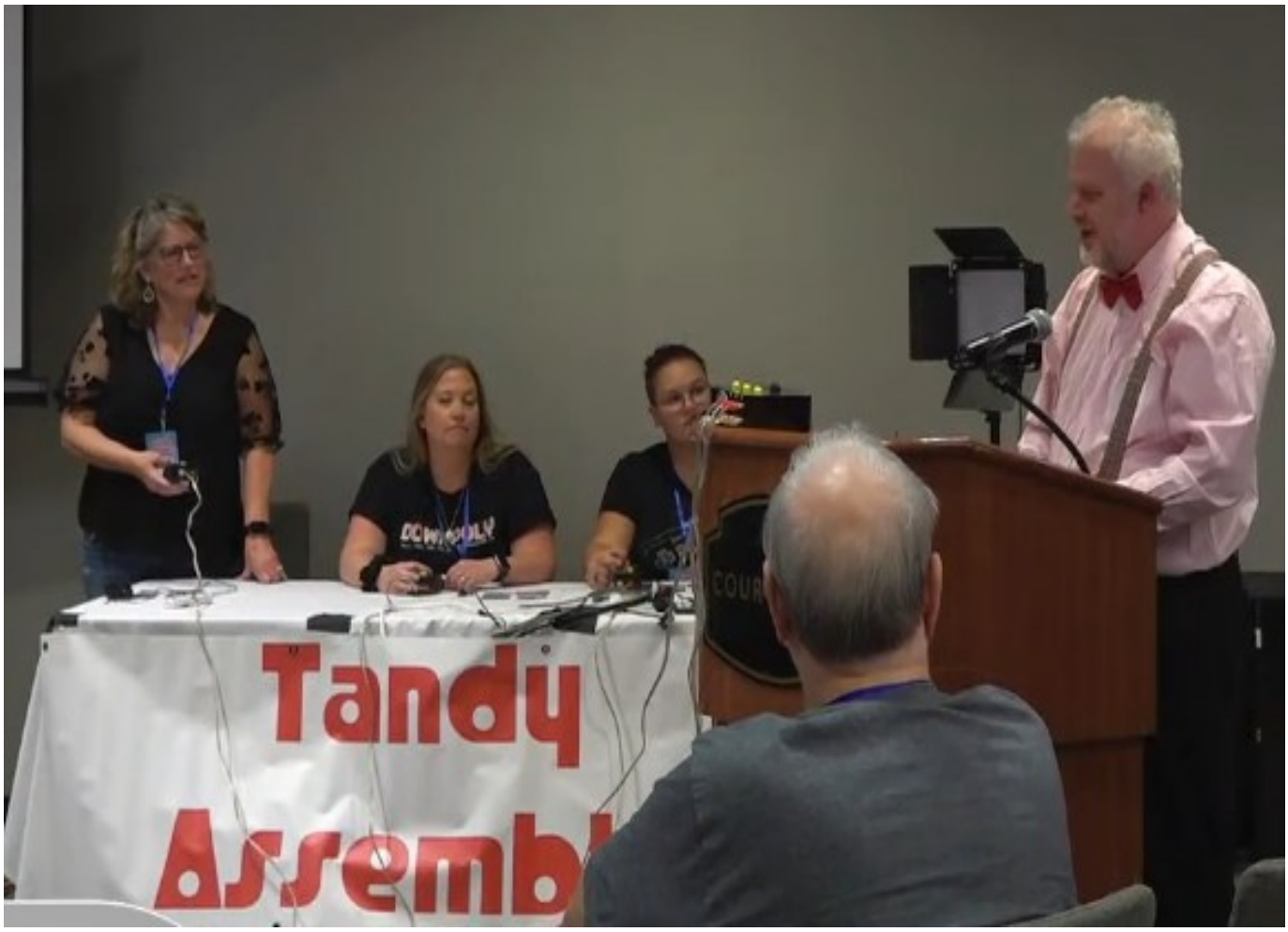


Photo: the ladies game was a big success in 2025 so I'm bringing it back



PHOTO: LAST YEAR'S TRIVIA GAME WAS HOTLY CONTESTED

If **you** would like to be a contestant on the trivia game it's not to late to reach out to me and I'll put you up there where fame or infamy await!

See you all at the show!

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.





Universal Backpack Drive

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

Choose Options

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

$65.00

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



As always, of course, I'm on the lookout for any software for the U.K.'s **Aculab 'Floppy Tape'**. There were versions specifically 'zapped', (by Laurie Shields), for use on the AFT, of Electric Pencil, Scripsit, Zen, Level III BASIC and VisiCalc. There was also the **enhanced BASIC, XBAS** (from Aculab). It's so disappointing to think that they have all been lost!

Please email me, at my usual address, if you can help.

I HAVE HAD A MASSIVE COLLECTION OF MAINLY MODEL 1 GOODIES, DONATED TO THE SITE, DUE TO A BEREAVEMENT, SPECIFICALLY REQUESTED TO HELP WITH THE RUNNING COSTS OF THE U.K.'S SITE AND THE PRODUCTION OF TRS8BIT.

IF THERE IS ANYONE IN THE U.K., (OR WITHIN REASONABLE TRAVELLING DISTANCE), WHO WOULD CARE TO MAKE AN OFFER FOR THE 'LOT', (TO SAVE ME ENDLESS HOURS ON EBAY), PLEASE CONTACT ME AT THE USUAL EMAIL ADDRESS,

dustym@fabsitesuk.com

YOU WILL NEED A VAN OR ESTATE CAR! THERE IS RATHER A LOT!



HERE'S A LIST OF THE ITEMS...

1 CGP115 printer, complete, boxed, spare pens
1 ribbon inker
6 boxes 5.25 disks
2 5.25 drives with power supply
1 Epsom TRS-80 printer interface
1 E.I. keyboard connector (no top)
1 M1 printer interface (for keyboard)
1 "Audio Amp"
1 M1 power brick
various cassettes
1 EPROM burner
1 3.5 drive
1 "New Clock-80"
1 M1 TV interface

1 M1 E.I. with power and keyboard connector
2 M1 boards, spares
1 M1 16K L2 keyboard
1 FreHD complete with Quinnterface and power supply
1 M1 printer interface (for keyboard)
1 M1 joystick
1 MT25c "Interfaker"
1 RS TR-10 serial printer
1 RS Mini-cassette recorder with tapes
various ribbons
1 M3 serial I/F
2 M1 light pens, complete
1 Tandy WP2 word processor

2 TRS-80 disk drives, cased, with power supplies
2 Tandy LP VII printers
1 Tandy printer controller
1 40-way male extender
1 CRT-80 cassette recorder
1 M1 Expansion Interface

1 Aculab 'Data buffer'
2 M1 Expansion Interfaces
1 M1 16K L2
1 M1 48K L2
2 M1 keyboard printer interfaces
1 Radio Shack CRT55 cassette recorder

1 Microline-80 printer with manual
2 Model 1's and an Expansion Interface
2 single sided disk drives
1 double sided disk drive
1 Co-Co joystick
various cables

1 Tandy Model 102 portable computer
Tandy 1100FD P.C. laptop. Complete with case & power supply disks and manuals
Amstrad NC200 Notebook
2 40T 5.25 drives
1 large box of cassette software
4 boxes of TRS-80 books, magazines, software manuals etc.
2 40T 5.25 drives
TRS-80 DMP printer
TRS-80 Acoustic modem
"Steepletone" SCR803 cassette recorder
RS 256K printer buffer
TRS-80 printer controller PTC-64
1 TRS-80 "No. 1" disk drive complete
2 bare 5.25 disk drives
1 Model 1 original style, monitor

I did try a couple of the Model 1's and both powered up OK with the monitor, however, most of the items are getting on for 45+ years old, so there's no warranty !!


















Microline 80
Matrix Printer
Operation Manual

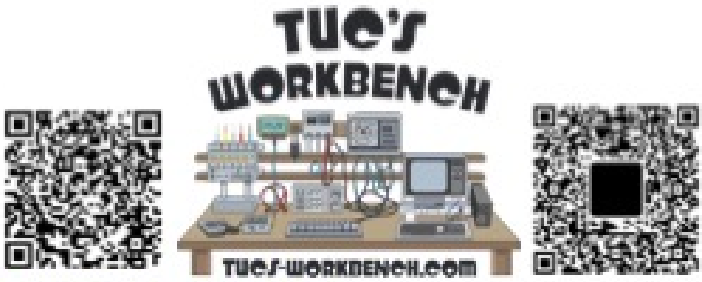

ENCYCLOPEDIA FOR THE TRS-80*
ENCYCLOPEDIA FOR THE TRS-80*
ENCYCLOPEDIA FOR THE TRS-80*
subLOGIC
T80-FSI FLIGHT SIMULATOR
BLACK JACK
HAUNTED HOUSE
STRIKE FORCE
EDITOR/ASSEMBLER
EDITOR/ASSEMBLER
ATLANTIAN ODYSSEY
Acorn Software Products Inc
BASIC COMPUTER GAMES
TRSDOS 2.3 DECODED & OTHER MYSTERIES


Radio Shack
Lewis Rosenfelder
BASIC FASTER AND BETTER
& OTHER MYSTERIES
BASIC
TRSDOS 2.3 DECODED
& OTHER MYSTERIES
80micro
SPECIAL ANNIVERSARY ISSUE
Learning TRS-80 BASIC
Using the TRS-80 in Your Home
EPSON PRINTER Users Handbook
LEARNING Level II
BASIC COMPUTER GAMES


Radio Shack
DMP 200


Microfiles
Double Density
Disk System Owner's Manual
HMRS: The Beatles
HMRS: The Hound of the Baskervilles
HMRS: Moby Dick
IN-MEMORY INFORMATION SYSTEM
EDITOR/ASSEMBLER
RS-232C COMMUNICATIONS SOFTWARE
STATISTICAL ANALYSIS
Radio Shack


Radio Shack
TRS-80
COMPUTER PRODUCTS
Cat. No. 26-1269
PTC-64 PRINTER CONTROLLER
Custom Manufactured in USA by Radio Shack, A Division of Tandy Corporation
Dysan
TEN 5.25 INCH DISKETTES
DOUBLE SIDED - DOUBLE DENSITY - 96 TPI
BASF
10 Diskettes 5.25" 2S/HD Extra
10+1



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

"Our First TRS8BIT Ad" Featured Items

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

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

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

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

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

This transcript is from Episode 8 of the Floppy Days Podcast

(https://www.floppydays.com), whose audio you can listen to at: https://floppydays.libsyn.com/floppy-days-episode-8-the-trs-80-model-i-part-i

Patreon: https://www.patreon.com/FloppyDays
Sponsors:
8-Bit Classics - https://www.8bitclassics.com/?ref=Floppydays
Arcade Shopper - https://www.arcadeshopper.com/
FutureVision Research - https://www.fvresearch.com/
Tuc's Workbench - https://shop.tucs-workbench.com/

Floppy Days Episode 8 - TRS-80 Model I History (Part 1) and Interview with

David & Theresa Welsh (Part 1), authors of "Priming the Pump"

Memories

Randy Kindig

When I was a poor college student at Purdue University, I had a TI58C programmable calculator. And that essentially was my personal computer with its storage registers, 480 program steps, and permanent memory. Although it wasn't programmed in BASIC, its machine language-like programming language was easy enough to use and was very useful in my engineering curriculum.

However, by the time I was a senior in 1978, the TRS-80 Model I had started appearing in the local Radio Shack stores. With a monitor, computer, keyboard, and cassette recorder, it was ready to use right out of the box. It's easy to learn and use BASIC programming language, 4K of RAM and relatively low price hooked me from the first time I saw it.

I was in engineering, not computer science, so I wasn't really that aware of what was going on in the computer industry. I knew about the Apple II and the Commodore PET, the other most common home computers available at the time, but I had only read about them and had never seen either one in person. I also was aware that the Apple II was out of my price range.

I knew that I wanted a TRS-80.

I don't recall talking about it a lot, but I must have discussed it with my wife more than I remember. When I graduated from college, my wife completely surprised me and brought home a TRS-80 Model I with 4K of RAM and 4K of ROM. I remember opening that box, pulling out the parts, hooking them together, and spending hours upon hours getting to know the level one BASIC in the machine. My wife may have later regretted buying that computer for me as I spent a lot of time on it.

That computer inspired me to decide to go back to school to get my master's degree in computer science and embark me on an IT career with which I've been very happy and a computer hobby that I love.

It all started with this little computer.

Introduction

Randy Kindig

I decided to break up the episode on the TRS-80 Model I into two parts. I was able to get an interview with the authors of the book, “Priming the Pump: How TRS-80 Enthusiasts Helped Spark the PC “Revolution”, that being David and Teresa Welsh. They were absolutely awesome. I decided to break the interview and my TRS-80 history discussion into two parts to keep each episode to approximately an hour in length.

So, in this episode I will discuss the history of the TRS-80 Model I up to the time it was announced and then present the first half of the interview with David and Teresa. Next show I will discuss the history of the TRS-80 Model I starting with the time after it was announced and we'll then present the second half of the interview.

History of the TRS-80 Model I (Part 1)

Randy Kindig

Okay, let's jump into the history of the TRS-80 Model I. In particular, right now, I'm going to focus on history leading up to the machine as it was being developed.

Back when Steve Wozniak was demonstrating his Apple 1 to the Homebrew Computer Club, one member who watched with great interest was a young engineer from National Semiconductor named Steve Leininger, who had graduated from Purdue University with a master's in electrical engineering. A computer hobbyist, Leininger had bought an IMSAI, which was an early kit computer, and was moonlighting at the Byte Shop in Santa Clara.

In the spring of 1976, there were few representatives from Tandy Corporation that were visiting National Semiconductor to look at a new processor for the Radio Shack stores. They were there because of Don French, who worked at the Fort Worth, Texas headquarters, and who had been urging his superiors to consider selling a kit from which customers could build their own home computer. In fact, entirely on his own, French had begun quietly working on a computer product that could be sold by Radio Shack stores.

Steve Leininger was asked to meet with the men from Tandy. Leininger talked to his visitors about what they would need to build a computer. They were interested, asked questions, and as they were leaving, asked if there was a computer store anywhere nearby. Since Radio Shack was in the retail business, they wanted to see how computers were sold. Leininger gave them directions to the Byte Shop that he worked at, which was owned by Paul Terrell.

When the Tandy buyers went to the store that evening, they were surprised to see Leininger, who was working behind the counter. After checking out the merchandise and asking some more questions, one of the men asked Leininger if he would consider consulting on a computer kit project for Tandy.

Later, Tandy flew Leininger out to their facilities. John Roach, who was then vice president of manufacturing, made Leininger a job offer. The job would consist of designing and building a computer kit to be sold in the Radio Shack stores.

It was just what Don French had hoped for.

With Leininger on board, the kit product could take off.

They worked on kit concepts, trying to keep down the cost. Tandy management wanted a $200 computer, so every part had to be cheap. Originally, Leininger planned to use the 8080 microprocessor. Those plans changed when he learned about the Z80. The Z80 had the same instruction set as the 8080, but added a few more instructions. Leininger liked the Z80 for reasons that would allow him to build a cheaper computer.

The whole project just about died one day when a heavy package came in the mail to the engineers. It was an expensive digital clock kit that a customer had put together and then sent back. He had put it together completely wrong. The vice president in charge of buyers, who was not keen on the computer idea anyway, saw the bungled clock project and was afraid this would happen with the computer kit builders and he shouted, "That's it. There won't be any computer kit!”

They decided to try to design a finished product instead. Customers wouldn't have to solder anything. To sell this idea, Charles ran this idea by Charles Tandy.

They would need to build a prototype.

Alone in his room, Leininger put together what would be the first TRS-80. As he worked on his creation, Leininger constantly had to find ways to keep the cost down so that they could make Radio Shack type profits. That meant building it for about half of the final selling price. Despite these limitations, Leininger wanted to provide a monitor. Adding a monitor would add cost, but Leininger wanted a better display than you could get out of an ordinary television set, which displayed only 32 to 40 characters across. By modifying a TV and providing it as a monitor, they could get that up to 64 characters across.

The reason for the black and silver color of the final product was because the modified RCA TV it used already came in those colors, which RCA called Mercedes silver. Tandy even opted out of displaying lowercase letters on the monitor to save $1.99.

In the end, the TRS-80 cost less than $300 to make, including the monitor and tape recorder.

After demoing the computer to Lew Kornfeld, president of Radio Shack, and Charles Tandy, they only got the go-ahead to build 1,000 units. It seems they were a little unsure how well it would sell, so they wanted to start out cautiously. Later that week, they changed the number to 3,500 because they discovered they couldn't get a good enough deal on parts if they only built 1,000. It also turned out to be the exact number of how many Radio Shack stores they had at the time. Roach figured if they couldn't sell the computer to customers, the stores could use them at least.

They planned to supply BASIC and Leininger was using Palo Alto Tiny BASIC, which was public domain, for demonstration purposes. On their low budget buying Bill Gates BASIC already in use on the MITS Altair was not even under consideration. Leininger made his own fixes, including writing floating point, to the Palo Alto BASIC and that became known as the TRS-80 Level I BASIC. Leininger says that one of his innovations was the CLS (or clear screen) command which Microsoft later borrowed to use in its BASIC.

The final development cost of the TRS-80 was less than $150,000 according to John Roach.

Leininger was interviewed for the December 1981 issue of Popular Computing. Here's what he said about his work at Tandy. “I think I'm still proudest of the Model I because I did most of that by myself. None of

what we're doing today would ever have taken place if it hadn't been for that computer. And it was great doing a job that no one no one else had done before, that no one even thought possible. It was a real challenge, one of those once in a-lifetime happenings that I hope only happens once."

The TRS-80, TRS standing for Tandy Radio Shack, was announced to the world on Wednesday, August 3rd, 1977 at the Warwick Hotel in New York City. The New York Times, like most other general news vehicles of the day, didn't see anything newsworthy about the entry of a complete off-the-shelf affordable micro computer.

Interview with David and Teresa Welsh (Part 1)

Randy Kindig

Now we're going to present the first half of my interview with David and Theresa Welsh concerning their book, ***Priming the Pump***:

Tell us a little about your book. I already have a copy, but perhaps you could summarize what it's about and who the intended audience is.

David Welsh

The audience for the book is probably people who had some association with the TRS-80 back in the day and are interested in how it came to be what it was and everything that happened.

A lot of these other computers, such as the Apple, have been well documented—or at least many people have written about them.

Because you can't ignore the story of two guys in a garage making a fortune.

But in the case of the TRS-80, equally interesting things happened. But they were mostly behind the corporate veil of Radio Shack, so not many people heard about it.

We wanted to dig out that part of the story.

We were also in the business of writing and selling software during that period, in the early 80's, when the TRS-80 was, at one point, the most populous computer in the world.

So we combined those two themes in the book: where the TRS-80 came from, how it affected the people who fell in love with it, and some of the people who wrote software for it.

Theresa Welsh

I think we felt the audience would include people who had a TRS-80—a nostalgia audience—as well as people who are just interested in computer history. It really seemed to us that this was a neglected corner of computer history that people didn't know much about.

People who think IBM invented personal computing, or they know about Apple and assume those were the only pioneers. That kind of thing; to kind of set the record straight.

As far as I can tell, ours is really the only book…

I was just reviewing some of my ancient notes from the period of time I was working on it. Other people along the way would often say, "Oh, I've thought about writing a book," or, "I'm going to write a book." But to my knowledge, nobody besides us has actually written a book specifically about the TRS-80 and its niche in computer history.

Also, from the reviews we've gotten, which have been overwhelmingly positive, the people who like it the most are those in the nostalgia category—the people who had the computer. We've gotten lots of reviews

and emails from people saying things like, "It brought back such great memories," or, "I really enjoyed reading it." That kind of thing.

David Welsh

Even people whose dad had the computer. They remember that being something exciting that happened when they were a kid.

Randy Kindig

I'm definitely one of those people who falls into that first category. The TRS-80 Model I was my first computer, so I've got a bit of nostalgia for that machine, and I've got one in my office here.

I'm impressed by how much detail you went into in your book and I was just curious—how much research did you have to do to come up with all that information, and how long did it take you to put the this together?

Theresa Welsh

I have to say, I worked on this for a solid eight years. Again, I went back to my notes and see that they started in 1999. And, ended in, well, we didn't publish until 2007, so that's eight years.

Most of the interviews were done from 1999 to 2002. We worked on it; I had a full-time job until the end of 2005. What really let me finish up the book was losing my job. I suddenly had time to actually finish it up.

But we have audio files of the people we interviewed, so I could go back to what they said and listen as many times as I needed to and be accurate in any quotes that I had. So this was a long haul.

This was a long-term project. This was not something thrown together quickly.

It was the interviews, but also we had a great mass of documents and stuff left over from our business records. I had notes from day-to-day work. And then there were the old magazines to this day in my basement. And I spent part of today down cleaning out the basement because I have a shelf down there that's about ready to collapse onto my sewing machine and I had to do something about it.

And since you have the book, you know we've got a lot of old ads in there. I don't see how you would do a book like this —your memory begins to fade after a while, and you really do need to have records where you can check up: "let's see, when did this particular thing happen?" Go back to some document that would have a date on it, that kind of thing.

Randy Kindig

David, did you want to add something here?

David Welsh

Well, actually, when we started to get into this project, I had forgotten a lot about what went on back then and especially as to what the timeline was and what came first and second and so forth. And reading all the material and going back through stuff from that era, it all came back. And, things that weren't in the documents were memories of what was going on at the time but were triggered by the documents. And, "Oh, yeah, I remember that." So without that collection of old magazines and other things, it wouldn't have been easy to write the book.

We were heavily involved with the TRS-80 from about 1978, I think. We actually got the machine, and there was no software for it. And so I was interested in writing software for my own purposes. But then I had a friend who had a friend, and those two guys were interested in trying to make money off this. And that appealed to me, too. We didn't really know what we were going to do, but we thought we were going to do some kind of software for vertical applications for a particular industry or something to keep track of things that were interesting. Of interest to them that they could track on the computer.

In other words, sort of an early database operation.

And I thought that, well, in order to do that, we need documentation. And I hate; I like to write, but I don't like to type and go back and correct myself and everything. So I thought what we needed was a word processor. And I thought I would whip one up, which is what comes from being ignorant.

Sometimes you do amazing things because you don't know that you can't.

And by then I had mastered something of assembler. And in those early computers, they didn't really; they weren't very fast. So, if you tried to write something like a word processor in BASIC, it was a trial to do — especially since Microsoft had something called garbage collection. And garbage collection was a cleanup of the strings; that's the words that you type into the computer. And they were stored in strings. And there was a routine to collect all those strings together and put them in a particular area of memory. And every time that happened, the computer just stopped. And if you were doing relatively little with strings and you had games and things like that, it really wasn't much of a problem. But to write a word processor, you don't want to be typing along and have it suddenly stop.

So, I ended up doing it in assembler.

I had a printout that was about six feet tall or so. And it did inserts and it did deletes and it did some of the things you associate with word processing. I said, oh, gee, this is pretty easy. Maybe we should write a word processor for our first project. And, of course, by the time we were done writing a word processor that was acceptable for people who actually were doing important things, it was piles of printouts and code and much, much longer than I had anticipated. Because you have to accommodate so many different things and have it print out the way the person expected it to print out. And, not have a buggy routine where you get something you didn't expect.

And so that was the start of the word processor.

After thinking I would spend a couple weeks on this, after about a year or a year and a half or so of writing, I had something and I thought we could sell. And we went to the Boston Computer Show to sell it. And I was still working on parts of it the night before. And, actually, after we got to the show, I was very cautious until the first 10 people or so bought it that there would be a quick update. And, it was still not guaranteed to be error free.

But that's how we got started.

Randy Kindig

Can you tell me—maybe each of you—what your first encounter was with the TRS-80 and what that was like? First time you ever saw one and touched one?

Theresa Welsh

Yeah, going to the Radio Shack store and seeing them there. You know, like any customer walking in.

Or did you know about it before then?

David Welsh

Well, I occasionally picked up a computer magazine before that.

And I was interested in —we both were interested in — word processing.

Theresa Welsh

I was real interested in word processing since I was a writer.

David Welsh

There was a small computer store that started up in the suburbs of Detroit and there was I think one or two in Ann Arbor and we went to those places and looked at what they had. The first one was the one in Ann Arbor and they had an Altair hooked up to a teletype machine and you could play little games on it but it wasn't up to word processing standard and a little later we went to one in the Detroit area and the guy says "well, for that, you're going to need about $6,000 or so of equipment."

He says "I'm writing the best one. I'm in the process of writing the best word processor."

Theresa Welsh

I don't think you've ever heard about his product.

David Welsh

And we didn't have the $6,000, so we passed on that one.

And since I was a small child, I've been interested in electronics. So I had kind of the background on the hardware all through this period because I was reading *Popular Electronics* and things like that. But I didn't know much about programming.

I didn't want to build my own computer because it seemed like an expensive project. And, if you ended up with nothing, then it was a waste of money.

When the TRS-80 came out, here was everything in the box. They gave you the video and it was already set up to work with that. There was Tiny BASIC in it, and it had great expansion plans. And it was all for $600, which back in the 70s was a considerable amount of money, but not the back-breaking amounts that some of the other people were charging.

Theresa Welsh

It was a lot of money for us, but on the other hand it was really cheap compared to the word processing that was out there. You may remember the dedicated machines like the Lanier and the Wang. There were a couple of brands. They cost like $20,000 — $15,000 to $20,000. Your giant law office would buy them, or that was the kind of customer. But ordinary mortals couldn't afford anything like that.

So the Radio Shack machine sounded cheap, but it was still a lot of money for us. I was a little reluctant to show up the money.

David Welsh

I was a photographer at the time, and I thought I could probably write a program that would do sort of my accounting in the photography business. And I had sort of stood up in Radio Shack stores and banged on the keyboard and they had a wonderful manual for it. And I flipped through the manual, and I would try BASIC. It seemed to me like something I could master.

So, I told her, well, we're going to get this thing, and we'll never upgrade because it's expensive. And, of course, six months later, I broke that promise.

But I did write a program for Level I BASIC and 4K of memory which is what the computer provided which would kind of track my invoices for the month in the photography business. The only problem was if business expanded and I was in trouble because there was no way to expand it beyond what the 4K would support.

Randy Kindig

Until the 16K came out finally, right?

David Welsh

That's right. Well, I got the 16K upgrade.

Theresa Welsh

Well, the other thing, though, was, of course, originally the Model I was all uppercase. It wouldn't be any good for a writer. You can't turn in a manuscript in uppercase. You can't write a business letter in uppercase. I mean, it was one of those stupid things Radio Shack did, really, which they corrected, that originally you could buy a kit or you could buy something to fix it because that needed to be fixed right away.

David Welsh

But when I wrote Lazy Writer, I didn't have that upgrade and I had an uppercase only machine. But the secret was that the keyboard would store lowercase and uppercase. But you wouldn't see it on the screen. It would be on the upper case.

Theresa Welsh

That's right. When you printed it, you would see the lower case, assuming you'd done everything right. It really wasn't a very good fix to it.

The other problem, of course, was the output. As I said in the book, any good typist with an IBM Selectric would laugh at stuff that turned out of a TRS-80 onto a dot matrix printer. You didn't really have word processing until you had upper and lower case and you had the daisy wheel printers.

David Welsh

The dot matrix printer, the first one that came out, that was really decent and affordable, was the MX-80 from Epson. Everybody was buying those.

Theresa Welsh

Everybody had one of those. They were really nice little printers, but they just weren't good enough for any kind of professional work.

David Welsh

Well, people used them anyway. Because people who had computers earlier on, they had hooked up teletypes to the computer. As did we. A Model 15 teletype. And that was the output device for a while. It was fine for things like printing out listings of your programs, but it was all uppercase and not acceptable for business use in particular, but you could produce some text with it. So when MX-80 came out, it even had TRS-80 graphics built into it, which were pretty primitive, but nevertheless it let you do something. And it had upper and lower case, plus some modes to show bold.

Theresa Welsh

Remember it had double wide? Little tricks you could do with it. Nice.

David Welsh

Yeah. So that was a huge step up, even though somebody was used to their secretary's product on a Selectric typewriter would look at that and say “this is really primitive looking.” But we loved it. So that's what the word processor was initially written for. Then I talked the partners into buying a Qume printer, which was a daisy wheel printer with upper and lowercase and the potential for proportional spacing. And, we used that for quite a lot of time. And that did give a very professional looking output.

But you could not see that on the screen because even after, even if you got, the conversion kit for upper and lower case, it didn't show you what it was going to look like on the paper as you're used to now. It only showed you in upper, well, if you had the upper and lower case conversion, there was, as I remember, no descenders on the lowercase. It was a pretty awful look on the screen. But it did show you what was uppercase and lowercase. There were a lot of problems like that. If you're writing software that you hoped would be useful back in the day, there were many more things to overcome in just the deficiencies of the

equipment. The upper/lower case thing, we handled by having a mode in 'LazyWriter' that would hashmark all the uppercase stuff. You could switch that to that and see your screen reprinted with hash marks where all the uppercase characters were. So, you could at least check what you had made uppercase. But it obviously doesn't hold a candle to what we do today. Things like Microsoft Word and publishing programs and so on, you can get, finally, a very good representation of what you're going to get on the paper.

Randy Kindig

So you had a business for a while that revolved around the TRS-80. What exactly did you do in that business, and how long did that business run?

David Welsh

Well, we sold the word processor that finally got developed as Lazy Writer. And we wrote a bunch of add-ons for the word processor to help it do other things that people found useful. And we had a lawyer friend in Muskegon who said, "well, I would like something that would let me set up documents for legal purposes." And he had some things that he wanted to do, like, for instance, have a boilerplate original and be able to put in the names and the addresses and different things in there to make it a legal paper that he could take to court. So we wrote some modules for that and for other things as well. So we ended up with a very complete system that did a lot of what today's word processors do.

But you had to do a little thinking because we couldn't show all that stuff on the screen the way you can with a modern computer. In a modern computer, most of the high power that you're paying for in the processor and everything is going basically to the presentation on the screen and user interface. The things it actually does for you are relatively trivial and could be done by the TRS-80 or some slightly expanded version of it. But to do all the processing that goes on now so that you can see a picture on Skype that's an advertisement pop-up in full color. That's something that takes a lot of computing power, which we didn't have then.

Theresa Welsh

Yeah, and I'd like to point out a really big change. I mean, we were obsessed with getting good output on paper because the object in those days was getting it to the paper. But today, the screen is the paper. Reading your book on the screen of your TRS-80 would have been unthinkable. It wasn't something that you could do. And then the portability factor, too, that you can carry your device around with you. I was just going through my old files, and I popped up a picture of an old ad that said, make your TRS-80 portable. And it showed this guy with his giant suitcase, you know, which has disk drives. And I laughed. You know, who would want to do that? Imagine lugging around your Model I.

David Welsh

People did, though.

Theresa Welsh

But it turns out, I mean, of course, we had, you remember the Osborne, the first transportable computer, that it turns out people didn't really take anywhere because it was too big and heavy, too. But not so today. You've got your Kindle; I have it in my purse and I pull it out.

David Welsh

Your iPad or your phone.

Theresa Welsh

Yeah, your iPad, even your phone. You could, you know, be reading something. So that's an enormous change, just a really big change. And now that I'm retired, I do feel I'm the old generation now. The young generation; it's just normal for them to read a book on a device and not worry about how their output's going to look on paper because it's never going to paper.

David Welsh

The original TRS-80 came with 4K of memory. That's 4,000 bytes of memory. And you could expand it to 48K of memory. And there was, inside the machine, there was a ROM area that had BASIC, and that was another 16K or 8K. Well, yeah, the ROM was 4K, and then it was 16K altogether with space in there for some other things. And that came to a grand total of 64K, which was pretty much the norm for CP/M computers and other serious machines in those days. But if you took that 48K today, it wouldn't be enough memory to display a picture on your phone as a camera that takes a three megapixel picture. So, 48K would just do a little corner of it. And there wouldn't be any room for anything else.

So that's an enormous change. And in fact, people in technology back in those days all expected there would be growth in processing and growth in memory so that things would get cheaper and there would be more memory and so forth. But even in technology, there was a lot of under estimation of how fast things were going. There was a period in the 80s and 90s in particular where any estimate of what we were going to have two years from now was probably wrong.

Randy Kindig

What are your thoughts concerning the—I'm sure you've heard the nickname **"Trash-80"** that was given to the TRS-80 Model I. What are your thoughts concerning that rather unfortunate nickname?

David Welsh

Well, we called it Trash 80 a good deal of the time. And it was unfair in some respects. This was an era — preceding the TRS-80 and the microcomputer what people did with computers is they put them in giant air-conditioned rooms with special flooring and special electrical circuits just so that they would function well. And the TRS-80, people bought off the shelf. They took it and set it in the middle of their factory and they ran business programs on it. And then they were surprised when the thing rebooted. All this electrical noise going on next to them and everything.

Theresa Welsh

But I think it was mostly an affectionate term, too. It was an obvious way to pronounce "TRS"—to turn it into "trash." I think people really loved these computers.

Radio Shack got sort of grandiose. In a way, it was making fun of RadioShack. The plastic, kind of cheap looking, case it was in and then it turns out Radio Shack called that Mercedes Silver. Which I thought was kind of funny. But I think it was an affectionate term to a great extent and a kind of parody of Radio Shack's attempt to make it sound highfalutin when it wasn't. But it was built of kind of cheap looking materials.

David Welsh

Well, another factor was that the Radio Shack brass brought this computer out. They had no idea how well it was going to sell. Infact the story is that their initial run was just enough to supply one for each store so that if the thing bombed at least maybe they could make some use of it in the store.

Theresa Welsh

Yeah, we heard that story mainly from Don French who was one of the guys credited with being the father of the TRS-80 or whatever. Don French and Steve Leininger. Don French tried mightily to convince his bosses that people were going to buy this machine. They were going to get sales. They really didn't think they were going to sell many of them. The story was they had 250,000 orders backed up at the end of the first year that they couldn't fill because they hadn't built enough of them. The desire for a computer, it turned out, was really huge and greatly underestimated. I mean, maybe they would have actually made the machine a little more elegant or put a little better materials into it if they'd known how many they were going to sell.

David Welsh

Well, also, it rushed them into things like disk drives and stuff and disk operating system. The first version of which came out just flat out didn't work. And, all of that, they fairly quickly fixed their mistakes. And they did solve most of those problems by the end of the Model I era.

Which didn't last too long because they had to do something about radio emissions. The FCC got on everybody who was making the computer and saying you're emitting too much radio interference, and you have to do something about that, which was one of the main reasons why the Model I ceased abruptly, and we had the Model III instead because they could solve that problem with the Model III. It was all in a closed case, et cetera, whereas the Model I had the expansion interface and had wires going back to the expansion interface that were exposed. And this was very hard to guarantee that it wouldn't emit radio signals that were going to interfere with something else.

But in the short life of the TRS-80, the original TRS-80, they did fix most of the problems fairly quickly. It's just that everyone was impatient to get the latest and greatest and they just didn't anticipate how much pressure there was going to be to do that. So they would have done better had they had a better look at the horizon and what people were telling them.

In fact, one of the things I find amusing is that just about everybody we talked to who had talked to the TRS-80 people — or the Radio Shack people — early on had told them that they were going to sell a lot of these things; at least 50,000.

Theresa Welsh

Yeah, another one who told them that was Stan Veit. I emailed with him. He owned one of the early computer stores in New York City, and he had that story about being on the airplane with Charles Tandy. Charles Tandy took a big personal interest in the machine, and Stan Veit told him they were going to sell in big numbers.

But somehow… that didn't sink in.

Don French told them – a number of people told them this was going to be a big success. People were going to want this. But they somehow didn't think so, and they thought, well, we'll build enough so every store can have one. And then, you know, people were ordering them and had to wait a long time to get one. It was really a big runaway success for Radio Shack.

David Welsh

The skeptics inside Radio Shack pointed out that Radio Shack hadn't sold hardly any products for $600. Their price range was a lot less than that.

Theresa Welsh

Yeah, that was one of their ways they were skeptical. They were known as — they sold cheap products. You know — the battery of the month club and all that.

David Welsh

One guy said, "well, the only thing we've sold 50,000 of in a year is batteries."

Theresa Welsh

So, they just didn't think they could.

It's sort of interesting that they built the machine at all. But then they almost didn't do that either.

When we talked with Leininger, he literally built the thing by himself. I mean, this was a real low-budget thing for them. And then they wanted him to do other work. He was supposed to build the TRS-80, but also handle customer complaints and work on other stuff. And he finally told them, look, I got to work on this full time to get it done. And they sent him off to, what was it, an old saddle factory or someplace. They found him some space, said, “okay, you go over there and build the computer.” So, it was interesting how the thing got built.

And it was literally — first it was at the prodding of Don French, who told him he was into the whole hobbyist thing. And he said, you know, we've got to get into this computer business. This is going to go somewhere. This is going to be big, so they did it in a half-hearted way. Well, okay, we'll hire somebody. We'll have them put something together and see what comes of it. And they hired Leininger from Silicon Valley. He had been a member of the Homebrew Computer Club, that whole crowd, and he and his wife went to Texas to work on this computer project and several times they almost scrapped the whole thing. But it was not one of their big priorities although Charles Tandy did take an interest in it.

David Welsh

But, well, everybody did when they started getting thousands of phone calls and saying where is this darn TRS-80? And, when they first introduced it, it didn't take too long because they were sort of in an ideal



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



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



Tandy Corporation, TRS-80 Division

Name Radio Shack _____

Address _____

City _____ State _____

Payment enclosed/Check one ☐

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



E.T. FONEHUME

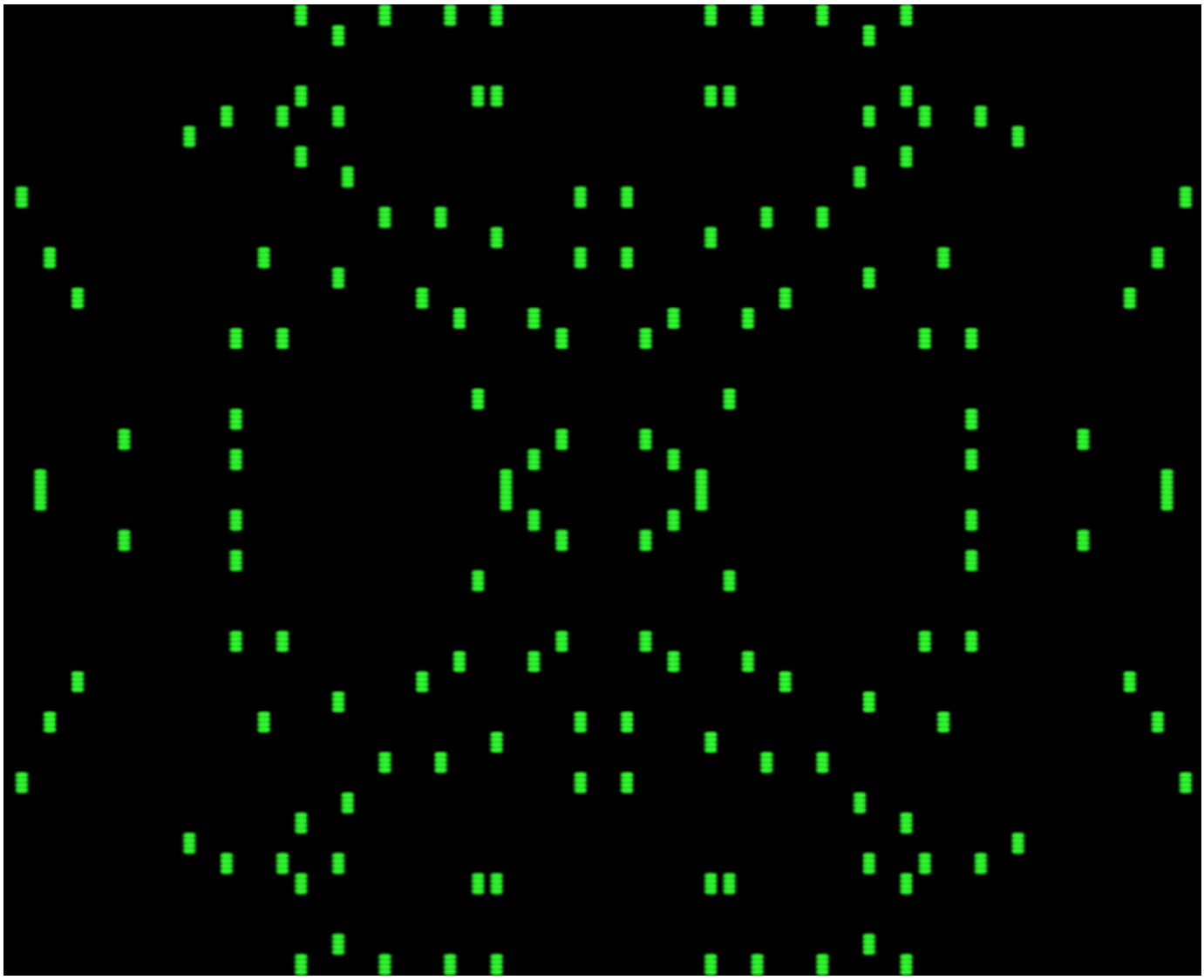
HI DUSTY. I WAS SURFING ON 'YOUTUBE' THE OTHER DAY AND CAME ACROSS A RATHER NICE BASIC PROGRAM (FOR THE BBC MODEL B) TO DISPLAY, ON THE SCREEN, IN COLOUR, A KALEIDOSCOPE. SUPER IN FULL COLOUR BUT I THOUGHT, WHAT WOULD IT LOOK LIKE ON 'TRS80GP' RUNNING A MODEL 1? I MESSED AROUND FOR A WHILE AND I HAD A THOUGHT; FOLLOWING IN IRA'S FOOTSTEPS, I'LL ASK 'GENISIS' WHAT HE (SORRY, I ALWAYS THINK OF HIM AS PAL), THOUGHT OF THE IDEA. IT TURNED OUT HE WAS MOST ENTHUSIASTIC!

HE OFFERED TO WRITE A L2 BASIC PROGRAM, HAVING TOLD HIM IT WAS TO RUN ON A TRS-80 MODEL 1. IT WAS PRODUCED IN SECONDS AND CREATED A MOST SATISFACTORY KALEIDOSCOPE.

HE THEN SUGGESTED THAT INSTEAD OF JUST CREATING ONE IMAGE THE PROGRAM COULD BE IMPROVED! WOULD I LIKE TO SEE A PERPETUAL RE-GENERATING IMAGE HE ASKED? THIS HAS NOT OCCURRED TO ME SO I REPLIED, YES PLEASE! THIS TURNED OUT TO BE MOST IMPRESSIVE WHEN YOU THINK OF THE L2 M1 SPEED AND GRAPHIC RESTRICTIONS. RUNNING ON TRS80GP WITH THE 'TURBO' OPTION, IT NEARLY APPROACHES IMPRESSIVE!

AS PER USUAL. I HAD THE URGE TO CHANGE THE PROGRAM INTO A ONE-LINER, AS SHOWN BELOW. PLEASE REMEMBER, YOU WILL NEED TO USE THE EDIT 'X' COMMAND TO COMPLETE THE LINE! IT'S RIGHT UP THERE WITH THE LONGEST OF THEM!

```
0 CLS:RANDOM:DEFINTA-Z:L=40:DIMX(L),Y(L):P=0:FORZ=0TO1STEP0:X=RN
D(64):Y=RND(24):V=X(P):W=Y(P):RESET(V,W):RESET(127-V,W):RESET(V,
47-W):RESET(127-V,47-W):SET(X,Y):SET(127-X,Y):SET(X,47-Y):SET(12
7-X,47-Y):X(P)=X:Y(P)=Y:P=P+1:IFP>=LTHENP=0:NEXTELSENEXT
```





HAPPY 49TH BIRTHDAY TO THE MODEL 1 (AUGUST 1977)


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


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

XII. Basque Tournament of TRS-80

The last 28th of July, Tuesday, the Euskal Retro organized the XII. Basque Tournament of TRS-80. We met 5 players in Derio, in my own house because I have a sprained ankle and it was more comfortable for me.

It was a funny afternoon funny as always we gather to play these classic and old consoles and computers . We started playing TRS-80, but after playing one hour it, we played also NES (Nintendo Entertainment System), Master System, PS 2 and another old console; Schmid TVG 2000. Although the image is quite bad because my television is PAL, we were able to enjoy the TRS-80 Downland.

In the end of the tournament the player who got more points was Egoitz, but Daniel and the rest of players played well and enjoyed a lot. We will organize probably the next tournament in February of 2027 and we hope to have new games like Dragon Fire or Robot Battle, I have to buy new ones in Ebay to change a bit the list of games of our tournaments.

Here you have the results:

Monster Maze
1-Egoitz Campo 4 points (1,335)
2-Daniel Barrientos 3 points (880)
3-Arrate Elorrieta 2 points (240)
4-Lander Campo 1 point (110)

Canyon Climber
1-Egoitz Campo 5 points (19.000)
2-Daniel Barrientos 4 points (7.200)
3-Jon Proaño 3 points (2.600)
4-Lander Campo 2 points (1.200)
5-Arrate Elorrieta 1 point (1.200)

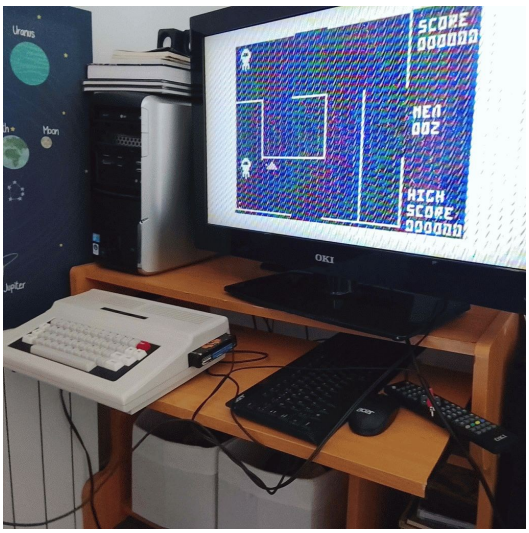
Downland
1-Egoitz Campo 3 points (2.600)
2-Daniel Barrientos 2 points (900)
3-Arrate Elorrieta 1 point (880)

Classification of the XII.Basque Tournament of TRS-80

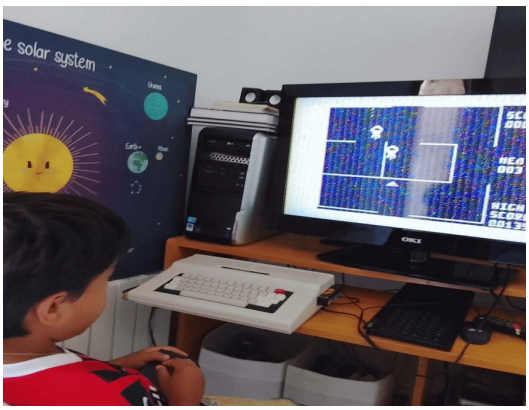
1-Egoitz Campo 12 points (Basque Country)
2-Daniel Barrientos 9 points (Basque Country)
3-Arrate Elorrieta 4 points (Basque Country)
4-Lander Campo 3 points (Basque Country)
5-Jo Prosño 3 points (Basque Country)



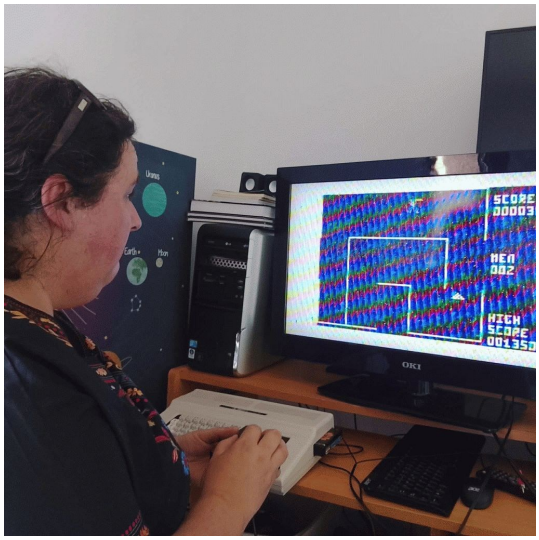

The solar system

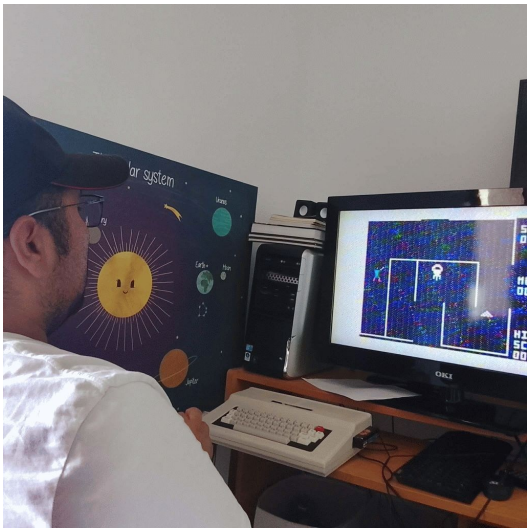

SCORE
000000
MEN
002
HIGH
SCORE
000000


The solar system

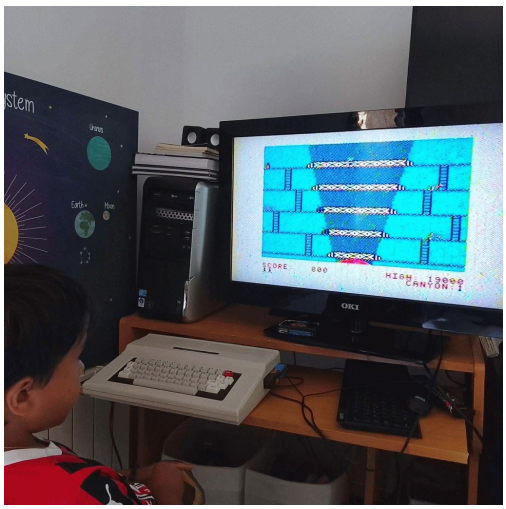

solar system


solar system


SCORE
00003
MEN
002
HIGH
SCORE



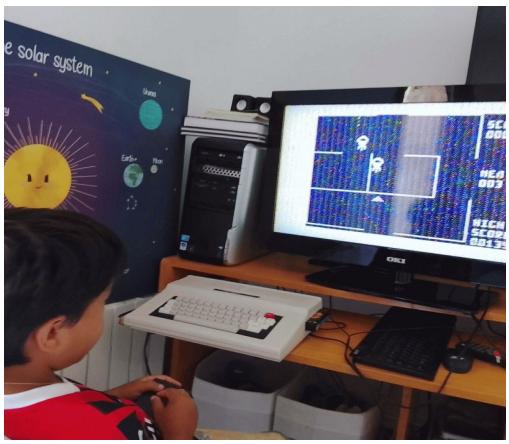














LEVEL 2.5 (TWOHAF) BASIC ALLOWS CASSETTE AND ACULAB FLOPPY TAPE USERS MANY OF THE TANDY DISK BASIC FEATURES, NAMELY -

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

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

WWW.TRS-80.ORG.UK



Vince Otten

Part 7 - Double trouble

I know exactly who to blame. It's all Kenneth Iverson's fault.
He's the one who created APL, the weirdest-looking programming language I've ever seen.

```
4 x -/ ÷ ¯1 + 2 x ι 3190
```

What does this APL program do? It approximates the value of Pi using 3,190 fractions in this series:

$$PI = 4 \times \left(\frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots + \ldots \right)$$

I explain how that one line of APL code does all this in the September 2025 issue of TRS8bit. But why am I bringing up APL in a series on Assembly Language Programming? It's all because of Double Precision arithmetic.
You see, the APL-80 implementation for the TRS-80 Model I does *all* its math in 8-byte Double Precision. Nevertheless, it manages to return a result in ~3:11 (that's minutes and seconds) on an 8-bit micro running at 1.7 MHz! I wondered how that speed compared with Level II BASIC, optimized as much as I could for speed.

```
00/00/00 00:00:43              00/00/00 00:06:30
 3.141279173973749
5 PRINTTIME$,
10 DEFDBLA-Z:S%=3190:DIMA(S%):FORI%=1TOS%:A(I%)=CDBL(I%):NEXT
20 FORI%=1TOS%:A(I%)=2*A(I%):NEXT
30 FORI%=1TOS%:A(I%)=A(I%)-1:NEXT
40 FORI%=1TOS%:A(I%)=1/A(I%):NEXT
50 F%=0:T=A(1)
100 FORI%=2TOS%
110 IFF%THENT=T+A(I%)ELSET=T-A(I%)
115 F%=NOTF%
120 NEXT
200 T=T*4:PRINTTIME$,T:LIST-300
300 END
READY
>_
```

That works out to ~5:47, or about 2.5 minutes slower. So, even though APL is an interpreted language just like TRS-80 BASIC, it's capable of remarkable speed.
Now, if you're math-savvy, you'll notice that the BASIC code does a lot of repetitious work. It fills the 3190-element with double-precision integers from 1..3190; then it doubles each element; then it subtracts 1 from each element in order to get a list of 3190 odd numbers; then it replaces each element with its inverse (1/x); and finally it alternately adds or subtracts each successive element before multiplying that total by 4 to get our final approximation to PI. We could write a much faster BASIC program by avoiding *any* array work and simply writing code like:

```
5 FL%= -1
10 T#= 1
20 FOR I%= 2 TO 3190: X%= I% * 2 – 1: IF FL% THEN T#= T# + 1 / X% ELSE T#= T# - 1 / X%
30 FL%= NOT FL%
40 NEXT I%
```

(removing all the spaces for speed, of course). So why did I choose to write it in the first way? …because that's how APL did it, and I wanted to compared execution timing.
Then I decided to compare APL with a true compiled language, one which was designed to be as fast as the assembly language code it was meant to replace: FORTRAN. Here's that program:

```
C
C     PI/FOR -- CALCULATE PI FROM APL SOURCE
C
      DOUBLE PRECISION A(3190), T
      INTEGER I
      LOGICAL FLAG
C
      DO 10 I= 1, 3190
   10       A(I)= I
      DO 20 I= 1, 3190
   20       A(I)= 2. * A(I)
      DO 30 I= 1, 3190
   30       A(I)= A(I) - 1.
      DO 40 I= 1, 3190
   40       A(I)= 1. / A(I)
      FLAG= .FALSE.
      T= A(1)
```

```
      DO 120 I= 2, 3190
C       "IF NOT FLAG THEN"
          IF (FLAG) GO TO 110
            T= T - A(I)
            GO TO 115
C       "ELSE"
  110       T= T + A(I)
C       "ENDIF"
  115     FLAG= .NOT. FLAG
  120 CONTINUE
C
      T= T * 4.
      WRITE( 5, 999) T
  999 FORMAT (G24.14)
      END
```

If you're unfamiliar with FORTRAN IV code, you'll still see that it looks remarkably similar to BASIC. Here's some of the less familiar elements:

FORTRAN	***BASIC***
C	REM
DOUBLE PRECISION A(3190)	DIM A#(3190)
INTEGER I	I%
LOGICAL FLAG	F%
DO 10 I= 1, 3190	FOR I%= 1 TO 3190
	(the NEXT can be found at "line 10")
CONTINUE	"NEXT"
FLAG= .FALSE.	F%= 0
FLAG= .NOT. FLAG	F%= NOT F%
WRITE (5, 999) T	*Sort of like*
	PRINT USING F999$; T
999 FORMAT (G24.14)	F999$= "##########.###############"

Surely FORTRAN must be *much* faster than an interpreted language like APL, right? When I compiled this with Microsoft's Fortran-80 and ran the resulting code, it came in at…3:11. Exactly the same speed! APL-80 is a remarkably fast and complete implementation of the original language.

But this made me wonder—and here we finally arrive at a relevant application for this series on "The Power of the ROM"—how fast would it be in Assembly Language? Like the FORTRAN code, I'd have to fill a Double Precision array with the numbers 1..3190, then double them, then subtract 1, then divide 1 by each of them, then alternately add or subtract, and finally multiply them by 4. Here's a real (pun intended) application for all 4 elements of Double Precision arithmetic, as well as conversion from integer to D.P. and from D.P. to a string that we can print out.

I plan to optimize my Assembly code as much as possible *without compromising the algorithm.* That is, I *will* fill, multiply, and alter the entire array. However, unlike APL, BASIC, or FORTRAN, I'll do integer arithmetic as much as possible before converting to Double Precision. (If I'd done the same in BASIC or FORTRAN I'd have to create a duplicate array of Integer size.) As well, I used the wisdom of the Internet—particularly, that of George Phillips—to make my Z80 code as fast as possible. Here's the working result.

```
MAIN    EQU     $
        CALL    INIT
        CALL    I00CRM

        CALL    CURSOF
```

"INIT" prepares our program to use ROM calls by clearing any errors and turning off BASIC tracing. "I00CRM" initializes all the vectors necessary to use ROM calls, especially arithmetic routines. Both subroutines were introduced in the June 2025 installment of "Power of the ROM". "CURSOF" simply turns off the screen cursor, and it was discussed in the last issue. We're going to replicate the FORTRAN version of the PI approximation code. That program first declares some variables, so here's our "data declarations":

```
;
;     ** DATA DIVISION **
;
;     * SYSTEM DATA *
;
WRA1    EQU     4121H       ; WORKING REGISTER 1
                            ; (INT & S.P. BASE ADDR)
WRA1DP  EQU     411DH       ; WORKING REGISTER 1A --
                            ; (DOUBLE-PRECISION)
WRA2DP  EQU     4127H       ; WORKING REGISTER 2 --
                            ; (DOUBLE-PRECISION)
NTF1    EQU     40AFH       ; "NUMBER TYPE FLAG" 1
```

We first talked about "WRA1" in the September issue of TRS8bit. It's the primary "accumulator" for the ROM's Integer and Single Precision math operations. Now we're introducing the Double Precision counterparts, "WRA1DP" and "WRA2DP". Here's how James Farvour diagrams WRA1 in his *Microsoft BASIC Decoded.*

Address	Integer	Single Precision	Double Precision
411D			LSB
411E			NMSB
411F			NMSB
4120			NMSB
4121	LSB	LSB	NMSB
4122	MSB	NMSB	NMSB
4123		MSB	MSB
4124		Exponent	Exponent

Where:
LSB = Least significant byte
NMSB = Next most significant byte
MSB = Most significant byte

WRA2 has an identical format.

Figure 1.7: Working Register Area layout.

Although Integer and Single Precision numbers start at the same address, 4121H, Double Precision starts 4 bytes "earlier" at 411DH. Also note that although Farvour labels 411DH (and 4121H) as the "LSB" or "lowest significant byte", it's actually where the *most* significant portion of a floating-point number (Single or Double Precision) is stored. (4121H remains the true "least significant byte" for Integers.) This also means that a Double Precision number with a mantissa of only 6 digits will have exactly the same representation as a Single Precision number.

```
;
;    * USER DATA *
;
ARRAYI      DEFS        8*3191          ; DIM I(3191)
```

The array A(I) was "DIMENSION"ed in our FORTAN code as A(3190), but here we allocate 3191 elements, simply because we start with element 0 (even though we only use subscripts 1..3190).

```
DP01    DEFB    0           ; DOUBLE
        DEFB    0           ;    PRECISION
        DEFB    0           ;        1.0
        DEFB    0
        DEFB    0
        DEFB    0
        DEFB    0
        DEFB    129

DP04    DEFB    0           ; DOUBLE
        DEFB    0           ;    PRECISION
        DEFB    0           ;        4.0
        DEFB    0
        DEFB    0
        DEFB    0
        DEFB    0
        DEFB    131
```

Here we also have a couple of Double Precision constants, DP01 and DP04. How did I come up with their memory representation? You may remember the BASIC program I used to display Single Precision values in the September issue. Here's my Double-Precision version:

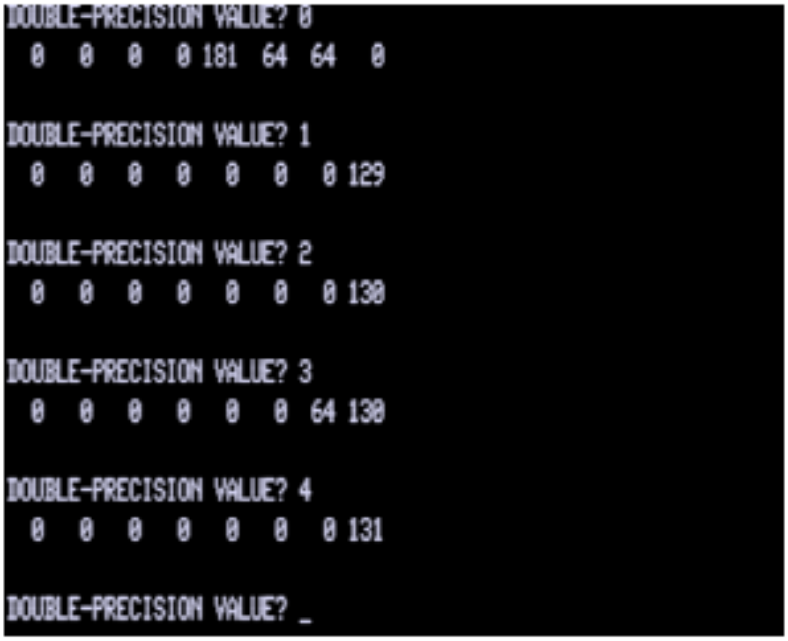
```
10 '
    SHOWDUBL/BAS -- "SHOW DOUBLE": DISPLAY
            DOUBLE-PRECISION BYTES

20 DP#= 0.0D0
30 AD%= VARPTR( DP#)
40 '
    INPUT VALUE TO DISPLAY

50 INPUT "DOUBLE-PRECISION VALUE"; DP#
60 '
    DISPLAY ALL 8 BYTES

70 FOR B%= 0 TO 7
80    PRINT USING "### "; PEEK( AD% + B%);
90    NEXT B%
100 PRINT: PRINT
110 GOTO 50
```

And here's what it looks like in operation:



Finally, our last two variables:

```
FLAG    DEFB    0           ; "FLAG"
TOTAL   DEFS    8           ; "TOTAL"
```

Now on to the actual code. First, we fill A() with the numbers 1..3190.

```
        LD      BC,3190     ; "DO I= 1, 3190 A(I)= I"
        LD      DE,1        ; I= 1
        LD      HL,ARRAYI   ; A()
LP0010  EQU     $           ; REPEAT:
        LD      A,8             ; SIZE OF A() ELEMENT
        ADD     A,L             ; HL += A
        LD      L,A             ;      (CODE FROM
        JR      NC,LPE010       ;     GEORGE PHILLIPS
        INC     H               ;     "ADVZ80")
LPE010  EQU     $               ; HL= &A(I)
        LD      (HL),E          ; A(I)=
        INC     HL
        LD      (HL),D          ;    I (DE COUNTER)
        DEC     HL
        INC     DE              ; I++
        DEC     BC              ; LOOP INDEX
        LD      A,C
        OR      B
        JR      Z,LPX010
        JR      LP0010      ; :UNTIL (BC == 0)
LPX010  EQU     $
```

Between LP0010 and LPE010 we add the 8 in A to the address of A() in HL to get the next element. I'm indebted to George Phillips for this quick code which I found in his "Advanced Z-80 Programming", the print version of his talk at Tandy Assembly (http://48k.ca/advz80.pdf) . We can simply place there the next value of DE as an Integer. Then we count down BC to make sure we fill 3190 values. Next, we'll multiply each element x2.

```
        LD      BC,3190         ; "DO I= 1, 3190 A(I)= A(I) * 2"
        LD      DE,1
        LD      HL,ARRAYI
LP0020  EQU     $
        LD      A,8
        ADD     A,L
        LD      L,A
        JR      NC,LPE020
        INC     H
LPE020  EQU     $
        EX      DE,HL           ; &A(I) NOW IN DE
        LD      A,(DE)
        LD      L,A
        INC     DE
        LD      A,(DE)
        LD      H,A             ; HL= A(I)
        ADD     HL,HL           ; HL = HL * 2
        LD      A,H
        LD      (DE),A
        DEC     DE
        LD      A,L
        LD      (DE),A          ; A(I)= HL= A(I) * 2
        EX      DE,HL           ; &A(I) NOW BACK IN HL
        DEC     BC
        LD      A,C
        OR      B
        JR      Z,LPX020
        JR      LP0020
LPX020  EQU     $
```

(16-bit arithmetic is so much faster than Double Precision!) Now we subtract 1 from each element, thus giving us every odd number for the denominator of our formula.

```
        LD      BC,3190         ; "DO I=1, 3190 A(I)= A(I) – 1"
        LD      HL,ARRAYI
LP0030  EQU     $
        LD      A,8
        ADD     A,L
```

```
            LD      L,A
            JR      NC,LPE030
            INC     H
LPE030      EQU     $
            EX      DE,HL
            LD      A,(DE)
            LD      L,A
            INC     DE
            LD      A,(DE)
            LD      H,A
            DEC     HL              ; HL= HL - 1
            LD      A,H
            LD      (DE),A
            DEC     DE
            LD      A,L
            LD      (DE),A
            EX      DE,HL
            DEC     BC
            LD      A,C
            OR      B
            JR      Z,LPX030
            JR      LP0030
LPX030      EQU     $
```

Now we get to the actual Double Precision bits (pun intended). We're going to have to divide 1 by the number in each cell of A() and store that back into A(). Here's what we'll do:

1. Grab the integer "x" in the current cell of A().
2. Convert it to Double Precision.
3. Make sure it's in the right "accumulator", WRA2DP.
4. Load the "top" numerator accumulator WRA1DP with a Double Precision "1.0".
5. Divide 1/x.
6. Store the result back into A().

Here's steps 1 and 2:

```
            LD      BC,3190         ; "DO I= 1, 3190 A(I)= 1.0 / A(I)"
            LD      HL,ARRAYI
LP0040      EQU     $
            LD      A,8
            ADD     A,L
            LD      L,A
            JR      NC,LPE040
            INC     H
LPE040      EQU     $
            PUSH    BC              ; PRESERVE COUNT DOWN
```

```
PUSH    HL              ; PRESERVE &A(I)
LD      DE,WRA1         ; D.P. ACCUM 1=
LDI                     ;       INT VALUE IN
LDI                     ;       A(I)
LD      A,2             ; NTF=
LD      (NTF1),A        ;       2:  INTEGER
CALL    GETSPC          ; ON ERROR RESUME NEXT
CALL    0ADBH           ; WRA1DP= CDBL( WRA1)
POP     HL              ; RESTORE &A(I)
POP     BC              ;       AND COUNTER
LD      A,(BERROR)      ; IF (ERROR):
OR      A                   ; REPORT & EXIT
JP      NZ,ERRORS       ; :ENDIF
```

As suggested by George Phillips in the Tandy Discord, LDI is the quickest way to transfer a series of bytes. Here, we're moving the 2-byte Integer in A() to the Integer/Single Precision accumulator WRA1 at 4121H. Then we'll call 0ADBH to convert that to Double Precision in preparation for a D.P. division. A reminder that CALL GETSPC essentially forces a RET to the address after our CALL 0ADBH in case of a ROM-encountered error. (In that case, the ROM routine will have stored its error number in BERROR, which we immediately check, and if there's any, we JP to "ERRORS", which prints out a brief diagnostic and bails out.) GETSPC, its support vector and routines, and BERROR are all introduced in the June 2025 issue. Now on to steps 3 and 4, where we make sure the converted value is in the correct D.P. accumulator, and we place a D.P. "1.0" in the "numerator" accumulator preparatory to dividing them.

```
PUSH    BC
PUSH    HL
LD      HL,WRA1DP       ; COPY D.P. ACCUM 1
LD      DE,WRA2DP       ;   -> D.P. ACCUM 2
LDI                     ; (FASTEST:
LDI                     ;   8 INDIVIDUAL
LDI                     ;   MOVES)
LDI
LDI
LDI
LDI
LDI
LD      HL,DP01         ; DOUBLE PRECISION "1.0"
LD      DE,WRA1DP       ; DEST: ACCUMULATOR 1
LDI
LDI
LDI
LDI
```

```
LDI
LDI
LDI
LDI
```

"What's with all those LDIs? Why didn't you just load BC with 8 and use LDIR?" *Speed* is the factor here. If we used LDIR, we'd have to spend an extra 10 "t-states" LDing BC with an 8, and our last loop through the LDIR batch would cost an extra 5 t-states as well. What we'll do here for purposes of clarity is pretend we've made a little macro called LDI8 which is just replaced with the 8 LDIs in our code.

Step 5: We divide 1.0 by the value of the current cell of A()—we get its inverse.

```
CALL    GETSPC          ; ON ERROR RESUME NEXT
CALL    0DE5H           ; D.P. 1.0 / A(I)
POP     HL
POP     BC
LD      A,(BERROR)
OR      A
JP      NZ,ERRORS
```

Step 6: We store the D.P. value back into the current cell of A().

```
PUSH    BC
PUSH    HL
EX      DE,HL           ; DEST: &A(I)
LD      HL,WRA1DP       ; SOURCE: D.P. ACCUM 1
LDI8                    ;     (8 LDIs)
POP     HL
POP     BC
```

At this point, we've arrived at the end of the "DO loop", or "CONTINUE" in FORTRAN-speak. Let's test whether we've counted down through 3190 cells:

```
DEC     BC
LD      A,C
OR      B
JP      NZ,LP0040
```

We're now ready to alternately add or subtract each cell's value from T (in our FORTRAN program) or TOTAL (in our Assembly code). We'll set up FLAG to "false" and TOTAL to 1.0:

```
        XOR     A
        LD      (FLAG),A        ; "FLAG= .FALSE."
        LD      HL,DP01
        LD      DE,TOTAL        ; "T=
        LDI8                    ;    1.0"     (8 LDIs)
```

Now we can loop once more through A() from A(2) to A(3190). If FLAG is set, we *subtract* the cell's value from TOTAL; otherwise we *add* the value to TOTAL. Then we "flip" the setting of FLAG in order to alternate operations on the next cell.

```
        LD      BC,3190-1       ; "DO I= 2, 3190"
        LD      HL,ARRAYI
        LD      DE,8            ; START WITH A(2)
        ADD     HL,DE
LP0120  EQU     $
        LD      A,8
        ADD     A,L
        LD      L,A
        JR      NC,LPE120
        INC     H
LPE120  EQU     $
        PUSH    BC
        PUSH    HL
        LD      DE,WRA2DP           ; WRA2DP= A(I)
        LDI8                        ; 8 LDIs
        LD      HL,TOTAL
        LD      DE,WRA1DP           ; WRA1DP= TOTAL
        LDI8
        LD      A,(FLAG)
        OR      A
        JP      NZ,LPS120           ; IF (!FLAG):
        CALL    GETSPC
        CALL    0C70H                   ; "T= T - A(I)"
        JR      LPC120
LPS120  EQU     $                   ; ELSE:
        CALL    GETSPC
        CALL    0C77H                   ; "T= T + A(I)"
LPC120  EQU     $                   ; :ENDIF
        LD      HL,WRA1DP
        LD      DE,TOTAL
        LDI8                        ; T= WRA1DP (8 LDIs)
```

```
LD      A,(FLAG)
CPL                             ; "FLAG= .NOT. FLAG"
LD      (FLAG),A
POP     HL
POP     BC
LD      A,(BERROR)
OR      A
JP      NZ,ERRORS
DEC     BC
LD      A,B
OR      C
JP      NZ,LP0120               ; "CONTINUE"
```

The last bit of arithmetic we need to perform is to multiply TOTAL by 4.

```
LD      HL,TOTAL
LD      DE,WRA1DP
LDI8                            ; T=                (8 LDIs)
LD      HL,DP04                 ;     T * 4.0
LD      DE,WRA2DP
LDI8                            ;     (8 LDIs)
CALL    GETSPC
CALL    0DA1H                   ; WRA1DP= T * 4.0
LD      A,(BERROR)
OR      A
JP      NZ,ERRORS
LD      HL,WRA1DP
LD      DE,TOTAL                ; TOTAL= WRA1DP
LDI8                            ;     (8 LDIs)
```

Finally, we're ready to print out the result. TOTAL will need to be converted to a string, which we can then use PRINT to write to the screen. We'll also append a newline ("return") to make things look presentable.

```
LD      A,8                     ; 8: D.P. VALUE
CALL    FP2ASC
LD      A,(BERROR)
OR      A
JP      NZ,ERRORS
CALL    PRINT                   ; PRINT T
LD      HL,RETURN
CALL    PRINT
```

What's "FP2ASC"? Here it is.

```
;
;     FP2ASC -- CONVERT FLOATING POINT
;          (S.P. IN WRA1 OR
;          D.P. IN WRA1DP)
;          TO ASCII
;     ARG2:          A     NTF    (4: S.P.; 8: D.P.)
;     RETURNS:
;                    HL    POINTER TO BUFFER$
;
FP2ASC      EQU     $
            LD      (NTF1),A              ; INDICATE S.P. OR D.P.
            CALL    GETSPC                ; ON ERROR RESUME NEXT
            LD      HL,FP2BUF             ; BUFFER$
            CALL    0FBDH                 ; CONVERT # TO $
            RET
FP2BUF      DEFS    25                    ; BUFFER$
```

That's it! We've filled a 3190-element Double Precision array with 1..3190, doubled that, subtracted 1 to obtain all the odd numbers in that range, converted each element to Double Precision and divided each value into 1.0, then alternately added or subtracted each from our TOTAL. Finally, we multiplied TOTAL x 4.0, and printed out the result. For each Double Precision operation, we tested for errors. What I want to know is this: is our Assembly Language code any faster than FORTRAN? After all, Backus and his team invented that language originally to run as fast as the machine programs it was going to replace.

Language	*Timing*
FORTRAN	3:10
Assembly	2:38

YES! Our code is 1.2x faster than FORTRAN, and about 2.2x faster than BASIC. I believe that Double Precision arithmetic is probably the most challenging speed trial for a TRS-80. There's no math co-processor or built-in real number instructions as there was in the IBM 704 for which that the FORTRAN team was working. Everything has to be manipulated 8 bits at a time in the BASIC ROMs that Microsoft developed for Tandy, Apple, and Commodore. (This also explains why we heard—and believed—the reports that languages like Forth and C were so much faster on the TRS-80, with claims like "3 to 5 times quicker!" The benchmarks that were used to compare those programming environments, like the Sieve of Eratosthenes used to calculate prime numbers, typically didn't use floating point calculations, and definitely didn't use Double Precision.)

I haven't been able yet to determine if Microsoft's Fortran-80 FORLIB/REL, which provides math subroutines to FORTRAN and to Macro-80 assembly programs, calls the ROM or uses its own floating point routines. In any case, if speed is the goal—say, in developing a Fast Fourier Transform package—then Assembly Language calls to the ROM can provide an advantage.

Here's the updated list of "items"—mostly subroutines—that have been discussed in this and previous installments of "Power of the ROM".

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

	Double Precision Add	This issue
	Double Precision Subtract	
	Double Precision Multiply	
	Double Precision Divide	
CALL SPCMP	Single-Precision Compare Operation	V19 #3 Sept 2025
CALL FP2ASC	Convert Floating Point (Single or Double Precision) to String	This issue
	Convert Single-Precision...	V19 #2 June 2025,
CALL SP2STR	...to String	V20 #1 Mar 2026
CALL STR2SP	Convert String to Single-Precision	V19 #3 Sept 2025
CALL CONVRT	Convert Signed Integer to Single-Precision	V19 #2 June 2025
CALL SUI2SP	Convert Unsigned 24-bit Integer to Single-Precision	V19 #4 Dec 2025
	Single-Precision Constants and Constant Expressions	V19 #3 Sept 2025
	Floating Point Representation of Integers	V19 #4 Dec 2025
	BASIC Program to display all 4 bytes in a Single-Precision value	V19 #3 Sept 2025
WRA1	Integer & Single Precision Floating Point "Accumulator"	V19 #2 June 2025
WRA1DP	Double Precision "Accumulator" 1	This issue
WRA2DP	Double Precision "Accumulator" 2	This issue
	BASIC Program to display 4-byte Single Precision representations	V19 #2 June 2025
SHOWDUBL/BAS	BASIC Program to display 8-byte Double Precision representations	This issue
	Graphics:	
CALL GRFXSB	SET, RESET, or POINT x, y co-ordinates	V20 #2 June 2026
	Setup and Cleanup (Start and Finish) Routines:	
JP FINISH	Exit to TRSDOS, DEBUG, or Editor Assembler Plus' Z-BUG	V19 #1 Mar 2025
CALL INIT	Clear "BERROR" and turn off Tracing ("TROFF") Also clear Video cursor and Line Printer head position to 0	V19 #2 June 2025
CALL I00CRM	"USES MATH": Initialize Math Support routines, vectors, error-trapping	V19 #2 June 2025
CALL V4000R	"FORGET MATH": restores vectors at 4000JH..4012H to original state	V19 #2 June 2025
CALL SVSYNX	Save RST 08H vector prior to using graphics. Installs ADJRET vector there in its place.	V20 #2 June 2026
CALL RESYNX	Restore RST 08H vector saved previously by SVSYNX	V20 #2 June 2026
	Error-trapping:	
CALL GETSPC	Place this instruction directly before calling a ROM routine that may result in errors (like "Divide by Zero")	V19 #2 June 2025
CALL INITER	Sets up ROM routine error-trapping, using "ERTRAP"	V19 #2 June 2025
CALL ERTRAP	Error-trapping routine set up by CALL INITER. Catches ROM BASIC errors like "Divide by Zero", allowing user Assembly-Language programs to resume	V19 #2 June 2025
BERROR	Address of BASIC error code produced by a ROM call which fails (like "Divide by Zero")	V19 #2 June 2025
CALL BERTST	Checks BERROR and sets Z flag accordingly.	V20 #1 Mar 2026
	Data Structures:	
BARGS, BRETS	Byte (8-bit) Argument List, Return List	V20 #1 Mar 2026
WARGS, WRETS	Word (16-bit) Argument List, Return List	V20 #1 Mar 2026
RETURN	Carriage Return, aka "Newline", as a DEFW	V19 #1 Mar 2025
	Miscellaneous:	
4049H	Highest RAM Address as reported by TRSDOS 2.3	V19 #4 Dec 2025
	"FREE" Memory calculation for Assembly-Language Programs	V19 #4 Dec 2025
CALL HLSUB	"CALL HL": CALL subroutine whose address is stored in HL	V20 #1 Mar 2026
CALL ADJRET	Increments return address on stack by 1	V20 #2 June 2026

Now that I'm reviewing this table, I realize that I've never documented how to do a Single Precision *subtraction*. Here's some sample code:

```
;
;    SAMPLE CODE TO SUBTRACT S.P. VALUE IN NUM2
;         FROM NUM1, AND LEAVE RESULT IN SUBNUM
;
         LD       A,4              ; 4-BYTE /
         LD       (NTF1),A         ;    "SINGLE" PRECISION
         LD       HL,NUM2          ;         OPERATION
         LD       DE,WRA1
         LD       BC,4
         LDIR                      ; WRA1= NUM2
         LD       DE,(NUM1)
         LD       BC,(NUM1+2)      ; BC/DE= NUM1
         CALL     GETSPC           ; ON ERROR RESUME NEXT
         CALL     0713H            ; WRA1= BC/DE – WRA1
         LD       HL,WRA1
         LD       DE,SUBNUM
         LD       BC,4
         LDIR                      ; SUBNUM= NUM1 – NUM2
;
NUM1     DEFB     0                ; SINGLE
         DEFB     0                ;         PRECISION
         DEFB     16               ;              "9.0"
         DEFB     132
NUM2     DEFB     0                ; SINGLE
         DEFB     0                ;         PRECISION
         DEFB     32               ;              "5.0"
         DEFB     131
NUM3     DEFS     4                ; S.P. VARIABLE
;
```


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

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

The amazing

"FreHD"

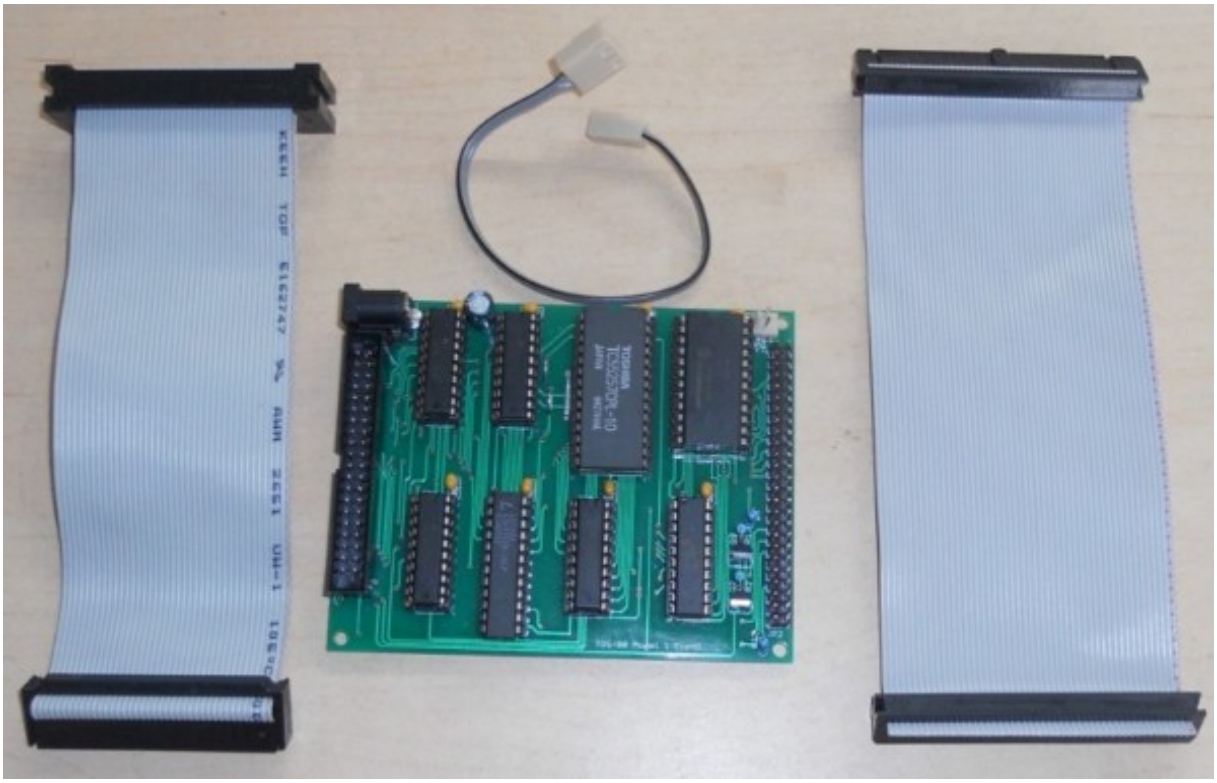


-Emulates a TRS-80 hard drive, but faster than any hard drive!
-Works with your favourite DOS (LS-DOS, LDOS, CP/M, Newdos/80 2.5)
-Uses SD card for storage medium
-Bonus free Real Time Clock function!
-Designed in Belgium and proudly built and shipped from Australia
-Kit form or fully assembled

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

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

- COMPLETLY ASSEMBLED AND TESTED -

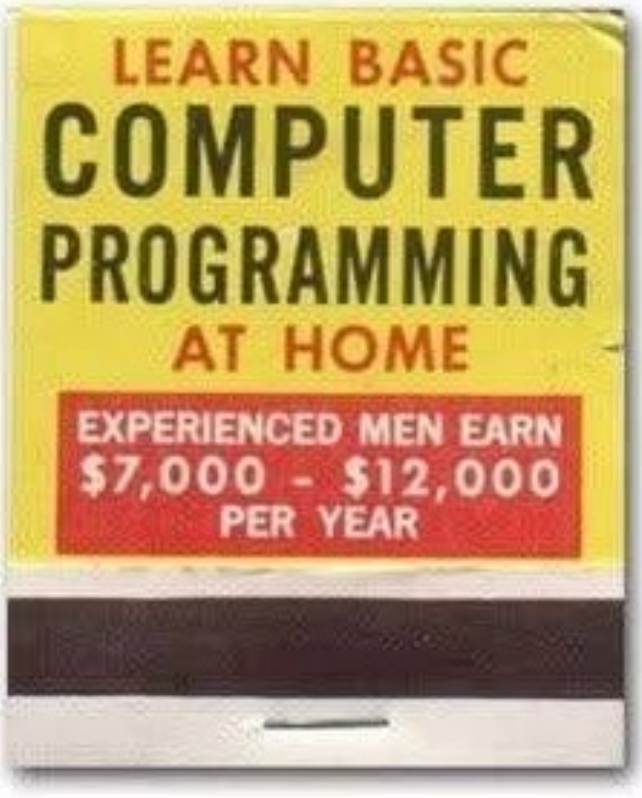


This is THE perfect device for all Model 1 users who own a 16K Level II unit, (which is most of us!), but no Expansion Interface or disk drives, especially if you don't want to modify your M1 with upgraded boot ROM or memory upgrade.

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

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


THE WAY
WE WERE
1985
1976
1983
1979
1978
1980
1981
1977
1982
1984


LEARN BASIC
COMPUTER
PROGRAMMING
AT HOME
EXPERIENCED MEN EARN
$7,000 - $12,000
PER YEAR


*115,000
MORE PROGRAMMERS
NEEDED BY 1975
HIGH SCHOOL
GRADS ELIGIBLE!
* U.S. DEPT OF LABOR
61-491

GENIE COMMANDS

Andrew Howard

If you've got a 16K or larger Video Genie and you want to start writing programs in structured BASIC, you need VGBAS1. It also contains some very useful routines that satisfy other needs.

VGBAS1 is a machine code program written for a Video Genie with a minimum of 16K memory which adds 13 useful new general-purpose commands to BASIC, three of them encouraging structured programming. The program occupies just under 1.5K and loads into 6000-672E (hex). The extra portion of end code is for initialisation only. Upon running, it relocates itself to be below BASIC but above everything else — it can therefore be used with Level III, for example. When loading several programs, VGBAS1 should be the last program to be loaded and run.

All of the new commands may be typed in as if they were normal BASIC statements. The 13 new commands are:

- REPEAT
- UNTIL
- PROC
- DEFPROC
- ENDPROC
- WHILE
- ENDWHILE
- JUMP
- !
- LINPUT
- CALL
- CANCEL
- EXEC

The first seven are structured statements, but the next two are not. The two after that are general-purpose, and the final two are special commands.

REPEAT-UNTIL

This has the following structure:

REPEAT
(Program section)
UNTIL expression = true

The program section is repeatedly executed until the expression in the UNTIL statement gives a TRUE (ie value of 1) value when evaluated. On completion of the loop, control is transferred to the next statement following UNTIL.
Format of REPEAT: REPEAT
Format of UNTIL: UNTIL expression
Example programs are given in Fig. 1.

Nested REPEAT-UNTILs are allowed, up to 10 levels. They work a little like FOR-NEXT loops: you cannot jump in or out without an error. Any attempt to attain an eleventh level of nesting will result in a "TOO MANY REPEATS" error. If an UNTIL is encountered that has not been initialised by a REPEAT then an "UNTIL WITHOUT REPEAT" error will occur.

PROC, DEFPROC — ENDPROC

The program structure of these commands looks like this:

(program)
PROC TITLE
(rest of program)
DEFPROC TITLE
Subroutine "TITLE"
ENDPROC

This command works like GOSUB except that instead of line numbers, labels are specified — an obvious advantage. The subroutine or procedure is called by the PROC command followed by the procedure name. The BASIC program is then switched for a DEFPROC with the same name following it. When found, the program is executed from there onwards, until an ENDPROC is found, upon which control is returned to the statement following the PROC. Note that DEFPROC does nothing on its own — it is merely a subroutine-beginning 'marker'.

```
10 PRINT"PRESS ANY KEY TO CONTINUE": A$=INKEY$
20 REPEAT: UNTIL INKEY$<>""
30 .....


10 PRINT"FIND SUM OF NUMBERS": PRINT"TERMINATE WITH
ZERO": T=0
20 REPEAT
30 INPUT"ENTER NEXT NUMBER";N: T=T+N
40 UNTIL N=0: PRINT"TOTAL =";T
```

Fig. 1

The DEFPROC at the beginning of the procedure **must** be the first item in a program line, otherwise it will not be found. Other statements may follow it, or start on new lines.
Format of PROC: PROC procedure name
Format of DEFPROC: DEFPROC procedure name
Format of ENDPROC: ENDPROC
The procedure name includes every character up to a colon or the end of a line. Program examples are given in Figs. 2 and 3.

As stated earlier, all VGBAS1 commands may be typed in as normal BASIC commands with one exception: after THEN and ELSE a colon should precede the VGBAS1 command, as illustrated in the example of Fig. 3. This example also shows that leading blanks and control codes before the procedure name are ignored, ie:
340 PROC DEMO
will find
2300 DEFPROCDEMO: . . . .
as if it were
340 PROCDEMO
Spaces may also be inserted before the procedure name in the DEFPROC statement, just to improve readability.

Nested PROCedure calls are allowed, up to 10 levels. Any attempt to exceed 10 levels will result in "TOO MANY PROCEDURES". If an ENDPROC is encountered that is not part of a subroutine, an "ENDPROC WITHOUT PROC" error will result

```
10 CLS: PRINT"ENTER A HEXADECIMAL NUMBER OF FOUR
DIGITS": PRINT
20 INPUT H$: PROCHEXDEC: PRINT"DECIMAL =";D: END
50 DEFPROCHEXDEC: REM ** HEX TO DEC CONVERSION
60 D=0: P=3: H=1: REPEAT: A=ASC(MID$(H$,H,1)):
A=A-48: A=A+(7*(A>9))
70 D=D+(A*16*P): H=H+1: P=P-1: REM ** IS POWER RAISE
FUNCTION
80 UNTIL P<0: ENDPROC
```

Fig. 2

```
10 CLS: REPEAT: PRINT"TYPE SOMETHING IN": INPUT S$
20 IF S$<>"STOP" THEN: PROCINVERT: PRINT"REVERSED":
PRINTR$
30 PRINT: UNTIL S$="STOP"
40 END
50 DEFPROCINVERT: R$="": FOR A=LEN(S$) TO 1 STEP -1
60 R$=R$+MID$(S$,A,1): NEXT A: ENDPROC
```

Fig. 3

TRS8BIT - VOLUME 20 ISSUE 03 - SEPTEMBER 2026 - PAGE 59 - TRS8BIT

```
6000                    00010           ORG     6000H
6000 E3         00020 TEST      EX      (SP),HL
6001 7C         00030           LD      A,H
6002 FE1D       00040           CP      1DH
6004 2003       00050           JR      NZ,NOTEXC
6006 7D         00060           LD      A,L
6007 FE5B       00070           CP      5BH
6009 E3         00080 NOTEXC    EX      (SP),HL
600A 2823       00090           JR      NZ,VECTOR
600C 114F60     00100 R0        LD      DE,CMDTBL
600F 23         00110           INC     HL
6010 0EFF       00120           LD      C,255
6012 E5         00130           PUSH    HL
6013 E1         00140 CMDLP     POP     HL
6014 E5         00150           PUSH    HL
6015 0C         00160           INC     C
6016 1A         00170 CMDLP1    LD      A,(DE)
6017 E67F       00180           AND     7FH
6019 47         00190           LD      B,A
601A 7E         00200           LD      A,(HL)
601B E67F       00210           AND     7FH
601D B8         00220           CP      B
601E 2812       00230           JR      Z,CHRMTC
6020 1A         00240 SKIP      LD      A,(DE)
6021 B7         00250           OR      A
6022 FA2860     00260 R1        JP      M,NXTCMD
6025 13         00270           INC     DE
6026 18F8       00280           JR      SKIP
6028 13         00290 NXTCMD    INC     DE
6029 1A         00300           LD      A,(DE)
602A B7         00310           OR      A
602B 20E6       00320           JR      NZ,CMDLP
602D E1         00330           POP     HL
602E 2B         00340           DEC     HL
602F C3781D     00350 VECTOR    JP      1D78H
6032 1A         00360 CHRMTC    LD      A,(DE)
6033 B7         00370           OR      A
6034 FA3B60     00380 R2        JP      M,FOUND
6037 13         00390           INC     DE
6038 23         00400           INC     HL
6039 18EB       00410           JR      CMDLP1
603B 23         00420 FOUND     INC     HL
603C D1         00430           POP     DE
603D E3         00440           EX      (SP),HL
603E 69         00450           LD      L,C
603F 2600       00460           LD      H,0
6041 29         00470           ADD     HL,HL
6042 117A60     00480 R3        LD      DE,ADDRS
6045 19         00490           ADD     HL,DE
6046 5E         00500           LD      E,(HL)
6047 23         00510           INC     HL
6048 56         00520           LD      D,(HL)
6049 D5         00530           PUSH    DE
604A DDE1       00540           POP     IX
604C E1         00550           POP     HL
604D DDE9       00560           JP      (IX)
604F 52         00570 CMDTBL    DEFM    'REPEAT'
6050 45
6051 50
6052 45
6053 41
6054 D4         00580           DEFB    54H+80H
6055 55         00590           DEFM    'UNTI'
6056 4E
6057 54
6058 49
6059 CC         00600           DEFB    4CH+80H
605A 43         00610           DEFM    'CANCE'
605B 41
605C 4E
605D 43
605E 45
605F CC         00620           DEFB    4CH+80H
6060 43         00630           DEFM    'CALL'
6061 41
6062 4C
6063 CC         00640           DEFB    4CH+80H
6064 A1         00650           DEFB    21H+80H
6065 4A         00660           DEFM    'JUMP'
6066 55
6067 4D
6068 D0         00670           DEFB    50H+80H
6069 50         00680           DEFM    'PROC'
606A 52
606B 4F
606C C3         00690           DEFB    43H+80H
606D 80         00700           DEFB    80H
606E 57         00710           DEFM    'WHILE'
606F 48
6070 49
6071 4C
6072 C5         00720           DEFB    45H+80H
6073 45         00730           DEFM    'EXEC'
6074 58
6075 45
6076 C3         00740           DEFB    43H+80H
6077 4C         00750           DEFM    'LF'
6078 89         00760           DEFB    89H
6079 00         00770           NOP
607A DA60       00780 ADDRS     DEFW    REPEAT
607C 5D61       00790           DEFW    UNTIL
607E 3F61       00800           DEFW    CANCEL
6080 AD62       00810           DEFW    CALL
6082 0663       00820           DEFW    DEFLAB
6084 6F63       00830           DEFW    JUMP
6086 CA63       00840           DEFW    PROC
6088 0964       00850           DEFW    END
608A 8564       00860           DEFW    WHILE
608C CF62       00870           DEFW    EXEC
608E 9A61       00880           DEFW    LINPUT
6090 9260       00890 REPPNT    DEFW    REPTBL
6092            00900 REPTBL    DEFS    40
60BA E5         00910 REPEAT    PUSH    HL
60BB 2A9060     00920 R4        LD      HL,(REPPNT)
60BE 11DA60     00930 R5        LD      DE,REPEAT
60C1 DF         00940           RST     18H
60C2 2006       00950           JR      NZ,REP2
60C4 E1         00960           POP     HL
60C5 3E01       00970           LD      A,1
60C7 C3C661     00980 R6        JP      ERROR
60CA E1         00990 REP2      POP     HL
60CB E5         01000           PUSH    HL
60CC DD2A9060   01010 R7        LD      IX,(REPPNT)
60D0 DD7500     01020           LD      (IX+0),L
60D3 DD7401     01030           LD      (IX+1),H
60D6 2AA240     01040           LD      HL,(40A2H)
60D9 DD7502     01050           LD      (IX+2),L
60DC DD7403     01060           LD      (IX+3),H
60DF 110400     01070           LD      DE,4
60E2 DD19       01080           ADD     IX,DE
60E4 DD229060   01090 R8        LD      (REPPNT),IX
60E8 E1         01100           POP     HL
60E9 C31E1D     01110           JP      1D1EH
60EC E5         01120 CANC0     PUSH    HL
60ED 2A9060     01130 R9        LD      HL,(REPPNT)
60F0 119260     01140 R10       LD      DE,REPTBL
60F3 DF         01150           RST     18H
60F4 2004       01160           JR      NZ,CAN2
60F6 E1         01170           POP     HL
60F7 C31E1D     01180           JP      1D1EH
60FA 2B         01190 CAN2      DEC     HL
60FB 2B         01200           DEC     HL
60FC 2B         01210           DEC     HL
60FD 2B         01220           DEC     HL
60FE 229060     01230 R11       LD      (REPPNT),HL
6101 E1         01240           POP     HL
6102 C31E1D     01250           JP      1D1EH
6105 E5         01260 CANC1     PUSH    HL
6106 2AA063     01270 R45       LD      HL,(PRCPNT)
6109 11A263     01280 R46       LD      DE,PRCTBL
610C DF         01290           RST     18H
610D 2004       01300           JR      NZ,CAN3
610F E1         01310           POP     HL
6110 C31E1D     01320           JP      1D1EH
6113 2B         01330 CAN3      DEC     HL
6114 2B         01340           DEC     HL
6115 2B         01350           DEC     HL
6116 2B         01360           DEC     HL
6117 22A063     01370 R47       LD      (PRCPNT),HL
611A E1         01380           POP     HL
611B C31E1D     01390           JP      1D1EH
611E            01400 TEMP      DEFS    2
6120 E5         01410 CANC2     PUSH    HL
6121 16FF       01420           LD      D,255
6123 221E61     01430 R61       LD      (TEMP),HL
6126 CD3619     01440           CALL    1936H
6129 F9         01450           LD      SP,HL
612A 22E840     01460           LD      (40E8H),HL
612D FE91       01470           CP      91H
612F 1E04       01480           LD      E,4
6131 C2A219     01490           JP      NZ,19A2H
6134 E1         01500           POP     HL
6135 211E1D     01510           LD      HL,1D1EH
6138 E3         01520           EX      (SP),HL
6139 2A1E61     01530 R60       LD      HL,(TEMP)
613C C3031F     01540           JP      1F03H
613F CD022B     01550 CANCEL    CALL    2B02H
6142 7B         01560           LD      A,E
6143 B7         01570           OR      A
6144 28A6       01580           JR      Z,CANC0
6146 FE01       01590           CP      1
6148 28BB       01600           JR      Z,CANC1
614A FE02       01610           CP      2
614C 28D2       01620           JR      Z,CANC2
614E FE03       01630           CP      3
6150 C29719     01640           JP      NZ,1997H
6153 E5         01650           PUSH    HL
6154 211E1D     01660           LD      HL,1D1EH
6157 E3         01670           EX      (SP),HL
6158 AF         01680 CANC3     XOR     A
6159 32CC62     01690 R69       LD      (EXECF6),A
615C C9         01700           RET
615D E5         01710 UNTIL     PUSH    HL
615E 2A9060     01720 R12       LD      HL,(REPPNT)
6161 119260     01730 R13       LD      DE,REPTBL
6164 DF         01740           RST     18H
6165 2006       01750           JR      NZ,UNT2
6167 E1         01760           POP     HL
6168 3E02       01770           LD      A,2
616A C3C661     01780 R14       JP      ERROR
616D E1         01790 UNT2      POP     HL
616E CD3703     01800           CALL    337FH
6171 E5         01810           PUSH    HL
6172 CD9409     01820           CALL    994H
6175 E1         01830           POP     HL
6176 C2EC60     01840 R71       JP      NZ,CANC0
6179 DD2A9060   01850 R15       LD      IX,(REPPNT)
617D DD2B       01860           DEC     IX
617F DD7E00     01870           LD      A,(IX+0)
6182 32A340     01880           LD      (40A3H),A
6185 DD2B       01890           DEC     IX
6187 DD7E00     01900           LD      A,(IX+0)
618A 32A240     01910           LD      (40A2H),A
618D DD2B       01920           DEC     IX
618F DD6600     01930           LD      H,(IX+0)
6192 DD2B       01940           DEC     IX
6194 DD6E00     01950           LD      L,(IX+0)
6197 C31E1D     01960           JP      1D1EH
619A E5         01970 LINPUT    PUSH    HL
619B 211E1D     01980           LD      HL,1D1EH
619E E3         01990           EX      (SP),HL
619F CD2828     02000           CALL    2828H
61A2 7E         02010           LD      A,(HL)
61A3 CDCD21     02020           CALL    21CDH
61A6 CD8D26     02030           CALL    268DH
61A9 CDF40A     02040           CALL    0AF4H
61AC D5         02050           PUSH    DE
61AD E5         02060           PUSH    HL
61AE CD6103     02070           CALL    361H
61B1 D1         02080           POP     DE
61B2 C1         02090           POP     BC
61B3 DABE1D     02100           JP      C,1DBEH
61B6 C5         02110           PUSH    BC
61B7 D5         02120           PUSH    DE
61B8 0600       02130           LD      B,0
61BA CD6828     02140           CALL    2868H
61BD E1         02150           POP     HL
61BE AF         02160           XOR     A
```


TRS8BIT - VOLUME 20 ISSUE 03 - SEPTEMBER 2026 - PAGE 60 - TRS8BIT

(the VGBAS1 equivalent of "RETURN WITHOUT GOSUB" error). Finally, if the PROC statement cannot find the specified procedure name, an "UNDEFINED LABEL" error will result.

WHILE — ENDWHILE

This structure is of a more complex nature. Its basic structure is as follows:

WHILE expression = true
(execute program section)
ENDWHILE
(rest of program)

While the expression, when evaluated, has a TRUE value (ie if A is equal to 2, then the expression "A=2" is said to be TRUE), then the program section is executed. If the expression is FALSE, then the program section is skipped and control is transferred to the ENDWHILE statement. Note that ENDWHILE is like DEFPROC: it does nothing on its own and is merely an index. On a FALSE condition, the BASIC program is searched from the WHILE statement onwards for an ENDWHILE. If no ENDWHILE can be found after the WHILE, a "NO ENDWHILE STATEMENT" error will occur.

Format of WHILE: WHILE expression
Format of ENDWHILE: ENDWHILE

```
10 CLS: PRINT"FIND SUM OF NUMBERS. PRECEDE HEX
NUMBERS WITH '$', IE $41 (65 DECIMAL)": T=0: PRINT
"TERMINATE WITH ZERO": REPEAT
20 INPUT"NEXT NUMBER IS ";N$
30 WHILE LEFT$(N$,1)="$"
40 N$=RIGHT$(N$,LEN(N$)-1): N=0: P=LEN(N$)-1
50 FOR A=1 TO LEN(N$): B=ASC(MID$(N$,A,1))-48:
B=B+(7*(B>9))
60 N=N+(B*16^P): P=P-1: NEXT A: N$=STR$(N)
70 ENDWHILE: T=T+VAL(N$): UNTIL VAL(N$)=0: PRINT
"TOTAL =";T
```

Fig. 4

Figure 4 gives an example of the use of these words. It should be noted that a statement such as:

```
300 WHILE(A=4 AND H=8)
310 B(6)=0: B(A*K-1)=2/B(A*K-J): PROCALCOTHERS: A=0:
H=6
320 ENDWHILE: B(6)=B(6)+2: GOTO 258
```

can be specified as:

```
300 WHILE(A=4 AND H=8): B(6)=0: B(A*K-1)=B(A*K-J):
PROCALCOTHERS: A=0: H=6
320 ENDWHILE: B(6)=B(6)+2: GOTO 258
```

The above illustrates only the basic capabilities of WHILE. More practical structures can be constructed using multiple WHILEs:

WHILE expression = true
(Execute program section)
WHILE expression = true
(Execute program section)
. . .
Other WHILE statements
. . .
ENDWHILE

This structure is exactly the same as the previous one but with one major improvement. If the expression in a WHILE statement evaluates to FALSE, then the program section following it is skipped. From that WHILE statement onwards, the BASIC program is searched for the next WHILE or ENDWHILE. That is, on a FALSE condition the WHILE statement searches for an ENDWHILE. If, however, another WHILE is found instead of an ENDWHILE, the program from there onwards is executed. The format of the commands remains the same. Figure 5 gives an example.

Again, upon a FALSE condition, if no more WHILEs can be found and there is no ENDWHILE, a "NO ENDWHILE STATEMENT" message will be given.

```
100 CLS: INPUT"HOW MANY NUMBERS ";N: DIM NM(N): FOR
A=1 TO N: PRINT"ENTER VALUE NUMBER ";A;: INPUT NM(A):
NEXT A: CLS
110 PRINT"YOU HAVE FOUR COMMANDS:": PRINT: PRINT "A -
CALCULATE MEAN": PRINT "S - CALCULATE SUM": PRINT"Q -
CALCULATE SUM OF SQUARES": PRINT "E - END"
120 PRINT: PRINT"YOU CAN SPECIFY STRINGS OF COMMANDS,
FOR EXAMPLE AS OR QAE OR SIMPLY S": PRINT
130 INPUT"ENTER COMMAND(S) ";CM$: IF CM$="" THEN 130
140 FOR A=1 TO LEN(CM$): C$=MID$(CM$,A,1):
PROCCOMAND: NEXT A: RUN
200 DEFPROCCOMMAND: REM ** COMMAND EXECUTION
210 WHILE C$="A"
220 AM=0: FOR B=1 TO N: AM=AM+NM(B): NEXT B: PRINT
"ARITHMETIC MEAN =";AM/N
230 WHILE C$="S"
240 SM=0: FOR B=1 TO N: SM=SM+NM(B): NEXT B: PRINT
"SUM =";SM
250 WHILE C$="Q"
260 SQ=0: FOR B=1 TO N: SQ=SQ+NM(B)*NM(B): NEXT B:
PRINT"SUM OF SQUARES =";SQ
270 WHILE C$="E"
280 CLEAR 50: END
290 ENDWHILE: ENDPROC
```

Fig. 5

Actually, it would be better to change the RUN in line 140 of Fig. 5 to GOTO 130.

As you may have worked out, nested WHILEs are not allowed. To overcome this, the WHILE structure becomes even more complex. The new statement formats are as follows (items in square brackets are optional):

Format of WHILE: WHILE [(expression1),] expression 2
Format of ENDWHILE: ENDWHILE [(expression 1)]
The structure is:

WHILE (expression 1), expression 2 = true
(Execute program section)
WHILE (expression 1), expression 2 = true
(Execute inner program section)
ENDWHILE (expression 1)
ENDWHILE (expression 1)

This mode of the WHILE structure can be very powerful if used well. Note that the 'normal' WHILE structure can still be used (normal being the structure as shown in Fig. 5): this new, more complex mode is optional.

When executed WHILE behaves as normal — upon a true condition the program section is executed as normal. However, upon a false condition, instead of searching for the next WHILE, the program is searched for the next WHILE with a matching value for expression 1. That is, if a WHILE is found with a differing value of expression 1 than that of the WHILE currently being executed, then it will be passed by and the next WHILE will be searched for. If, instead of a WHILE, an ENDWHILE is encountered with a matching value for expression 1, then control will be transferred here. As for WHILE, if an ENDWHILE with a differing value for expression 1 to that of the current WHILE is found, then it will be ignored.

The example above shows how this feature can be used in nesting WHILEs. As for DEFPROC, all ENDWHILEs and WHILEs must start on a new line:

456 PRINT: AG=0: ENDWHILE

is not allowed, but:

456 PRINT: AG=0
457 ENDWHILE

is allowed. An example of the new WHILE structure is given in Fig. 6. This is a rather silly example, but I couldn't think of anything else.

It should be noted that:
WHILE (expression 1), expression 2
where expression 1 evaluates to zero is exactly the same as:
WHILE expression
That is, on FALSE, WHILE(0), expression 2 would find the next WHILE(0), expression 2 or for the next WHILE expression or the next ENDWHILE(0) or the next ENDWHILE.

TRS8BIT - VOLUME 20 ISSUE 03 - SEPTEMBER 2026 - PAGE 61 - TRS8BIT

```
61BF C3321F   02170          JP      1F32H
61C2 00       02180 ERR      NOP
61C3          02190 ERREXT   DEFS    3
61C6 32C261   02200 ERR0R     LD      (ERR),A
61C9 3AA641   02210          LD      A,(41A6H)
61CC 2AA741   02220          LD      HL,(41A7H)
61CF 32C361   02230 R16      LD      (ERREXT),A
61D2 22C461   02240 R17      LD      (ERREXT+1),HL
61D5 3EC3     02250          LD      A,195
61D7 21E361   02260 R18      LD      HL,ERR0R2
61DA 32A641   02270          LD      (41A6H),A
61DD 22A741   02280          LD      (41A7H),HL
61E0 C39719   02290          JP      1997H
61E3 E1       02300 ERR0R2   POP     HL
61E4 3AC361   02310 R19      LD      A,(ERREXT)
61E7 2AC461   02320 R20      LD      HL,(ERREXT+1)
61EA 32A641   02330          LD      (41A6H),A
61ED 22A741   02340          LD      (41A7H),HL
61F0 3AC261   02350 R21      LD      A,(ERR)
61F3 87       02360          ADD     A,A
61F4 4F       02370          LD      C,A
61F5 0600     02380          LD      B,0
61F7 210862   02390 R22      LD      HL,ERRTBL-2
61FA 09       02400          ADD     HL,BC
61FB 5E       02410          LD      E,(HL)
61FC 23       02420          INC     HL
61FD 56       02430          LD      D,(HL)
61FE EB       02440          EX      DE,HL
61FF 3E3F     02450          LD      A,'?'
6201 CD3300   02460          CALL    33H
6204 CD752B   02470          CALL    2B75H
6207 C3FE19   02480          JP      19FEH
620A 1662     02490 ERRTBL   DEFW    MANYRP
620C 2762     02500          DEFW    NOREP
620E 3C62     02510          DEFW    NOLAB
6210 4C62     02520          DEFW    MANYPC
6212 6062     02530          DEFW    NOPRC
6214 7562     02540          DEFW    NOEND'
6216 54       02550 MANYRP   DEFM    'TOO MANY REPEATS'
6217 4F
6218 4F
6219 20
621A 4D
621B 41
621C 4E
621D 59
621E 20
621F 52
6220 45
6221 50
6222 45
6223 41
6224 54
6225 53
6226 00       02560          NOP
6227 55       02570 NOREP    DEFM    'UNTIL WITHOUT REPEAT'
6228 4E
6229 54
622A 49
622B 4C
622C 20
622D 57
622E 49
622F 54
6230 48
6231 4F
6232 55
6233 54
6234 20
6235 52
6236 45
6237 50
6238 45
6239 41
623A 54
623B 00       02580          NOP
623C 55       02590 NOLAB    DEFM    'UNDEFINED LABEL'
623D 4E
623E 44
623F 45
6240 46
6241 49
6242 4E
6243 45
6244 44
6245 20
6246 4C
6247 41
6248 42
6249 45
624A 4C
624B 00       02600          NOP
624C 54       02610 MANYPC   DEFM    'TOO MANY PROCEDURES'
624D 4F
624E 4F
624F 20
6250 4D
6251 41
6252 4E
6253 59
6254 20
6255 50
6256 52
6257 4F
6258 43
6259 45
625A 44
625B 55
625C 52
625D 45
625E 53
625F 00       02620          NOP
6260 45       02630 NOPRC    DEFM    'ENDPROC WITHOUT PROC'
6261 4E
6262 44
6263 50
6264 52
6265 4F
6266 43
6267 20
6268 57
6269 49
626A 54
626B 48
626C 4F
626D 55
626E 54
626F 20
6270 50
6271 52
6272 4F
6273 43
6274 00       02640          NOP
6275 4E       02650 NOEND'   DEFM    'NO ENDWHILE STATEMENT'G
6276 4F
6277 20
6278 45
6279 4E
627A 44
627B 57
627C 48
627D 49
627E 4C
627F 45
6280 20
6281 53
6282 54
6283 41
6284 54
6285 45
6286 4D
6287 45
6288 4E
6289 54
628A 00       02660          NOP
628B E3       02670 NEWRUN   EX      (SP),HL
628C 7C       02680          LD      A,H
628D FE1B     02690          CP      1BH
628F 2003     02700          JR      NZ,NOTRUN
6291 7D       02710          LD      A,L
6292 FE8F     02720          CP      8FH
6294 E3       02730 NOTRUN   EX      (SP),HL
6295 2803     02740          JR      Z,RUN2
6297 C30000   02750 RUNEXT   JP      0
629A E5       02760 RUN2     PUSH    HL
629B 219260   02770 R23      LD      HL,REPTBL
629E 229060   02780 R24      LD      (REPPNT),HL
62A1 21A263   02790 R42      LD      HL,PRCTBL
62A4 22A063   02800 R43      LD      (PRCPNT),HL
62A7 CD5861   02810 R70      CALL    CANC3
62AA E1       02820          POP     HL
62AB 18EA     02830          JR      RUNEXT
62AD 2B       02840 CALL     DEC     HL
62AE D7       02850          RST     10H
62AF FE24     02860          CP      '$'
62B1 2009     02870          JR      NZ,EXP
62B3 D7       02880          RST     10H
62B4 D29719   02890          JP      NC,1997H
62B7 CD5A1E   02900          CALL    1E5AH
62BA 1809     02910          JR      CALL2
62BC CD3723   02920 EXP      CALL    2337H
62BF E5       02930          PUSH    HL
62C0 CD7F0A   02940          CALL    0A7FH
62C3 EB       02950          EX      DE,HL
62C4 E1       02960          POP     HL
62C5 E5       02970 CALL2    PUSH    HL
62C6 211E1D   02980          LD      HL,1D1EH
62C9 E3       02990          EX      (SP),HL
62CA D5       03000          PUSH    DE
62CB C9       03010          RET
62CC 00       03020 EXECF0   NOP
62CD          03030 EXECST   DEFS    2
62CF CD3523   03040 EXEC     CALL    2335H
62D2 CF       03050          RST     8
62D3 29       03060          DEFM    ')'
62D4 3E01     03070          LD      A,1
62D6 32CC62   03080 R63      LD      (EXECF0),A
62D9 22CD62   03090 R64      LD      (EXECST),HL
62DC CD7929   03100          CALL    2979H
62DF 4E       03110          LD      C,(HL)
62E0 23       03120          INC     HL
62E1 5E       03130          LD      E,(HL)
62E2 23       03140          INC     HL
62E3 56       03150          LD      D,(HL)
62E4 0600     03160          LD      B,0
62E6 78       03170          LD      A,B
62E7 B1       03180          OR      C
62E8 CA9719   03190          JP      Z,1997H
62EB 2AA740   03200          LD      HL,(40A7H)
62EE E5       03210          PUSH    HL
62EF C5       03220          PUSH    BC
62F0 EB       03230          EX      DE,HL
62F1 EDB0     03240          LDIR
62F3 C1       03250          POP     BC
62F4 E1       03260          POP     HL
62F5 E5       03270          PUSH    HL
62F6 09       03280          ADD     HL,BC
62F7 363A     03290          LD      (HL),3AH
62F9 23       03300          INC     HL
62FA 3680     03310          LD      (HL),80H
62FC 23       03320          INC     HL
62FD 3600     03330          LD      (HL),0
62FF E1       03340          POP     HL
6300 CDC01B   03350          CALL    1BC0H
6303 C31E1D   03360          JP      1D1EH
6306 013A3A   03370 DEFLAB   LD      BC,3A3AH
6309 CD0B1F   03380          CALL    1F0BH
630C C31E1D   03390          JP      1D1EH
630F 00       03400 FNDCMD   NOP
6310 2B       03410 FINDLB   DEC     HL
6311 D7       03420          RST     10H
6312 4D       03430          LD      C,L
6313 44       03440          LD      B,H
6314 2AA440   03450          LD      HL,(40A4H)
6317 7A       03460          LD      A,D
6318 320F63   03470 R31      LD      (FNDCMD),A
631B 7E       03480 JPLOOP   LD      A,(HL)
631C 23       03490          INC     HL
```

```
10 CLS: PRINT"ENTER A TYPE OF ANIMAL": PRINT"MAKE IT
A CAT, DOG, OR MOUSE": REPEAT: INPUT"ANIMAL ";A$:
UNTIL A$="CAT" OR A$="DOG" OR A$="MOUSE"
20 INPUT"ENTER THE NAME OF A FOOD FOR THIS ANIMAL ";
F$
30 REM ** F$ SHOULD BE "STOP" TO STOP THE PROGRAM
40 WHILE F$<>"STOP"
50 PROC PRINTDETAILS: PROCDELAY: RUN
60 ENDWHILE: END

100 DEFPROC   DELAY: REM ** PROCEDURES
110 FOR A=1 TO 1000: NEXT A: ENDPROC

200 DEFPROC   PRINTDETAILS
210 WHILE(1),A$="DOG"
220 PRINT"A DOG IS MAN'S BEST FRIEND"
230 WHILE(2),F$="MEAT"
240 PRINT"DOESN'T YOUR DOG DESERVE MEAT?"
250 ENDWHILE(2): PRINT"DOGS BARK LOUDLY"
260 WHILE(1),A$="MOUSE"
270 PRINT"SOME PEOPLE ARE SCARED OF MICE"
280 PRINT"BUT CATS AREN'T"
290 WHILE(1),A$="CAT"
300 PRINT"CATS ARE WIDELY LOVED AS PETS":PRINT"THOUGH
NOT BY CT'S EDITOR": PRINT"DO YOU LIKE CATS ";
310 REPEAT: REPEAT: C$="": INPUT C$: UNTIL C$<>"":
C$=LEFT$(C$,1): UNTIL C$="Y" OR C$="N"
320 WHILE(2),F$="FISH"
330 WHILE(3),C$="Y"
340 PRINT"YOUR CAT LIKES FISH. YOU LIKE YOUR CAT."
350 PRINT"DO YOU LIKE FISH?"
360 ENDWHILE(3): PRINT"FISH SWIM . . ."
370 WHILE(2),F$="DOG FOOD"
380 PRINT"STRANGE CAT! DOES IT BARK ??"
390 ENDWHILE(2): PRINT"FISH HATE CATS."
400 ENDWHILE(1): ENDPROC
```

Fig. 6

In the listing of Fig. 6, ordinary numbers (constants) were used for expression 1. Variables may also be used, hence the notation 'expression 1':

```
560 WHILE(A*B+(INT(SIN(.75+B+J)/.05)+.6+PZ(J)),
expression 2

235 WHILE(PEEK(32745)),expression 2

65  WHILE(A),expression 2
```

and so on. The result of any expression 1 will be converted to integer form and must be between 0 and 255 inclusive.

But the really powerful feature of the VGBAS1 WHILE structure is the ability to change the program structure by changing the result of expression 1. For example, if in line 290 of Fig. 6, instead of WHILE(1), A$="CAT", there was WHILE(J), A$="CAT", and the variable J had been set to 1 at line 10, for example, then changing J to 2 would change the whole structure of the PRINTDETAILS procedure. That is, if J was 2, then upon a false condition in line 260, line 290 would be passed by because it had a different value for expression 1. This example is only a simple one — used properly, WHILE can provide very complex programs.

```
10 REM ** INITIALISE VARIABLES
20 ...

70 JUMPTITLES: REM ** GO AND DO TITLE AND START
PROGRAM
. . . REST OF PROGRAM . . .

900 !TITLES: CLS: PRINT" . . .


1000 ! LOOP: CLS: INPUT"TYPE ANY NUMBER ";N
1002 WHILE N=1
1004 PRINT"NUMBER ONE": PROC SUBONE
1006 WHILE N=2
1008 PRINT"NUMBER TWO": PROC SUBTWO

. . . OTHER WHILES . . .

1070 REM ** LOOP AROUND AND GET ANOTHER NUMBER
1080 FOR A=1 TO 1000: NEXT A: JUMPLOOP
```

Fig. 7

JUMP, !

This command is a new form of GOTO, in that no line numbers have to be specified:

Format of JUMP: JUMP label
Format of ! : ! label

The ! command is like DEFPROC: it marks the start of a section of program. It must be at the start of a program line. The jump command transfers control to the ! with a matching label to that of the JUMP. The label does not include leading blanks and control codes, but does include every character from there onwards to the next colon or end of the line. Figure 7 shows some examples of JUMP.

If the JUMP label cannot be found within the program, then "UNDEFINED LABEL" will result. Note that the following are not allowed:

```
560 PROCGETKEY: . . . rest of program
700 !GETKEY: . . . rest of program
```

and

```
2150 DEFPROCPRINTTITLE: . . .
3000 JUMP PRINTTITLE
```

LINPUT

This command allows a string literal to be input, including commas. The format is:
LINPUT ["message";] string variable
where the items in square brackets are optional. This command is exactly the same as an ordinary input except that no ? prompt is printed, commas and colons are allowed, and it can only be used with string variables. Figure 8 contains two examples: in the first case, when the program is RUN, CM$ will contain the **whole** string typed, including commas and colons. The second program is followed by a sample run.

```
10 CLS: PRINT"ENTER COMMAND": PRINT
20 PRINT"READY": CM$="": REPEAT: LINPUT CM$: UNTIL
CM$<>""
. . . rest of program . . .

10 CLS: PRINT"PRESONAL DETAILS": PRINT
20 LINPUT"ENTER YOUR NAME AND TELEPHONE NUMBER";PD$
30 PRINT: PRINT PD$

PERSONAL DETAILS

ENTER YOUR NAME AND TELEPHONE NUMBER:FRED BLOGGS,
01-636-7834

FRED BLOGGS, 01-636-7834
```

Fig. 8

CALL

This command has two formats:
CALL expression
CALL $absolute address
It is used to transfer control to a machine code routine. On entry to this routine, the HL register pair will contain the address of the next byte of the BASIC program to be executed. A RETurn instruction will return control to BASIC at the point left off.

In the CALL expression format, expression must be supplied as for a POKE or PEEK. That is, to call a machine code subroutine at 28672 (7000 hex) would require CALL 28672, a CALL to 32767 would require CALL 32767, and so on. However, to call 32768 would require CALL−32768, 61440 would require CALL −4096, and so on. Therefore expression must be supplied as an integer, that is if the address is greater than 32767 then subtract 65536. Expression can be any expression, and therefore the CALL command can be flexible. For example, a menu program in BASIC that executes the appropriate machine code subroutine according to the user's option is given in Fig. 9.

The CALL dollar sign format is not as flexible. It is absolute, ie the address is not evaluated — it is merely converted to binary. That means that variables cannot be used. However, it is

```
631D B6        03500          OR      (HL)
631E 2816      03510          JR      Z,NOLABL
6320 E5        03520          PUSH    HL
6321 23        03530          INC     HL
6322 5E        03540          LD      E,(HL)
6323 23        03550          INC     HL
6324 56        03560          LD      D,(HL)
6325 D7        03570          RST     10H
6326 C5        03580          PUSH    BC
6327 47        03590          LD      B,A
6328 3A0F63    03600 R32      LD      A,(FNDCMD)
632B B8        03610          CP      B
632C C1        03620          POP     BC
632D 280C      03630          JR      Z,LABEL
632F E1        03640 NOLBMT   POP     HL
6330 56        03650          LD      D,(HL)
6331 2B        03660          DEC     HL
6332 5E        03670          LD      E,(HL)
6333 EB        03680          EX      DE,HL
6334 18E5      03690          JR      JPLOOP
6336 3E03      03700 NOLABL   LD      A,3
6338 C3C661    03710 R25      JP      ERROR
633B 23        03720 LABEL    INC     HL
633C E5        03730          PUSH    HL
633D C5        03740          PUSH    BC
633E CD7F63    03750 R34      CALL    TSTPRC
6341 C1        03760          POP     BC
6342 3803      03770          JR      C,PRCDEF
6344 E1        03780          POP     HL
6345 1801      03790          JR      LABEL2
6347 F1        03800 PRCDEF   POP     AF
6348 2B        03810 LABEL2   DEC     HL
6349 D7        03820          RST     10H
634A E5        03830          PUSH    HL
634B C5        03840          PUSH    BC
634C 0A        03850 CPLABL   LD      A,(BC)
634D BE        03860          CP      (HL)
634E 2804      03870          JR      Z,LABMTC
6350 C1        03880          POP     BC
6351 E1        03890          POP     HL
6352 18DB      03900          JR      NOLBMT
6354 23        03910 LABMTC   INC     HL
6355 03        03920          INC     BC
6356 7E        03930          LD      A,(HL)
6357 B7        03940          OR      A
6358 2804      03950          JR      Z,LBMTC1
635A FE3A      03960          CP      3AH
635C 20EE      03970          JR      NZ,CPLABL
635E 0A        03980 LBMTC1   LD      A,(BC)
635F B7        03990          OR      A
6360 2804      04000          JR      Z,LBMTC2
6362 FE3A      04010          CP      3AH
6364 20E6      04020          JR      NZ,CPLABL
6366 F1        04030 LBMTC2   POP     AF
6367 F1        04040          POP     AF
6368 E3        04050          EX      (SP),HL
6369 23        04060          INC     HL
636A 5E        04070          LD      E,(HL)
636B 23        04080          INC     HL
636C 56        04090          LD      D,(HL)
636D E1        04100          POP     HL
636E C9        04110          RET
636F 1621      04120 JUMP     LD      D,%!
6371 CD1063    04130 R39      CALL    FINDLB
6374 ED53A240  04140          LD      (40A2H),DE
6378 C31E1D    04150          JP      1D1EH
637B 50        04160 MSPR0C   DEFM    %PR0C!
637C 52
637D 4F
637E 43
637F 117B63    04170 TSTPRC   LD      DE,MSPR0C
6382 0604      04180          LD      B,4
6384 1A        04190 TSTLP    LD      A,(DE)
6385 BE        04200          CP      (HL)
6386 2802      04210          JR      Z,TSTLPM
6388 AF        04220          XOR     A
6389 C9        04230          RET
638A 13        04240 TSTLPM   INC     DE
638B 23        04250          INC     HL
638C 10F6      04260          DJNZ    TSTLP
638E 37        04270          SCF
638F C9        04280          RET
6390 E5        04290 DEF      PUSH    HL
6391 CD7F63    04300 R33      CALL    TSTPRC
6394 3806      04310          JR      C,DEFPRC
6396 E1        04320          POP     HL
6397 2B        04330          DEC     HL
6398 D7        04340          RST     10H
6399 C30000    04350 DEFEXT   JP      0
639C D1        04360 DEFPRC   POP     DE
639D C30663    04370 R44      JP      DEFLAB
63A0 A263      04380 PRCPNT   DEFW    PRCTBL
63A2           04390 PRCTBL   DEFS    40
63CA 16B0      04400 PR0C     LD      D,0B0H
63CC CD1063    04410 R56      CALL    FINDLB
63CF E5        04420          PUSH    HL
63D0 D5        04430          PUSH    DE
63D1 2AA063    04440 R36      LD      HL,(PRCPNT)
63D4 11CA63    04450 R37      LD      DE,PR0C
63D7 DF        04460          RST     18H
63D8 2007      04470          JR      NZ,PR0C2
63DA D1        04480          POP     DE
63DB E1        04490          POP     HL
63DC 3E04      04500          LD      A,4
63DE C3C661    04510 R38      JP      ERR0R
63E1 D1        04520 PR0C2    POP     DE
63E2 DD2AA063  04530 R40      LD      IX,(PRCPNT)
63E6 DD7100    04540          LD      (IX+0),C
63E9 DD23      04550          INC     IX
63EB DD7000    04560          LD      (IX+0),B
63EE DD23      04570          INC     IX
63F0 2AA240    04580          LD      HL,(40A2H)
63F3 DD7500    04590          LD      (IX+0),L
63F6 DD23      04600          INC     IX
63F8 DD7400    04610          LD      (IX+0),H
63FB DD23      04620          INC     IX
63FD DD22A063  04630 R41      LD      (PRCPNT),IX
6401 E1        04640          POP     HL
6402 ED53A240  04650          LD      (40A2H),DE
6406 C31E1D    04660          JP      1D1EH
6409 2B        04670 END      DEC     HL
640A D7        04680          RST     10H
640B 201E      04690          JR      NZ,ENDCMD
640D 3ACC62    04700 R65      LD      A,(EXECFG)
6410 B7        04710          OR      A
6411 2808      04720          JR      Z,ENDNRM
6413 AF        04730          XOR     A
6414 32CC62    04740 R66      LD      (EXECFG),A
6417 E1        04750          POP     HL
6418 2ACD62    04760 R67      LD      HL,(EXECST)
641B C31E1D    04770          JP      1D1EH
641E E5        04780 ENDNRM   PUSH    HL
641F 211E1D    04790          LD      HL,1D1EH
6422 E3        04800          EX      (SP),HL
6423 C3AF1D    04810          JP      1DAFH
6426 57        04820 MSWHIL   DEFM    %WHILE%
6427 48
6428 49
6429 4C
642A 45
642B E5        04830 ENDCMD   PUSH    HL
642C CD7F63    04840 R48      CALL    TSTPRC
642F 302C      04850          JR      NC,ENDWHL
6431 E3        04860          EX      (SP),HL
6432 2AA863    04870 R51      LD      HL,(PRCPNT)
6435 11A263    04880 R52      LD      DE,PRCTBL
6438 DF        04890          RST     18H
6439 2006      04900          JR      NZ,ENDPRC
643B E1        04910          POP     HL
643C 3E05      04920          LD      A,5
643E C3C661    04930 R50      JP      ERROR
6441 E1        04940 ENDPRC   POP     HL
6442 2AA863    04950 R53      LD      HL,(PRCPNT)
6445 2B        04960          DEC     HL
6446 2B        04970          DEC     HL
6447 2B        04980          DEC     HL
6448 2B        04990          DEC     HL
6449 22A863    05000 R68      LD      (PRCPNT),HL
644C 5E        05010          LD      E,(HL)
644D 23        05020          INC     HL
644E 56        05030          LD      D,(HL)
644F 23        05040          INC     HL
6450 7E        05050          LD      A,(HL)
6451 32A240    05060          LD      (40A2H),A
6454 23        05070          INC     HL
6455 7E        05080          LD      A,(HL)
6456 32A340    05090          LD      (40A3H),A
6459 EB        05100          EX      DE,HL
645A C31E1D    05110          JP      1D1EH
645D E1        05120 ENDWHL   POP     HL
645E 112664    05130 R54      LD      DE,MSWHIL
6461 0605      05140          LD      B,5
6463 CD8463    05150 R55      CALL    TSTPRC+5
6466 D29719    05160          JP      NC,1997H
6469 C30663    05170 R72      JP      DEFLAB
646C 00        05180 WHLNO    NOP
646D 2B        05190 TSTWNO   DEC     HL
646E D7        05200          RST     10H
646F FE28      05210          CP      %(%
6471 C0        05220          RET     NZ
6472 D7        05230          RST     10H
6473 CD3723    05240          CALL    2337H
6476 CD852B    05250          CALL    2B85H
6479 C2B287    05260          JP      NZ,87B2H
647C D5        05270          PUSH    DE
647D 2B        05280          DEC     HL
647E D7        05290          RST     10H
647F CF        05300          RST     8
6480 29        05310          DEFM    %)%
6481 D1        05320          POP     DE
6482 AF        05330          XOR     A
6483 7B        05340          LD      A,E
6484 C9        05350          RET
6485 AF        05360 WHILE    XOR     A
6486 326C64    05370 R74      LD      (WHLNO),A
6489 CD6D64    05380 R75      CALL    TSTWNO
648C 2005      05390          JR      NZ,WHILE2
648E 326C64    05400 R73      LD      (WHLNO),A
6491 CF        05410          RST     8
6492 2C        05420          DEFM    %,%
6493 CD3723    05430 WHILE2   CALL    2337H
6496 E5        05440          PUSH    HL
6497 CD9409    05450          CALL    994H
649A E1        05460          POP     HL
649B C21E1D    05470          JP      NZ,1D1EH
649E CD071F    05480          CALL    1F07H
64A1 23        05490          INC     HL
64A2 E5        05500 SRCHWL   PUSH    HL
64A3 7E        05510          LD      A,(HL)
64A4 23        05520          INC     HL
64A5 B6        05530          OR      (HL)
64A6 3E06      05540          LD      A,6
64A8 CAC661    05550          JP      Z,ERROR
64AB 23        05560          INC     HL
64AC 23        05570          INC     HL
64AD E5        05580          PUSH    HL
64AE D7        05590          RST     10H
64AF FE80      05600          CP      80H
64B1 2001      05610          JR      NZ,SKPEND
64B3 D7        05620          RST     10H
64B4 112664    05630 SKPEND   LD      DE,MSWHIL
64B7 0605      05640          LD      B,5
64B9 CD8463    05650 R62      CALL    TSTPRC+5
64BC 302F      05660          JR      NC,NOWHIL
64BE 3A6C64    05670 R77      LD      A,(WHLNO)
64C1 B7        05680          OR      A
64C2 280B      05690          JR      Z,NOWLNO
64C4 CD6D64    05700 R76      CALL    TSTWNO
64C7 2024      05710          JR      NZ,NOWHIL
64C9 3A6C64    05720 R78      LD      A,(WHLNO)
64CC BB        05730          CP      E
64CD 201E      05740          JR      NZ,NOWHIL
64CF E1        05750 NOWLNO   POP     HL
64D0 E1        05760          POP     HL
64D1 23        05770          INC     HL
64D2 23        05780          INC     HL
64D3 5E        05790          LD      E,(HL)
64D4 23        05800          INC     HL
```

```
10 DIM MS(5): FOR A=1 TO 5: READ MS(A): NEXT A: REM
** READ M/C SUBROUTINE ADDRESSES
20 DATA 28672,29254,29998,30327,30600
30 !MENU: CLS: PRINT,"    MASTER MENU": PRINT
40 PRINT,"(1) . . . "
50 PRINT,"(2) . . . "
60 PRINT,"(3) . . . "
70 PRINT,"(4) . . . "
80 PRINT,"(5) . . . "
90 PRINT,"(6) EXIT PROGRAM": PRINT
100 UU=0: REPEAT: PRINT,"    YOUR OPTION ";: INPUT
UU: UNTIL UU>0 AND UU<7 AND UU=INT(UU)
110 WHILE UU<6
120 CALL MS(UU): FOR A=1 TO 1000: NEXT A: JUMPMENU
130 ENDWHILE: END
```

Fig. 9

available because addresses do not have to be specified in 'integer' form, so 61440 can be specified as CALL$61440.

The CALL command can be used to add new commands to BASIC, as in the program of Fig. 10 which adds a BEEP command. This routine is executed by:

CALL 32512, duration, pitch or
CALL $32512, duration, pitch
where duration and pitch must be less than 256. Examples are:

```
350 DEFPROCERROR: REM ** ERROR ROUTINE
360 PRINT"INPUT ERROR": CALL 32512,50,50: REM **
SIGNAL ERROR
370 ENDPROC
```

CANCEL

Cancel is a general 'abort' command and is used to abandon various operations.
Format: CANCEL expression
Expression, when evaluated, must result in one of four values:

Result of expression	Action taken
0	Cancel current REPEAT-UNTIL loop
1	Cancel current PROC call
2	Cancel current GOSUB
3	Cancel EXEC mode

IF a CANCEL2 is specified, that will cancel the RETURN address, so that a future RETURN will produce an RG error. That is, the outcome of a CANCEL 2 would be as if the calling program GOTOed instead of GOSUBed.

The same applies to CANCEL 1. It will cancel the ENDPROC address so any future ENDPROC will produce an "ENDPROC WITHOUT PROC" error.

```
                ORG   7F00H
7F00 CF         RST   8         ;check for comma
7F01 2C         DEFB  ','       ;
7F02 CD 21 7F   CALL  7F21H     ;evaluate duration
7F05 43         LD    B,E       ;B = duration
7F06 C5         PUSH  BC        ;
7F07 CF         RST   8         ;check for comma
7F08 2C         DEFB  ','       ;
7F09 CD 21 7F   CALL  7F21H     ;evaluate pitch
7F0C C1         POP   BC        ;
7F0D 4B         LD    C,E       ;C = pitch
7F0E C5         PUSH  BC        ;
7F0F 3E 02      LD    A,2       ;first half of sound
                                ;blip
7F11 41         LD    B,C       ;B = pitch
7F12 D3 FF      OUT   (255),A   ;
7F14 10 FE      DJNZ  7F14H     ;wait a bit
7F16 3E 01      LD    A,1       ;second half of blip
7F18 41         LD    B,C       ;
7F19 D3 FF      OUT   (255),A   ;
7F1B 10 FE      DJNZ  7F1BH     ;
7F1D C1         POP   BC        ;
7F1E 10 EE      DJNZ  7F0EH     ;loop until dur = 0
7F20 C9         RET             ;return to BASIC
7F21 CD 02 2B   CALL  2B02H     ;evaluate expression
7F24 C8         RET   Z         ;return if result OK
7F25 C3 B2 07   JP    07B2H     ;overflow if result>255
7F28            END
```

Fig. 10

If a program jumps out of a REPEAT-UNTIL loop, CANCEL0 will cancel the incomplete loop. CANCEL3 will abort EXEC mode (explained later).

If expression, when evaluated, results in a value less than zero or greater than three, a syntax error will occor.

EXEC

The EXEC command is a new type of command and is used to execute a string, just as if it was a line of BASIC.
Format: EXEC (string expression)
The string is tokenised and an END command is placed on the end. The tokenised string is then executed by the BASIC interpreter. When the END command is encountered, control is returned to the statement following the EXEC command. Note that nested EXECs are not allowed.

When EXEC is executed, EXEC mode comes into force. The END command exits EXEC mode. It is possible for the string to contain statements which transfer control to the areas of the program. This is perfectly okay, but must be used with care. If this feature is used, then END will return control to the statement

```
10 CLEAR 1000: READ NC: DIM SC$(NC): FOR A=1 TO NC:
READ SC$(A): NEXT A
20 REM ** ARRAY SC$ CONTAINS ALL COMMANDS
30 DATA 5,SAVE,LOAD,VERIFY,PRINT,EDIT
40 CLS: PRINT"WORD PROCESSOR": PRINT"COMMAND MODE":
PRINT
50 PRINT"COMMAND ?": CM$="": REPEAT: LINPUT"*";CM$:
UNTIL CM$<>""
60 REM ** TEST FOR COMMAND AND EXECUTE APPROPRIATE
PROCEDURE
70 A=1: REPEAT
80 WHILE CM$=SC$(A): CANCEL0: EXEC("PROC"+SC$(A)):
GOTO 50
90 ENDWHILE: A=A+1: UNTIL A=6: GOTO 40

100 REM ** PROCEDURES
110 DEFPROCLOAD: . . . load routine
200 DEFPROCSAVE: . . . save routine
. . . rest of procedures . . .
```

Fig. 11

following the EXEC command. This is where CANCEL3 is useful. If, via EXEC, control is transferred to another area of the program, then CANCEL3 will cancel EXEC mode, and the END command will act as normal, ending the program.

The EXEC command must be used with care. The only restriction is that INPUT and LINPUT cannot be used without some kind of error resulting, and EXEC cannot be used in FOR-NEXT loops. Examples of EXEC are given in Fig. 11.

The EXEC command can also be used to evaluate expressions, for example:

```
A$="A=B*A+C+.76+INT(COS(A)*.75*SIN(B)+(C/A+A))":
EXEC(A$)
```

The program in Fig. 11 shows that EXEC can be a very useful command. However, it also shows that care must be taken when using procedure names and labels. It should work okay when the user types VERIFY, PRINT or EDIT. However, if either SAVE or LOAD is typed, the machine will respond with a syntax error in line 80. This is because the string to be executed is either PROCSAVE or PROCLOAD. Note that PROC SAVE and PROC LOAD work. Why? -Because there is a hidden (embedded) command in the original examples — CSAVE and CLOAD. To overcome this, change line 80 to:

```
80 WHILE CM$=SC$(A): CANCEL0: EXEC("PROC "+SC$(A)):
GOTO 50
```

The DEFPROCs in lines 110 and 200 will also have to be changed:

```
110 DEFPROC LOAD: . . .
200 DEFPROC SAVE: . . .
```

The program should now work fully. If you are thinking "Well, he probably typed out the program, then typed it into his computer and found it didn't work. So he had to add a bit to his instructions" — you're right!!

That concludes the instructions for VGBAS1.

TRS8BIT - VOLUME 20 ISSUE 03 - SEPTEMBER 2026 - PAGE 65 - TRS8BIT

```
64D5 56       05810          LD      D,(HL)
64D6 ED53A240 05820          LD      (40A2H),DE
64DA 23       05830          INC     HL
64DB CD5803   05840          CALL    358H
64DE B7       05850          OR      A
64DF C4A01D   05860          CALL    NZ,1DA0H
64E2 22E640   05870          LD      (40E6H),HL
64E5 ED73E840 05880          LD      (40E8H),SP
64E9 2B       05890          DEC     HL
64EA C35A1D   05900          JP      1D5AH
64ED E1       05910 NOWHIL   POP     HL
64EE E1       05920          POP     HL
64EF 5E       05930          LD      E,(HL)
64F0 23       05940          INC     HL
64F1 56       05950          LD      D,(HL)
64F2 EB       05960          EX      DE,HL
64F3 18AD     05970          JR      SRCHWL
64F5 00       05980          NOP
64F6 56       05990 TITLE    DEFM    'VGBASI VIDEO GENIE LEVEL 2 INTERP
RETER EXTENSION 1'
64F9 47
64F8 42
64F9 41
64FA 53
64FB 31
64FC 20
64FD 56
64FE 49
64FF 44
6500 45
6501 4F
6502 20
6503 47
6504 45
6505 4E
6506 49
6507 45
6508 20
6509 4C
650A 45
650B 56
650C 45
650D 4C
650E 20
650F 32
6510 20
6511 49
6512 4E
6513 54
6514 45
6515 52
6516 50
6517 52
6518 45
6519 54
651A 45
651B 52
651C 20
651D 45
651E 58
651F 54
6520 45
6521 4E
6522 53
6523 49
6524 4F
6525 4E
6526 20
6527 31
6528 0D       06000          DEFB    13
6529 42       06010          DEFM    'BY ANDREW HOWARD 25/02/83'
652A 59
652B 20
652C 41
652D 4E
652E 44
652F 52
6530 45
6531 57
6532 20
6533 48
6534 4F
6535 57
6536 41
6537 52
6538 44
6539 20
653A 32
653B 35
653C 2F
653D 30
653E 32
653F 2F
6540 38
6541 33
6542 0D00     06020          DEFW    13
6544 CDC901   06030 INIT     CALL    1C9H
6547 21F664   06040          LD      HL,TITLE
654A CD752B   06050          CALL    2B75H
654D 3A0340   06060          LD      A,(4003H)
6550 2A0440   06070          LD      HL,(4004H)
6553 322F60   06080          LD      (VECTOR),A
6556 223060   06090          LD      (VECTOR+1),HL
6559 3ABB41   06100          LD      A,(41BBH)
655C 2ABC41   06110          LD      HL,(41BCH)
655F 329762   06120          LD      (RUNEXT),A
6562 229862   06130          LD      (RUNEXT+1),HL
6565 3A5B41   06140          LD      A,(415BH)
6568 2A5C41   06150          LD      HL,(415CH)
656B 329963   06160          LD      (DEFEXT),A
656E 229A63   06170          LD      (DEFEXT+1),HL
6571 3EC3     06180          LD      A,195
6573 320340   06190          LD      (4003H),A
6576 32BB41   06200          LD      (41BBH),A
6579 325B41   06210          LD      (415BH),A
657C 2AA440   06220          LD      HL,(40A4H)
657F 2B       06230          DEC     HL
6580 E5       06240          PUSH    HL
6581 065E     06250          LD      B,94
6583 DD21B765 06260          LD      IX,DEST
6587 FD217366 06270          LD      IY,ADDR
658B E5       06280 RELOC    PUSH    HL
658C DD5E00   06290          LD      E,(IX+0)
658F DD5601   06300          LD      D,(IX+1)
6592 19       06310          ADD     HL,DE
6593 FD5E00   06320          LD      E,(IY+0)
6596 FD5601   06330          LD      D,(IY+1)
6599 EB       06340          EX      DE,HL
659A 73       06350          LD      (HL),E
659B 23       06360          INC     HL
659C 72       06370          LD      (HL),D
659D FD23     06380          INC     IY
659F FD23     06390          INC     IY
65A1 DD23     06400          INC     IX
65A3 DD23     06410          INC     IX
65A5 E1       06420          POP     HL
65A6 10E3     06430          DJNZ    RELOC
65A8 D1       06440          POP     DE
65A9 210060   06450          LD      HL,TEST
65AC 01F604   06460          LD      BC,TITLE-6000H
65AF EDB0     06470          LDIR
65B1 CD4D1B   06480          CALL    1B4DH
65B4 C37200   06490          JP      72H
65B7 0000     06500 DEST     DEFW    0
65B9 4F00     06510          DEFW    CMDTBL-6000H
65BB 2800     06520          DEFW    NXTCMD-6000H
65BD 3B00     06530          DEFW    FOUND-6000H
65BF 7A00     06540          DEFW    ADDRS-6000H
65C1 BA00     06550          DEFW    REPEAT-6000H
65C3 5D01     06560          DEFW    UNTIL-6000H
65C5 3F01     06570          DEFW    CANCEL-6000H
65C7 AD02     06580          DEFW    CALL-6000H
65C9 0603     06590          DEFW    DEFLAB-6000H
65CB 6F03     06600          DEFW    JUMP-6000H
65CD 9000     06610          DEFW    REPPNTM6000H
65CF BA00     06620          DEFW    REPEAT-6000H
65D1 C601     06630          DEFW    ERROR-6000H
65D3 9000     06640          DEFW    REPPNTM6000H
65D5 9000     06650          DEFW    REPPNTM6000H
65D7 9000     06660          DEFW    REPPNTM6000H
65D9 9000     06670          DEFW    REPTBL-6000H
65DB 9000     06680          DEFW    REPPNTM6000H
65DD 9000     06690          DEFW    REPPNTM6000H
65DF 9000     06700          DEFW    REPTBL-6000H
65E1 C601     06710          DEFW    ERROR-6000H
65E3 9000     06720          DEFW    REPPNTM6000H
65E5 C201     06730          DEFW    ERR-6000H
65E7 C301     06740          DEFW    ERREXT-6000H
65E9 C401     06750          DEFW    ERREXT-5FFFH
65EB E301     06760          DEFW    ERROR2-6000H
65ED C301     06770          DEFW    ERREXTM6000H
65EF C401     06780          DEFW    ERREXTM5FFFH
65F1 C201     06790          DEFW    ERR-6000H
65F3 8802     06800          DEFW    ERRTBL-6000H
65F5 1602     06810          DEFW    MANYRP-6000H
65F7 2702     06820          DEFW    NOREP-6000H
65F9 3C02     06830          DEFW    NOLAB-6000H
65FB 8B02     06840          DEFW    NEWRUN-6000H
65FD 9200     06850          DEFW    REPTBL-6000H
65FF 9000     06860          DEFW    REPPNT-6000H
6601 F604     06870          DEFW    TITLE-6000H
6603 F804     06880          DEFW    TITLE-5FFEH
6605 C601     06890          DEFW    ERROR-6000H
6607 CA03     06900          DEFW    PROC-6000H
6609 9003     06910          DEFW    DEF-6000H
660B 0F03     06920          DEFW    FNDCMD-6000H
660D 0F03     06930          DEFW    FNDCMD-6000H
660F 7F03     06940          DEFW    TSTPRC-6000H
6611 7F03     06950          DEFW    TSTPRC-6000H
6613 A003     06960          DEFW    PRCPNTM6000H
6615 CA03     06970          DEFW    PROC-6000H
6617 C601     06980          DEFW    ERROR-6000H
6619 4C02     06990          DEFW    MANYPC-6000H
661B 1003     07000          DEFW    FINDLB-6000H
661D A003     07010          DEFW    PRCPNT-6000H
661F A003     07020          DEFW    PRCPNT-6000H
6621 A203     07030          DEFW    PRCTBL-6000H
6623 A003     07040          DEFW    PRCPNT-6000H
6625 0603     07050          DEFW    DEFLAB-6000H
6627 A003     07060          DEFW    PRCPNTM6000H
6629 A203     07070          DEFW    PRCTBL-6000H
662B A003     07080          DEFW    PRCPNT-6000H
662D 0904     07090          DEFW    END-6000H
662F 7F03     07100          DEFW    TSTPRC-6000H
6631 C601     07110          DEFW    ERROR-6000H
6633 6002     07120          DEFW    NOPRC-6000H
6635 A003     07130          DEFW    PRCPNTM6000H
6637 A203     07140          DEFW    PRCTBL-6000H
6639 A003     07150          DEFW    PRCPNT-6000H
663B 2604     07160          DEFW    MSWHIL-6000H
663D 7B03     07170          DEFW    MSPROC-6000H
663F 8403     07180          DEFW    TSTPRC-5FFBH
6641 1003     07190          DEFW    FINDLB-6000H
6643 1E01     07200          DEFW    TEMP-6000H
6645 1E01     07210          DEFW    TEMP-6000H
6647 8504     07220          DEFW    WHILE-6000H
6649 CF02     07230          DEFW    EXEC-6000H
664B 7502     07240          DEFW    NOENDW-6000H
664D 2604     07250          DEFW    MSWHIL-6000H
664F 8403     07260          DEFW    TSTPRC-57FBH
6651 CC02     07270          DEFW    EXECFG-6000H
6653 CD02     07280          DEFW    EXECSTM6000H
6655 CC02     07290          DEFW    EXECFG-6000H
6657 CC02     07300          DEFW    EXECFG-6000H
6659 CD02     07310          DEFW    EXECST-6000H
665B A003     07320          DEFW    PRCPNTM6000H
665D CC02     07330          DEFW    EXECFG-6000H
665F 5801     07340          DEFW    CANC3-6000H
6661 9A01     07350          DEFW    LINPUTM6000H
6663 EC00     07360          DEFW    CANC0-6000H
6665 0603     07370          DEFW    DEFLAB-6000H
6667 6C04     07380          DEFW    WHLNO-6000H
6669 6C04     07390          DEFW    WHLNO-6000H
666B 6D04     07400          DEFW    TSTWNO-6000H
666D 6D04     07410          DEFW    TSTWNO-6000H
666F 6C04     07420          DEFW    WHLNO-6000H
6671 6C04     07430          DEFW    WHLNO-6000H
6673 0440     07440 ADDR     DEFW    A004H
```

HOW THE PROGRAM WORKS

Most of the program is self-explanatory. I shall therefore restrict my description to the more obscure points.

Initialisation program (INIT), lines 6030 to 6490. This program firstly sets all the vectors for the extension routines. Lines 6220 to 6490 then relocate the main program to below BASIC but above anything else. BASIC is moved up by resetting the pointer to the start of the BASIC program area (40A4H) and calling the NEW routine (at 1B4DH). Control is then returned to the READY message (JP 72H). The large data tables following are used in changing all the absolute addresses in the main program, according to the program's new address location.

Test for VBBAS1 command routine (TEST) lines 20 to 560. This routine is an extension to the RST 10H routine. The Execution Driver (ED) program in ROM which executes a line of BASIC performs RST 10H. The ED is located at 1D5AH. Upon execution of RST 10H control is transferred to TEST. Lines 20 to 90 make sure that execution mode is progress. If not, then control is transferred to the real RST 10H routine at 1D78H.

Through the use of tables lines 100 to 410 check that the next item to be executed is a VGBAS1 command. If not, then control is returned to the ED via 1D78H.

REPEAT routine (REPEAT), lines 910 to 1110. This routine is straightforward. It uses an internal buffer (REPTBL) of 40 bytes to hold the current address and the line number of the loop start. On entry to all VGBAS1 routines, HL contains the current address (ie the next byte address, in the BASIC program). REPPNT is an index into this table.

CANCEL routines (CANCEL, CANC0, CANC1, CANC2 and CANC3), lines 1120 to 1700. The CANCEL routine calls the ROM subroutine at 2B02H. This evaluates the expression, converts the result to integer and puts it in the DE register pair (D=high byte, E=low byte). Lines 1560 to 1640 then execute the appropriate CANC routine.

It should be noted that the 'R' labels, for example R61 in line 1430, are used by the INIT program for relocating the main program.

UNTIL routine (UNTIL), lines 1710 to 1960. Lines 1710 to 1780 check that there is a REPEAT-UNTIL loop in progress — error if not. The call to 2337H evaluates the expression and the call to 0994H tests for TRUE/FALSE condition. Since true=1, or non-zero, and false=0, zero, then line 1840 takes care of the loop.

LINPUT routine (LINPUT), lines 1970 to 2170. The first three lines set return address for the LET routine. The call to 2828H checks for illegal direct while 21CDH prints the message if one is supplied. The routine at 260DH gets the address of the variable into DE and 0AF4H checks that the variable is a string variable (TM error if not). Lines 2050 and 2100 then receive the input — 361H receives a line of input from the keyboard, and sets the carry (C) flag if Break was pressed. Line 2100 calls the STOP/END routine if this is the case. Lines 2110 to 2170 then register the input — 2868H builds a literal string pool entry and 1F32 is the LET routine.

ERROR routine (ERROR), lines 2180 to 2660. On entry to this routine, the A register contains the error number (1 to 6). The routine then executes the ROM error routine by performing a syntax error. However, the ?SN error is not actually printed because lines 2220 to 2280 set a vector, so that just before the error is printed, control is returned to the VGBAS1 error routine, ERROR2. Lines 2310 to 2340 reset the vector.

Table reset routine (NEWRUN), lines 2670 to 2830. Lines 2670 to 2740 make sure that the NEW/RUN command is in execution. If not, then control is returned to the calling routine or to Level III, whichever is appropriate, via the RUNEXT vector, set by the INIT routine.

Lines 2760 to 2820 reset the internal REPEAT-UNTIL/PROC buffers and cancel EXEC mode. Control is then returned to the caller/Level III.

CALL routine (CALL and CALL2), lines 2840 to 3010. Lines 2840 to 2870 check for absolute addressing mode. The absolute routine, lines 2880 to 2910 converts the address to binary and places the result in DE (this is done by the call to 1E5AH) before executing the subroutine entry routine, CALL 2.

The expression evaluation routine calls the ROM expression evaluator 2337H, and then converts the result into binary integer (call 0A7FH). The final address is then placed in DE for the CALL2 routine.

CALL2 transfers control to the machine code subroutine, whose address is in DE.

EXEC routine (EXEC), lines 3040 to 3360. Lines 3040 to 3090 evaluate the expression, test for parentheses, set EXEC mode and store the current address into the BASIC program. The subroutine at 29D7H tests the result for string (TM error if not) and on exit, HL contains the address of a three byte 'string header' containing the string's length (first byte) and a pointer to the start of the string (bytes 2 and 3). Lines 3110 to 3190 set BC to the length of the string and DE to the start of the string (its address). If no string has been specified (EXEC("")) then a syntax error results.

! routine (DEFLAB), lines 3370 3390. This routine skips to the end of the label by finding the next colon or end of line (the call to 1F0BH), and then executes the BASIC program from there.

JUMP routine (JUMP), lines 4120 to 4150. This routine finds the '!' command in the program with a matching label, via the FINDLB routine (used by JUMP and PROC).

Find label routine (FINDLB), lines 3140 to 4110. On entry to this routine the D register must contain the command to be searched for (this will be 21H for '!' or 0B0H for DEF) and HL must point to the label following the JUMP or PROC. Lines 3140 to 3470 set BC to point to the label to be found, HL to the start of the BASIC program, and the variable FNDCMD to the command to be found.

BASIC programs are stored in a 'queue' fashion, for example:

```
10 CLS
20 PRINT "HELLO"

These two lines would be stored as:

42E9 EF     — these two bytes are pointer to next line
42EA 42
42EB 0A     — these two bytes are line number
42EC 00
42ED 84     — this byte is the CLS command
42EE 00     — this byte is the end of line marker
42EF FC     — this is the next line. These two bytes point to the
              following line.
42F0 42
42F1 14     — this is the line number (0014 is hex for 20)
42F2 00
42F3 B2     — this is the PRINT command
42F4 22     — this is a quote
42F5 48     — 'H'
42F6 45     — 'E'
42F7 4C     — 'L'
42F8 4C     — 'L'
42F9 4F     — 'O'
42FA 22     — end quote
42FB 00     — end of line marker.
42FC 00     — this would be the pointer to the next line,
42FD 00     — but as it is 0000 it signifies the end of the program.
```

After line 3470, HL points to the start of the program, and in this exmple, HL would contain 42E9H. However, it should be noted that with VGBAS1 present BASIC programs start higher up in memory.

Lines 3480 to 3510 make sure that the end of the program has not been reached, and if so, an error is printed ("UNDEFINED LABEL").

Lines 3520 to 3590 set DE to the line number for that line, HL to the start of the text in that line (in the above example, that would be 42EDH or 42F3H), and B to the first character in the line. Line 3600 compares this character with the command to be looked for, and if the same, executes the compare label routine, LABEL. Otherwise, lines 3640 to 3690 set HL to point to the line and the program loops round to test this next line.

Lines 3720 to 3800 are executed when the correct command has been found, and set HL to point to the label after the command (skipping over the word PROC if it is there).

TRS8BIT - VOLUME 20 ISSUE 03 - SEPTEMBER 2026 - PAGE 67 - TRS8BIT

```
6675 0D60    07450          DEFW    R0+1
6677 2360    07460          DEFW    R1+1
6679 3560    07470          DEFW    R2+1
667B 4360    07480          DEFW    R3+1
667D 7A60    07490          DEFW    ADDRS
667F 7C60    07500          DEFW    ADDRS+2
6681 7E60    07510          DEFW    ADDRS+4
6683 8060    07520          DEFW    ADDRS+6
6685 8260    07530          DEFW    ADDRS+8
6687 8460    07540          DEFW    ADDRS+10
6689 BC60    07550          DEFW    R4+1
668B BF60    07560          DEFW    R5+1
668D C860    07570          DEFW    R6+1
668F CE60    07580          DEFW    R7+2
6691 E660    07590          DEFW    R8+2
6693 EE60    07600          DEFW    R9+1
6695 F160    07610          DEFW    R10+1
6697 FF60    07620          DEFW    R11+1
6699 5F61    07630          DEFW    R12+1
669B 6261    07640          DEFW    R13+1
669D 6B61    07650          DEFW    R14+1
669F 7B61    07660          DEFW    R15+2
66A1 C761    07670          DEFW    ERROR+1
66A3 D861    07680          DEFW    R16+1
66A5 D361    07690          DEFW    R17+1
66A7 D861    07700          DEFW    R18+1
66A9 E561    07710          DEFW    R19+1
66AB EB61    07720          DEFW    R20+1
66AD F161    07730          DEFW    R21+1
66AF FB61    07740          DEFW    R22+1
66B1 0A62    07750          DEFW    ERRTBL
66B3 0C62    07760          DEFW    ERRTBL+2
66B5 0E62    07770          DEFW    ERRTBL+4
66B7 BC41    07780          DEFW    41BCH
66B9 9C62    07790          DEFW    R23+1
66BB 9F62    07800          DEFW    R24+1
66BD A440    07810          DEFW    40A4H
66BF F940    07820          DEFW    40F9H
66C1 3963    07830          DEFW    R25+1
66C3 8660    07840          DEFW    ADDRS+12
66C5 5C41    07850          DEFW    415CH
66C7 1963    07860          DEFW    R31+1
66C9 2963    07870          DEFW    R32+1
66CB 9263    07880          DEFW    R33+1
66CD 3F63    07890          DEFW    R34+1
66CF D263    07900          DEFW    R36+1
66D1 D563    07910          DEFW    R37+1
66D3 DF63    07920          DEFW    R38+1
66D5 1062    07930          DEFW    ERRTBL+6
66D7 7263    07940          DEFW    R39+1
66D9 E463    07950          DEFW    R40+2
66DB FF63    07960          DEFW    R41+2
66DD A262    07970          DEFW    R42+1
66DF A562    07980          DEFW    R43+1
66E1 9E63    07990          DEFW    R44+1
66E3 0761    08000          DEFW    R45+1
66E5 0A61    08010          DEFW    R46+1
66E7 1861    08020          DEFW    R47+1
66E9 8860    08030          DEFW    ADDRS+14
66EB 2D64    08040          DEFW    R48+1
66ED 3F64    08050          DEFW    R50+1
66EF 1262    08060          DEFW    ERRTBL+8
66F1 3364    08070          DEFW    R51+1
66F3 3664    08080          DEFW    R52+1
66F5 4364    08090          DEFW    R53+1
66F7 5F64    08100          DEFW    R54+1
66F9 8863    08110          DEFW    TSTPRC+1
66FB 6464    08120          DEFW    R55+1
66FD CD63    08130          DEFW    R56+1
66FF 3A61    08140          DEFW    R60+1
6701 2461    08150          DEFW    R61+1
6703 8A60    08160          DEFW    ADDRS+16
6705 8C60    08170          DEFW    ADDRS+18
6707 1462    08180          DEFW    ERRTBL+10
6709 B564    08190          DEFW    SKPEND+1
670B BA64    08200          DEFW    R62+1
670D D762    08210          DEFW    R63+1
670F DA62    08220          DEFW    R64+1
6711 0E64    08230          DEFW    R65+1
6713 1564    08240          DEFW    R66+1
6715 1964    08250          DEFW    R67+1
6717 4A64    08260          DEFW    R68+1
6719 5A61    08270          DEFW    R69+1
671B A862    08280          DEFW    R70+1
671D 8E60    08290          DEFW    ADDRS+20
671F 7761    08300          DEFW    R71+1
6721 6A64    08310          DEFW    R72+1
6723 8F64    08320          DEFW    R73+1
6725 8764    08330          DEFW    R74+1
6727 8A64    08340          DEFW    R75+1
6729 C564    08350          DEFW    R76+1
672B BF64    08360          DEFW    R77+1
672D CA64    08370          DEFW    R78+1
6544         08380          END     INIT
00000 TOTAL ERRORS

ADDR    6673                ENDNRM  641E
ADDRS   607A                ENDPRC  6441
CALL    62AD                ENDWHL  645D
CALL2   62C5                ERR     61C2
CAN2    60FA                ERREXT  61C3
CAN3    6113                ERROR   61C6
CANC0   60EC                ERROR2  61E3
CANC1   6105                ERRTBL  620A
CANC2   6120                EXEC    62CF
CANC3   6158                EXECFG  62CC
CANCEL  613F                EXECST  62CD
CMRMTC  6032                EXP     62BC
CMDLP   6013                FINDLB  6310
CMDLP1  6016                FNDCMD  630F
CMDTBL  604F                FOUND   603B
CPLABL  634C                INIT    6544
DEF     6398                JPLOOP  631B
DEFEXT  6399                JUMP    636F
DEFLAB  6386                LABEL   633B
DEFPRC  639C                LABEL2  6348
DIST    65B7                LABMTC  6354
END     6489                LBMTC1  635E
ENDCMD  642B                LBMTC2  6366

LINPUT  619A                R47     6117
MANYPC  624C                R48     642C
MANYRP  6216                R5      60BE
MSPROC  637B                R50     643E
MSWHIL  6426                R51     6432
NEWRUN  628B                R52     6435
NOENDW  6275                R53     6442
NOLAB   623C                R54     645E
NOLABL  6336                R55     6463
NOLBMT  632F                R56     63CC
NOPRC   6260                R6      60C7
NOREP   6227                R60     6139
NOTEXC  6089                R61     6123
NOTRUN  6294                R62     64B9
NOWHIL  64ED                R63     62D6
NOWLNO  64CF                R64     62D9
NXTCMD  6088                R65     640D
PRCDEF  6347                R66     6414
PRCPNT  63A0                R67     6418
PRCTBL  63A2                R68     6449
PROC    63CA                R69     6159
PROC2   63E1                R7      60CC
R0      600C                R70     62A7
R1      6022                R71     6176
R10     60F0                R72     6469
R11     60FE                R73     648E
R12     615E                R74     6486
R13     6161                R75     6489
R14     616A                R76     64C4
R15     6179                R77     64BE
R16     61CF                R78     64C9
R17     61D2                R8      60E4
R18     61D7                R9      60ED
R19     61E4                RELOC   658B
R2      6034                REP2    60CA
R20     61E7                REPEAT  60BA
R21     61F0                REPPNT  6090
R22     61F7                REPTBL  6092
R23     629B                RUN2    629A
R24     629E                RUNEXT  6297
R25     6338                SKIP    6020
R3      6042                SKPEND  64B4
R31     6318                SRCHWL  64A2
R32     6328                TEMP    611E
R33     6391                TEST    6000
R34     633E                TITLE   64F6
R36     63D1                TSTLP   6384
R37     63D4                TSTLPM  638A
R38     63DE                TSTPRC  637F
R39     6371                TSTWNO  646D
R4      60BB                UNT2    616D
R40     63E2                UNTIL   615D
R41     63FD                VECTOR  602F
R42     62A1                WHILE   6485
R43     62A4                WHILE2  6493
R44     639D                WHLNO   646C
R45     6186
R46     6189
```

DEF routine (DEF), lines 4290 to 4370. This routine tests for the presence of the word 'PROC' following the DEF. If it is there, then the '!' routine is executed, skipping over the label. Otherwise, control is either returned to BASIC producing a syntax/Level III error or to Level III.

PROC routine (PROC), lines 4400 to 4660. The first four lines of this routine find the address (in HL) and line number (in DE) of the appropriate DEFPROC statement. Like REPEAT-UNTIL, PROC uses an internal buffer of 40 bytes (to allow 10 levels) to store the present address and line number.

END routines (END, ENDNRM, ENDPRC and ENDWHL), lines 4670 to 5170. This routine handles all END functions. The first three lines check for any suffix (ie check for ENDPROC/ENDWHILE), and so transfer control to the end - command routine, ENDCMD. Otherwise, lines 4700 to 4720 check for EXEC mode. If this is in force, then EXEC mode is cancelled, and control is returned to the statement following the EXEC. If EXEC mode is not in force, then the return address is placed on the stack and the normal END routine in ROM is executed (1DAFH).

WHILE routine (WHILE), lines 5360 to 5970. This routine first sets the value for expression1 (or the WHLNO) to zero. Then, if there is an expression1 it is evaluated, converted to integer and tested to make sure that it is 255 or less (overflow if not), and that it is enclosed in parentheses and followed by a comma. The result is then stored as WHLNO.

The routine at 1F07H is the REM routine and after line 5490 HL points to the next line pointer, following the end of line marker of the line containing the WHILE. As in FINDLB, lines 5500 and 5550 check that "NO ENDWHILE STATEMENT" has occurred or not.

This concludes the article on VGBAS1, a program providing 13 powerful new commands. VGBAS1 could be the first in a series of interpreter extensions programs, each enhancing the BASIC language that I hardly ever use!

TRS8BIT - VOLUME 20 ISSUE 03 - SEPTEMBER 2026 - PAGE 68 - TRS8BIT

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)



7,10,1,10,75,100

543

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

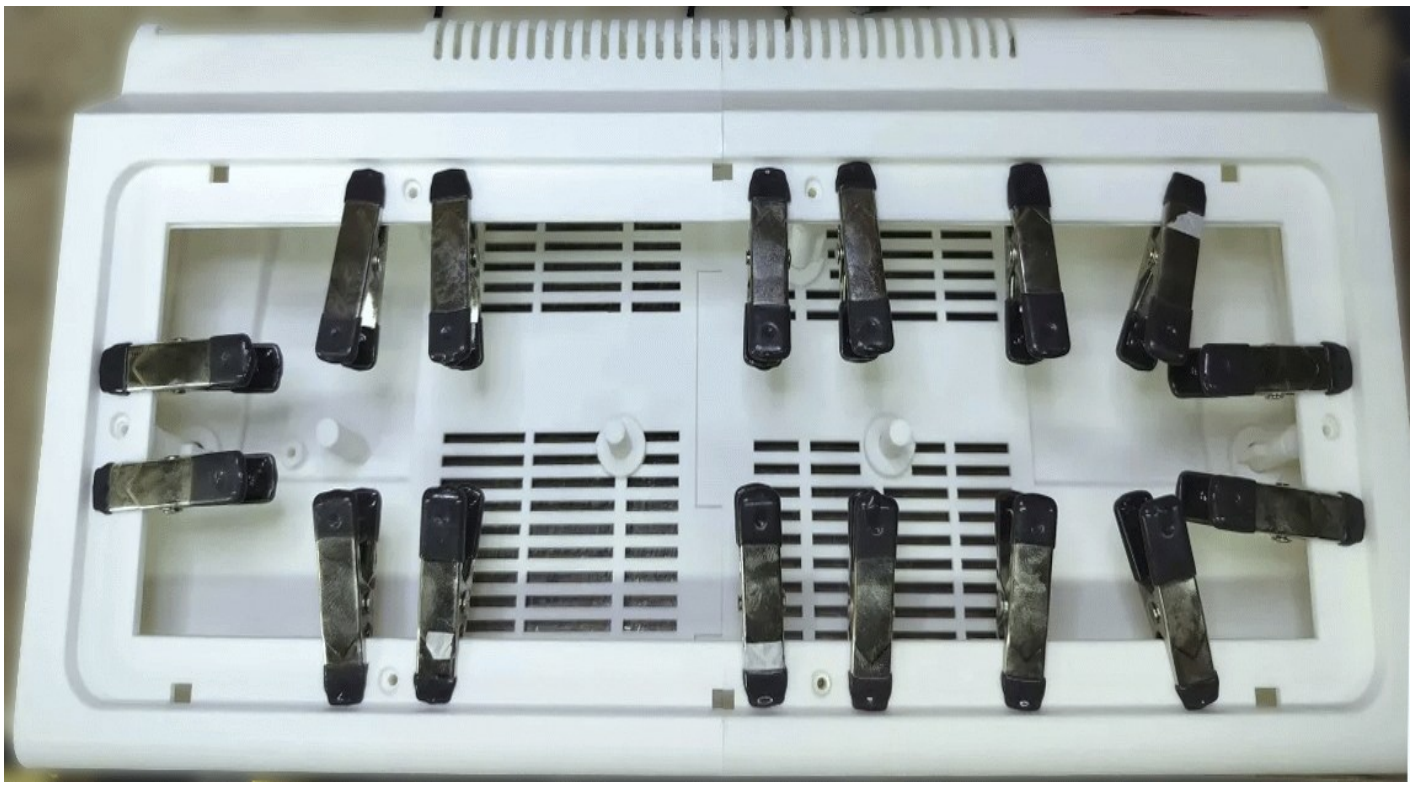
SIT DOWN WITH A COFFEE, PUT A TIMER ON AND SEE IF YOU CAN CALCULATE, **IN 30 SECONDS**, THE RED NUMBER FROM ANY OR ALL OF THE 6 BLACK NUMBERS, USING EACH NUMBER ONLY ONCE AND USING *,/,+ OR - ONLY.

NO CHEATING, BUT THE ANSWER IS ON PAGE 78

Surfin' The Net

Dusty of Dusty's Vintage Workshop (no relation!) has been fine-tuning his 3-D printable Model I case, and has released the STL files and builder's guide.

http://trs-80.com/cgi-bin/down-hardware-schematics.pl?Model%20I%20Reproduction%20Case%20with%20Builder%27s%20Guide%20v1.0%20(2026)(Dustys%20Vintage%20Workshop).zip



kDOS, lightweight OS for Floppy-80

KiwiSinceBirth, who has done so much for the modern TRS-80 community, has created kDOS, a Model I Operating System specifically for those Model I owners running a Floppy 80 (expansion interface NOT required). He describes kDOS

kDOS is an operating environment that runs on the Tandy TRS-80 Model 1 providing a TRS-DOS (DOS) like features to Level 2 BASIC
The DOS runs alongside Level 2 BASIC, rather than supplanting it. Typing a DOS command and a BASIC statement are done in the same shell
Files are stored and accessed from SD card in a standard FAT filesystem. There is no emulation of TRS-80 floppy disk or hard disk drives, nor any TRS -DOS (like) file-system

There are no imposed file size limits, or limits on the number of files that can be stored, for a TRS-80 a modern SD card could be considered infinite storage. Directories are fully supported. Files copied to the SD card by modern computer, are directly readable on the TRS-80

Programs that run on TRS-DOS "should" run under kDOS so long as they use standard DOS API calls. Programs that directly access FDC hardware or undocumented (non-standard) API calls will probably not function, unless patched

The core features are:
Modern Filesystem
Files are stored natively on the FAT filesystem
File directories (including nested) are supported
File storage space is unrestricted, can store any number of files
Automatic conversion of traditional filename/ext into filename.ext
DOS Features
DOS commands are typed into the BASIC command shell
Most DOS entry points at $4400 are supported
Interrupts with real Time Clock supported
BASIC features
Extended Basic functions are supported
Long error messages are displayed
Disk File IO Statements are currently NOT supported
Has a Low memory overhead than a traditional DOS
Requires about 0.5Kb of main memory space
Uses virtual memory from $3000-$37DF
47800 Bytes Free, compared with 48350 on Level 2 BASIC
Memory from 4500H is available for application usage

You can get more information at
https://github.com/kiwisincebirth/TRS-80-KDOS

TOUGH TIMES FOR RETRO COMPUTER BUYERS
by
Ian Mavric

In the last two weeks (I am writing this on the 1st of September, 2026) bizarre news has been coming out of the United States which should not be ignored by those who collect vintage computers and mainly buy them from the USA. I am by no means a lawyer, and indeed the information coming from people affected by this gets updated rapidly, but basically at the moment the United States treats all vintage computer and related accessories as e-waste, and will not allow shipping out of the country without a valid reason.

Some legislation was recently passed which seems to ban the shipping of old electronics out of the USA for reasons not entirely coherent but the reasons seem to be a) e-waste should be processed in the USA not shipped overseas for processing, and b) e-waste contains rare and/or valuable elements like gold and silver that should stay in the USA.

This mainly affects eBay sales from the USA using the eBay shipping program, and let's face it, if you buy vintage computer gear from the USA this is the way most of us do that. Sellers pack and send the items to an eBay shipping sorting centre, where they are bulk packed for shipping to other countries.

If an item is determined to be classified as e-waste then the eBay shipping centre 'liquidates' the item (their word, not mine), refunds the seller who then refunds the buyer, and the item is never heard from again. Items which are declared on the customs forms as vintage computer or vintage electronics, or not working/for repair, or have an extremely low declared value are especially vulnerable to liquidation due to this policy.

A post I saw online goes into some detail to explain this further:
"Yes, legally and logistically, vintage collectible electronics are strictly classified as e-waste, regardless of whether they are working, highly valuable, or destined for a museum display.

International customs laws and environmental agencies do not differentiate between a broken 2010 smartphone and a pristine, functioning 1980s Apple II computer. If it has a cord, a circuit board, or a battery, it falls under the universal legal umbrella of electronic waste (WEEE or e-waste).

The blanket restrictions affecting platforms like eBay stem from specific regulatory challenges associated with vintage gear:

The Legal Definition of "Waste" vs. "Product"

Under International treaties like the Basel Convention (which governs the transboundary movement of hazardous wastes), any electronic item being exported can be legally flagged as e-waste unless it undergoes rigorous, certified testing to prove it is fully functional.
Because eBay's automated shipping hubs cannot individually plug in, boot up, and test 40-year-old computers to verify functionality, the system treats the entire category as a compliance risk and blocks it.

Embedded Hazardous Materials Vintage electronics are heavily restricted by customs because they contain high concentrations of materials that are heavily restricted or outright banned today:
Leaded Solder: Older circuit boards used massive amounts of lead, which faces strict environmental import blocks in regions like the European Union (under RoHS regulations).

Degrading Components: Old motherboard batteries (NiCd or Lithium), massive leaking capacitors, and the toxic phosphor/ mercury coatings inside vintage CRT monitors are classified as hazardous material hazards for air cargo.

Missing Compliance and Recycling Fees Modern electronics include a built-in regulatory "eco-fee" or belong to modern Product Stewardship Schemes where manufacturers pay into a fund to handle the eventual recycling of the device.
Vintage items do not have this modern regulatory paperwork, meaning international customs agencies lack a mechanism to track their disposal, leading to strict automated blocks at shipping hubs like eBay's."

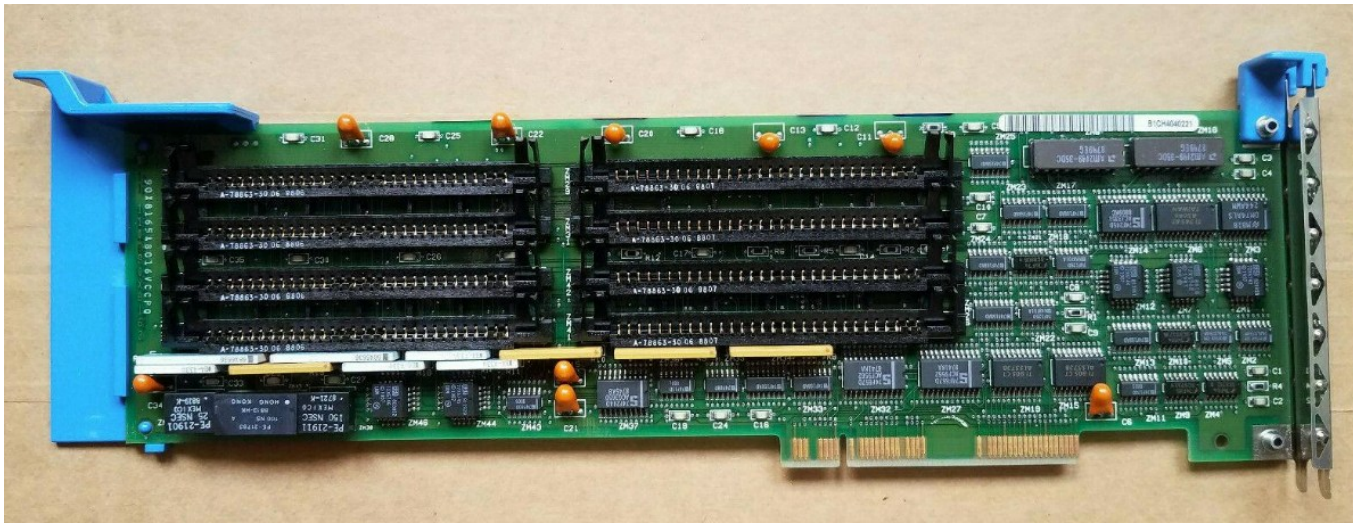
FOR THE FORSEEABLE FUTURE

You need to be aware of this when buying things out of the USA. Likely, you will be refunded so you will not be out of pocket, however there is no way this can be stopped once the item is in the eBay shipping program system. The best way I can see is to look at other countries when buying retro computing items as they are usually behind the USA and still ship items the same old way, with the postal system. This is of cold comfort if the item you are after is only available in the USA.

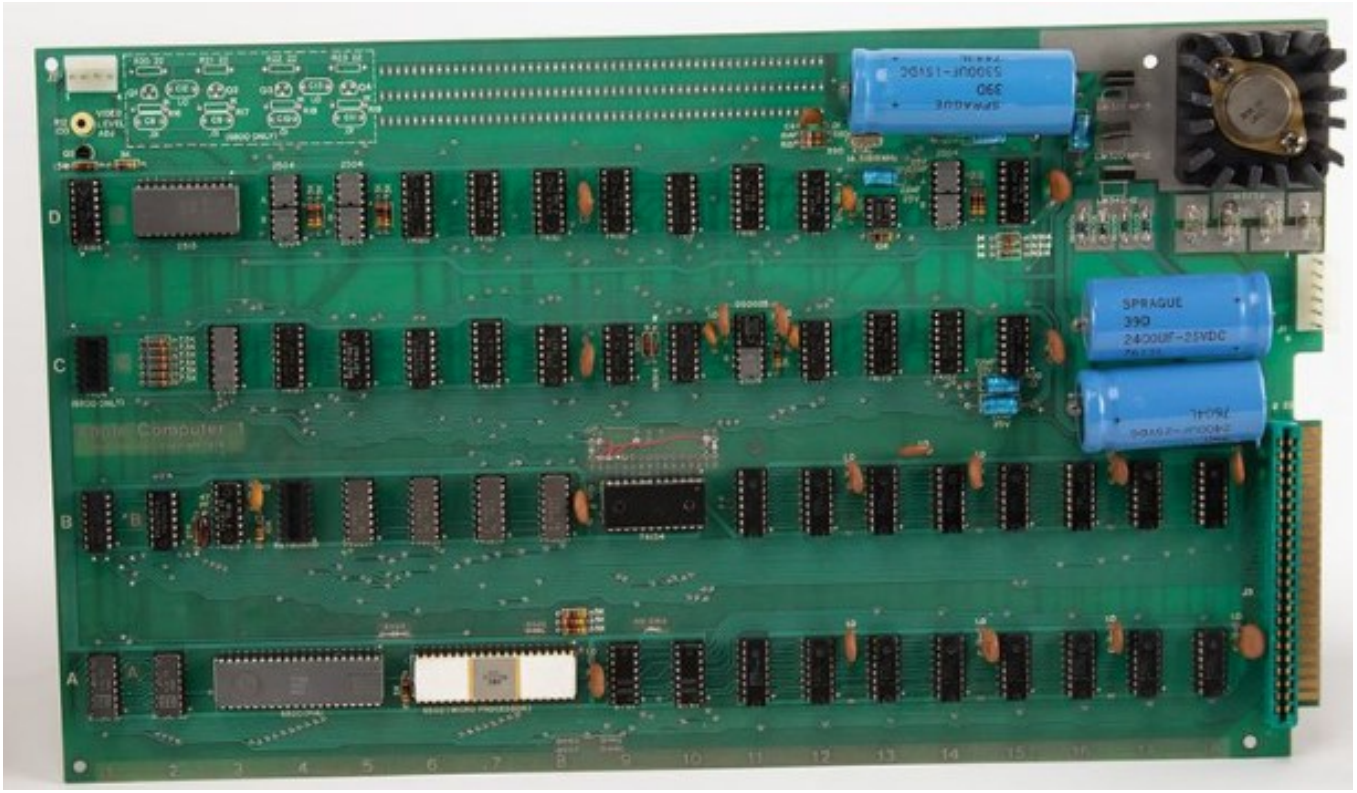


CONSIDERED AS E-WASTE

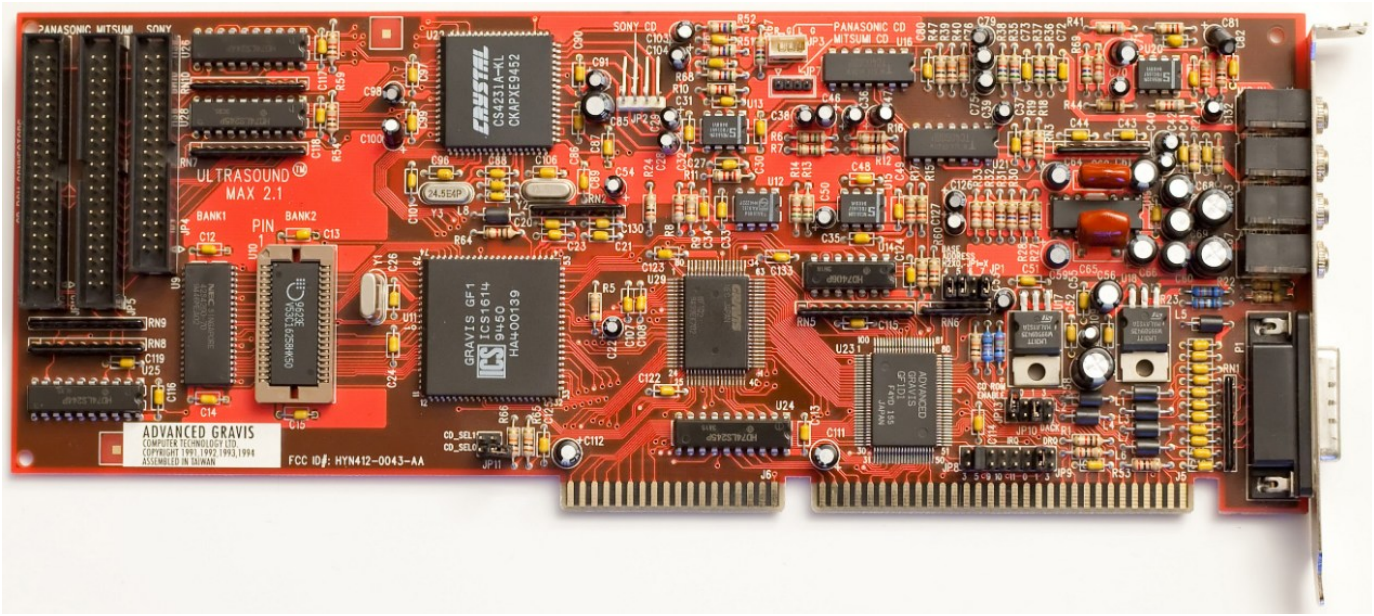
TRS-80 MODEL III



CONSIDERED E-WASTE: IBM MICROCHANNEL CARDS



CONSIDERED E-WASTE: 1976 APPLE 1



CONSIDERED E-WASTE: GRAVIS ULTRASOUND



CONSIDERED E-WASTE: IBM 486 WORKING COMPUTER SYSTEM



The *BEST* in TRS-80s

Call The Right Stuff

Ask for Ian
The number is +61 416 184 893

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





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

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

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

COMPETITION ANSWER - (75+1)*7+((100+10)/10) = 543