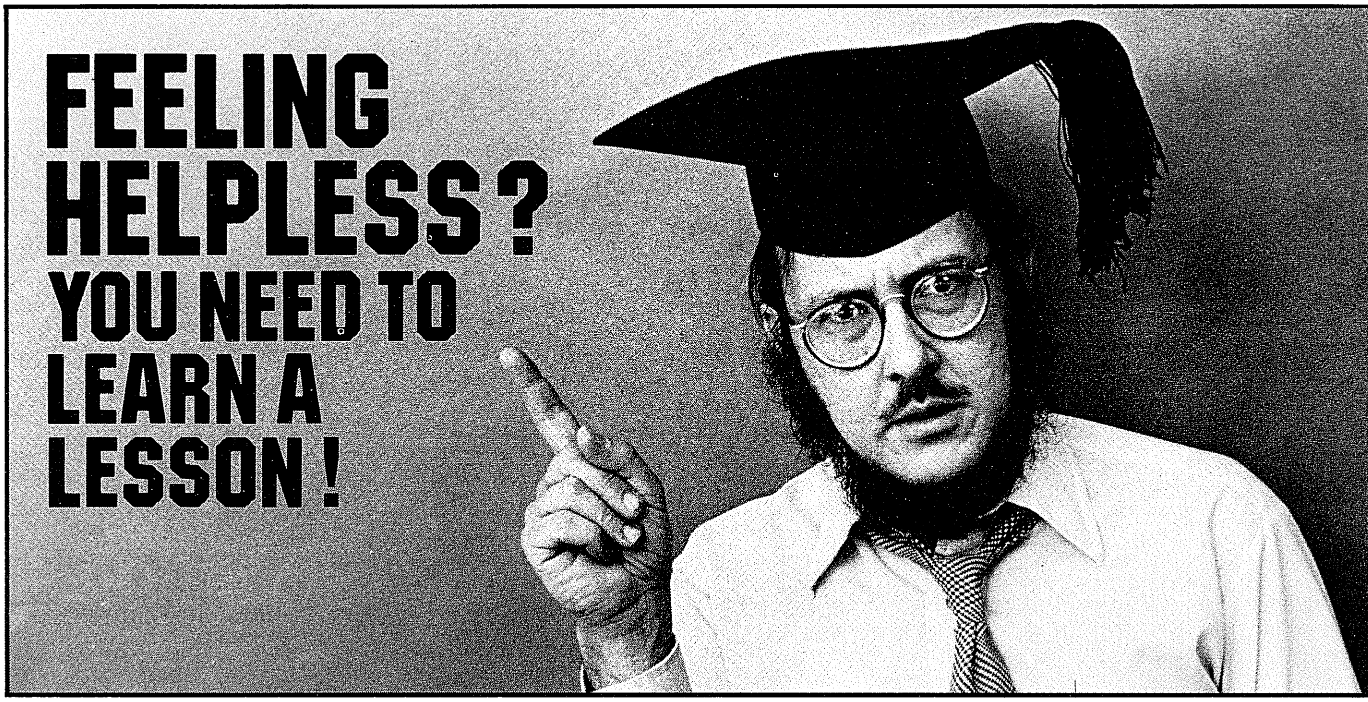
UNDERCOLOR

Vol. 1 No. 7 March 22, 1985



MICRO LANGUAGE LAB: LEARNING THE 6809

Feeling at the mercy of a programmer somewhere? Mystified by a machine? My **Micro Language Lab** will give you the power to end those helpless feelings.

Here's why. The heart of *any* computer program *anywhere* is machine language. Every piece of software uses it — your favorite game, spreadsheet, word processor, data base manager, or recipe file. Every time you hit "Enter", it's working. All the languages and operating systems are created from it. Basic, Fortran, Pascal, Flex, OS-9, TRSDOS... they're 6809 machine language, the whole lot!

And you can learn the language of the 6809, the programming heart of your Color Computer, with my **Micro Language Lab**.

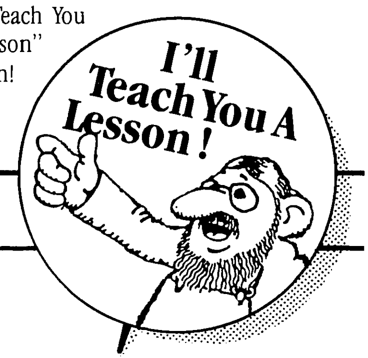
Not everyone can program. Writing a few lines in Basic now and then doesn't make you a programmer. But if you *can* program, then my **Micro Language Lab** will teach you — the right stuff, the right way.

I'll teach you in 24 half-hour lessons on 12 audio cassettes, with a 220-page textbook, with data booklets, with 35 sample programs, and with a programming reference card. You'll spend 50 hours or more with my course, listening, watching, and working. And when you're done, you'll be programming your Color Computer in the 6809's language.

Stop ignoring those gnawing feelings of helplessness. The 6809 is where the power lies in your computer. Turn on that power with the **Micro Language Lab.**

- **Micro Language Lab,** $99.00 (plus $3.50 shipping and handling)
  Requires 16K Extended Basic EDTASM +

Not sure? Write or call for a Table of Contents and sample pages — and your "I'll Teach You A Lesson" button!



ALSO AVAILABLE FROM GREEN MOUNTAIN MICRO

(Add $2.50 shipping and handling to your order)

Lowerkit III

- Full-time upper and lowercase installs in 15 minutes.
- Normal and reverse video standard
- Fully compatible with all Alpha and Graphic modes

$79.95 assembled and tested
$49.95 complete kit of parts
Important! Specify Color Computer or Color Computer II

TV Buff II, clear image for video monitors, $19.95 (specify CoCo or CoCo2)

CoCoPort parallel interface (now includes plastic case), $54.95 / $44.95 kit

ColorPack ROM/RAM pack, $29.95 (specify configuration) / $19.95 kit / case $3.00 extra

64K memory upgrade kit with full instructions, $49.95 / with memory test on tape, $54.95

Color Quaver Software Music Synthesizer on tape (requires 32/64K), $19.95

Color Burner EPROM Programmer (2716/32/32A/64/64A/128, 68764/66) with software, $69.95 / $56.95 kit

User Group, Educational, Club and Dealer Discounts are available.

TRS-80 and TRSDOS are trademarks of Tandy Corporation. Flex is a trademark of TSC, Inc. OS-9 is a trademark of Microware Corporation.

Green Mountain Micro
Bathory Road, Box U
Roxbury, Vermont 05669
802 485-6112
Hours: 9am - 5pm EST, Monday - Friday
COD/VISA/MASTERCARD

Who's Who

Editor
Debra A. Marshall

Technical
Dennis Bathory Kitsz

Contributing Photographer
Charley Freiberg

Production Manager
Clare McCarthy

Paste-Up
Janet Patterson

Logistics Coordinator
JoAnn Trottier

Technical Consultant and Publisher
Dennis Kitsz

Advertising:
Judy Knapp
(802)485-6139

Advertising:
Spencer Knowlton
(207)785-4151

Office
(802)485-6440

Editorial Office
(207)785-5148

Published By: ColorPlus Inc., Box 6809, Roxbury, VT 05669. (802)485-6440.

Circulation Problems? 2nd Class permit pending. Contact David Gunn, Box 6809, Roxbury, VT 05669, 802-485-6440.
Subscriptions: United States 1 year—$33; 2 years—$66. First class mail 1 year—$47; 2 years—$94. Canada and Mexico 1 year $43; 2 years $86. Overseas 1 year $53 (US dollars); Airmail $65 (US dollars). UnderColor is published 26 times a year. Call (802)485-6440, or write to UnderColor, Box 6809, Roxbury, VT 05669, Attn: Perley Swann.
Advertisers: rates available.
Editorial Office: c/o Debra Marshall, RFD 1, Box 786, Union, ME 04862, 207-785-5148. Submissions are welcome; include S.A.S.E. for returns. To receive author's guide, write: Author's Guide, Box 6809, Roxbury, VT 05669. Letters to Editor, manuscripts, article queries, etc. should be sent to editorial office.

Table of Contents

DefUsr

By Terry Kepner

I recently heard of a program by Dennis Kitsz that teaches Assembly Language. What are the facts on this? Also, I'm looking for an Epson printer for my 16K CoCo 2 cheap. Any ideas?

P.K.
Plaquemine, LA

You heard correctly. His 6809E lessons are reviewed in the January 1984 issue of *Hot CoCo*. The reviewer, Richard Ramella, gave them a good review and recommended them. The only warning is that you must *want* to learn to program or you'll become bored and not finish the tapes.

If you live near a large city, check the stores for "want-ad" booklets. These frequently have computer equipment for sale. Also, check the newspapers, local computer stores, and user groups. Someone is sure to have an Epson for sale that you can afford. Just be sure the Epson you get has an RS-232 board installed or it won't be of any use to you.

What are the differences between CSAVE "file-name" *and* CSAVE "file-name",A*? What are the pros and cons of each?*

Is there a command to data-files as SKIPF is to programs? If so, what is it?

Cory Sawyer
Lanesboro, MA

The CSAVE "filename",A syntax is used to save a Basic program in ASCII format instead of the normal tokenized format. All the Basic commands, such as Print, INKEY$, etc., are saved as text, just as you see them on the screen, instead of their one-byte tokenized forms as they are normally stored in memory and on tape (tokenized programs take less tape space and load quicker).

The advantage of the ASCII format is that programs saved in that form can be read by other programs with the normal INPUT#-1 statement, as if they were data files. Similarly, programs can be loaded into word processors for manipulation and modification and saved to tape for reading back into Basic.

The disadvantage is that you have to watch that the length of the ASCII Basic program lines aren't longer than 244 characters, or you'll get a DS Error message (direct statement error) when you try to reload the program (a DS Error means you tried to load in ASCII or tokenized characters without there being a line number in front of them).

The only safe way to do it is to save the program normally, then save it in ASCII and reload it. If you get a DS Error message, list the file to find the last line that loaded correctly. Now reload the normal file and list it to the line after that one. The line after the last one to load is the errant line. Edit the line to make it shorter, or break it into two new lines.

Now repeat the procedure until your ASCII file can reload entirely without a DS Error.

No, there isn't a SKIPF equivalent for data files. The only way to skip files is to open the file you want and wait for the tape to cue up to the proper spot on the tape.

I have been using a Color Computer for about two years, primarily for small-business accounting. A few months ago I upgraded my "E" board to Extended Basic and 32K. I used 64K chips but have not gotten around to soldering the wire jumper yet. Since the upgrade, several intermittent problems have occurred and no one seems to be able to help me.

When using the edit command in the insert mode, the cursor will occasionally hang up. It stays in one location, blinks, but doesn't respond to anything but the Reset button. This then requires that I retype nearly the entire (sometimes lengthy and complicated) line. Is this a faulty chip or an incurable problem?

My graphics functions operate sporadically. When trying the sample programs and exercises in the front of the Going Ahead with Extended Color Basic *book I often get an irregular oval on each side of the first three demonstratioin squares, or else a checkerboard pattern covers the top three-quarters of the graphics screen. Sometimes the same programs work fine. Can this problem be fixed?*

When using the Complete Personal Accountant package from Futurehouse, the cursor will occasionally disappear for twenty to thirty seconds. No data is lost, but I am concerned that someday it may never come back at all! This happens most frequently when entering a large number of checks but may happen when only a few checks have been entered. Is this a hardware or software problem?

Also, I cannot find out how many variables can be built into a program. The blue sheet in my original owner's manual says "you can now have a full 255 characters in data files without having to worry about losing any information." Does this mean that I can have 255 single-character-named variables each with any numerical amount but only 127 double-character variables (x 2) each of any value? If I dimension an array of variables (DIM x(20)) can I then include more than 255 variables? Or does "X(20))" copy five of the 255 allowed?

Roy Hakala
Red Wing, MN

Sounds like you are having problems with your RAM. Get out your 32K conversion instructions and double-check that everything is wired correctly. And *finish* the job! Anything that's only partially completed cannot be expected to perform correctly.

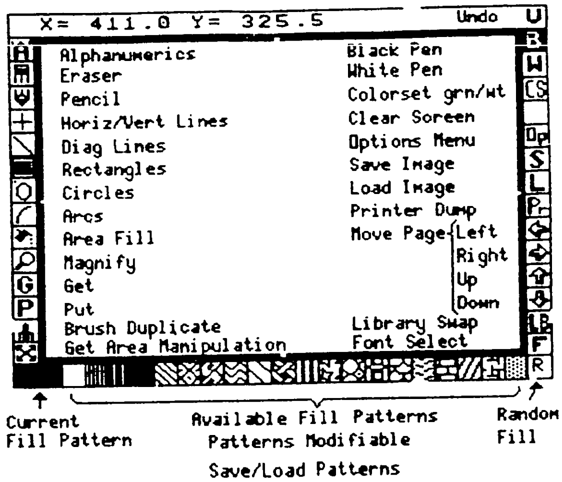
If you are still experiencing problems after that, get the Radio Shack Diagnostic ROMpak and run a memory check. You might have a marginal RAM chip. If you still can't get it to work right, take the unit to the repair center and have them examine it.

The time the cursor is gone is spent in "garbage collection." The computer is busy compacting data in RAM and deleting data no longer in use so that it will have more room for the current data. Don't worry about it.

The reference in the blue sheet to variable length refers to the length of the data that may be crammed into any one variable. The only limit on the number of variables in your computer is the amount of room you have available. You can have two variables or 500, Basic doesn't put a limit on the number, except that imposed by available RAM. If you have the RAM you can dimension an array of X(1000) if you want (or more if you have a small program).

VIZIDRAW 1.0

BY GEORGE ZINIEWICZ AUTHOR OF COLOR TOOLKIT



The INTERACTIVE DRAWING.BOARD and GRAPHIC DISPLAY PROCESSOR will produce amazing presentation-quality graphics. To build a picture use the 'ACTION ICONS' with your mouse or joysticks. ACTION ICONS include lines, rectangles, circles, arcs and fills. Once an object is created it can be copied and moved or undone. Magnify mode lets you 'zoom' in to do fine touch-up work. Brush Duplicate lets you 'air brush' any pattern. Comes with 14 FONT sets which can be enlarged or *italicised* or **BOLD FONT! Large worksheet (448x480 pixel)!**

REQUIRES 64K DISK or TAPE
2 JOYSTICKS or 1 MOUSE & JOYSTK
PRINTER OPTIONS: RS LP VIII FORMAT,
C-ITOH (NEC), EPSON, GEMINI 10-10X

PLEASE SPECIFY
DISK or TAPE

$49.95

(DEALER INQUIRIES INVITED) +$2 POSTAGE

VIZIDUMP

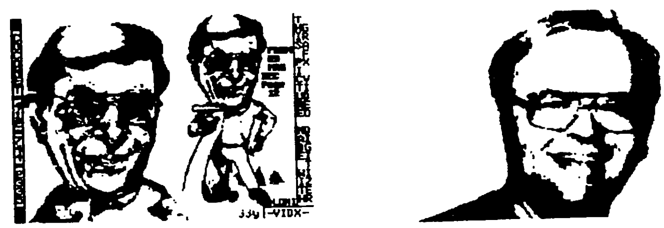
The screen dump utility that you've been waiting for! This does it all!. Easy PULL DOWN MENUS. Black/white backrounds. Select any area of the graphics screen and print it! You can also enlarge print 2X or 4X. Select tabs, baud rates, X or Y axis
PRINTER OPTIONS: RS LP VIII FORMATS
C-ITOH, EPSON, GEMINI 10 or 10X
REQUIRES DISK DRIVE & JOYSTICK.....

$19.95 +$2 POSTAGE

(DEALER INQUIRIES WELCOME)

GRAPHICS!

Turn your Color Computer into a "GRAPHICS MACHINE" with some of newest and most innovative graphics software programs yet!



VIDX - VIDEO DIGITIZER

Vidx connects between the video source and the RS-232 port of the color computer. No need for Y-cables or Multipacks! When VIDX is connected to your computer. you can digitize any source of video such as off air TV, video tape recorder, or live TV camera simply by issuing the proper command from the keyboard. The software used with VIDX is supplied on a non-protected RS format disk. The software has the following functions:

32K
64K
DISK or
CASSETTE

- SINGLE OR MULTIPLE EXPOSURE
- ABORT or RESUME SCAN
- FRAME PICTURE
- LOGICAL 'AND' SCREEN 2 & 1
- READ/WRITE GRAPHICOM DISK
- READ/WRITE RS-DOS DISK
- READ/WRITE TO CASSETTE

- OR - 16K CASSETTE

VIDX-1-**$99.95** DIGITIZER & SOFTWARE
VIDX-2-**$124.95** DIGITIZER & SOFTWARE
with two port RS-232 switcher, allows You to switch between VIDX and two other devices such as printer & modem.

PLUS $3.00 POSTAGE

HAM SOFTWARE by John Yurek used worldwide by Hams

CW - Turns your COCO into a CW keyboard and receive terminal. COCO version has a real time printer.
$44.95(tape) plus $2 postage

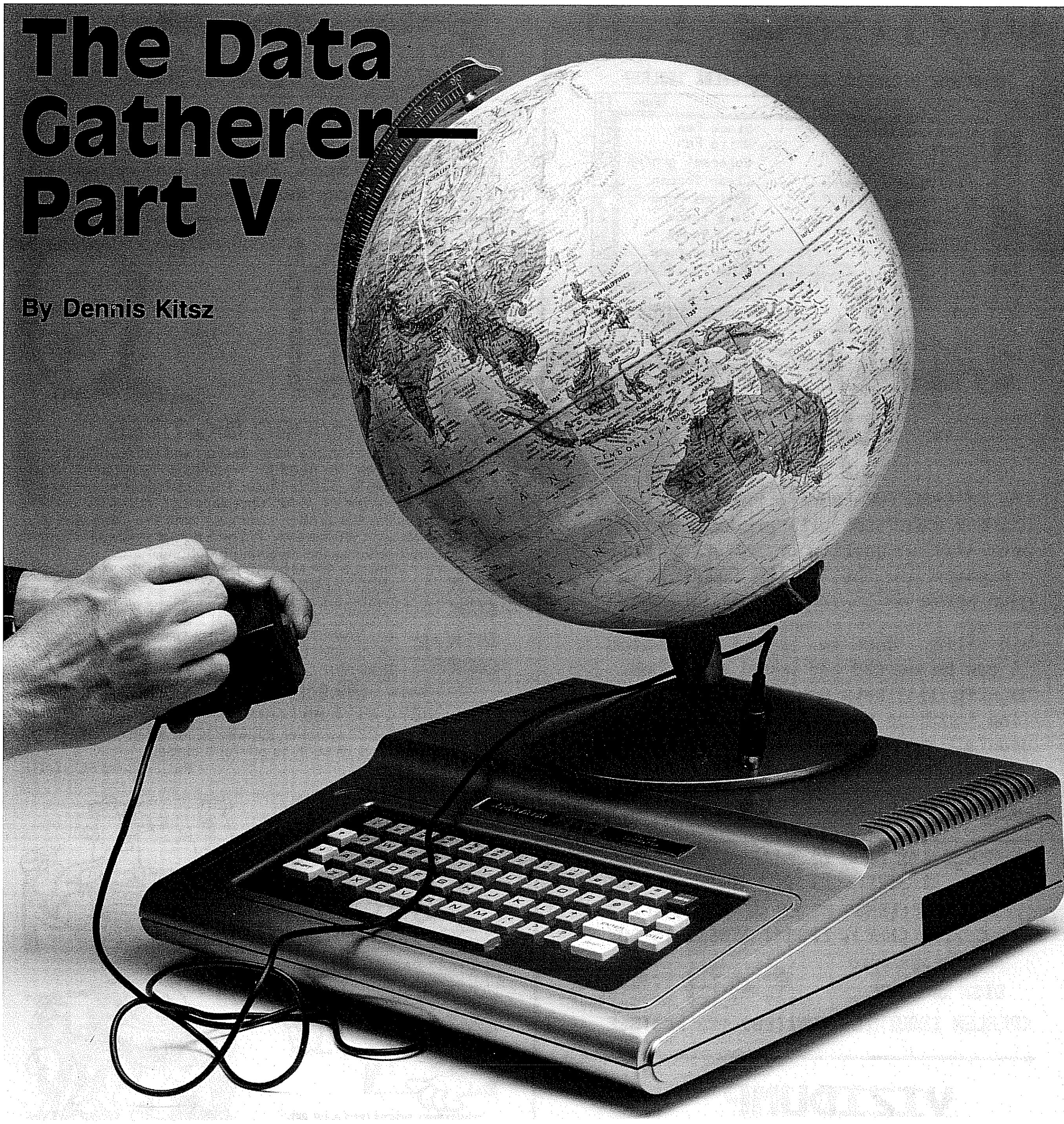
RY - Turns your COCO into a 5 level (Baudot Code) Teletype machine.
$49.95(tape) plus $2 postage

To Place Your Order Send Check or Money Order To:

GRAFX
P.O. Box 254
West Mifflin, Pa. 15122
Voice (412)466-6974 (6-9 PM)
24 Hour BBS-COCONET
(412) 744-2335
300 - 1200 Baud Modem Online
FREE DIGITIZED PICTURES!

The Data Gatherer—Part V

By Dennis Kitsz



I collect throwaway stuff. Not just any throwaway stuff, but things I can get lots of. Like the centers of Scotch tape rolls. Or old typewriter correction ribbon spools. Magic Marker tops are great. Polaroid instant film batteries are terrific. I even have a crate full of IBM carbon ribbon shells from when I worked as a typist—every one has a wheel, two gears, a spring, and a few other doo-dads! All those things are kept in bags . . . my module bags. When I needed hazardous waste containers to strap to a flatcar for my HO trains . . . there they were: Magic Marker modules! Perfect!

That point of view follows with computer hardware and software. Modularity is one of my favorite approaches. I have small boxes of cassettes, each cassette containing an assembly language module identical to the rest in that box. I pick out the modules I require to assemble long programs.

In the same way, I keep hardware modules around; many of the designs presented in my articles are modular, such as the CoCoPort and the clock/calendar.

No one of the modules is particularly interesting by itself, but the whole result sometimes seems different from those mundane-looking parts. The software for the Data Gatherer follows that pattern, containing about a dozen modules. Each unremarkable module is almost self-sufficient, but in sum they make up an effective, elegant and easy-to-use operating system.

The next two columns will present the complete listing for the Data Gatherer Operating System (DGOS), together with commentary, and a guide to using the software from Basic or assembly language.

Quick Look at the Software

The Data Gatherer Operating System (DGOS) is entirely created in assembly language, and hooks into the Basic language of a 32K Extended Basic Color Computer. DGOS contains initialization (and re-initialization) routines, copyright display, a test for Basic program auto-boot, plus a number of modular segments:

- **Module 1**—Analog-to-Digital Conversion

1a. Opens one analog channel.
1b. Converts one analog value to digital form.
1c. Converts 16 analog values to digital form.

- **Module 2**—Digital-to-Analog Conversion

2. Converts one digital value to analog form.

- **Module 3**—Real-Time Clock/Calendar Control

3a. Displays the real-time clock/calendar (RTCC).
3b. Reads and stores the RTCC.
3c. Sets time and date in the RTCC.
3d. Turns on continuous display of the RTCC.
3e. Turns off continuous display of the RTCC.
3f. Swaps to RTCC port addresses.
3g. Swaps to disk port addresses.

- **Module 4**—Parallel Printing

4a. Parallel prints one character.
4b. Parallel prints a string of characters.
4c. Parallel prints the video display.

- **Module 5**—D/A Conversion Alternate

5. Converts a digital value to analog form (faster method).

- **Module 6**—A/D Conversion Alternate

6. Converts an analog value to digital form (faster method).

(I have to admit that because these routines are modular, in the actual program they aren't in precisely the order shown above.)

Listing 1 presents the first half of DGOS. Lines 200–470 identify the memory locations DGOS will use for its operation, and lines 520–720 are the addresses of each of the DGOS subroutine modules, permitting use of assembly language indirect subroutine jumps. This type of reference table allows updating and expansion of the operating system without grief and pain. (For a full discussion of the merits of operating system vectors, see "Op Sys SIG Discussion" in issue 5).

Vanity

Ever wonder how the disk pack performs an autostart when you plug it in, even if you cut loose the so-called "hard start" connection (pins 7 and 8)? If you back up one more step, how does Color Basic know when the Extended Color Basic ROM has been added to the system?

Here's how. When you turn on the power, Color Basic goes through an initialization routine, setting up parameters, testing and reserving memory, and presenting you with a blank green screen. When Color Basic is finished, it examines memory locations $8000 and $8001 (decimal 32768 and 32769) for the values $45 and $58. These are the characters EX, for Extended Basic. If Color Basic finds them, it jumps to Extended Basic at $8002, where a second initialization is performed. When Extended Basic is finished, it looks at locations $C000 and $C001 (decimal 49152 and 49153) for values $44 and $4B. These are characters DK, for Disk Extended Basic. If Extended Basic finds them, it jumps to Disk Basic at $C002, where the third initialization is performed. Finally you get a sign-on message and the OK prompt.

So I took advantage of this DK character search business to give DGOS its autostart . . . and, by using a little deceptive assembly language coding, I was also able to indulge my vanity. Not only did I install the skimpy DK, but I installed a full DKITSZ *(Eds. Note: Groan!)*. When Extended Basic finds the DK and jumps to $C002, it tries to execute the remaining characters (I = $49, T = $54, S = $53 and Z = $5A) which represent the unimportant command sequence ROLA, LSRB, COMB and DECB. That is followed by the real thing, a branch to the actual DGOS initialization (BRA IPL).

The DGOS initialization (IPL, lines 740–830) informs the Basic language operating system that DGOS requires 128 bytes for its use. It does so by changing the values in the "top of memory" pointer, which Basic uses in its clear command; clear is prevented from using memory above $7F80 (decimal 32640). It sets up the stack and observes the remaining Basic set-up sequence. It then initializes DGOS itself.

The DGOS initialization (lines 1200–1640) sets up clock registers and configures the four 8-bit ports of the Data Gatherer. These ports, found at $FF50–$FF5F, are opened, 30 bits are set for output and two bits are set for input. The ports are then closed. A few important values are established immediately (D/A converter channel, high and low D/A converter strobes, printer mask byte, clock screen display, and screen starting location).

The interrupt vector—necessary to provide a continuous clock display on the screen—is "chained" into place. The old vector is used for the sound command, and the clock display vector links itself into the chain of interrupt services. The clock registers are again set, the clock is set up for 24-hour (military) time, and the clock is turned on.

The initialization routine continues with a sign-on message for DGOS, and finally branches to a Basic auto-boot test routine (from line 830). That module is particularly useful by itself, without DGOS, and can auto-start any Basic program into Extended Basic. Next time I'll take a look at that process, which jumps to the Basic ROM at $80B8—the OK prompt—if no program is found.

The first thing you see when you turn the power on, then, is the green screen, the usual pause during 32K memory testing, and the DGOS sign-on message, with real-time clock running in the upper right corner and the OK prompt in the upper left corner.

Klock Kalendar Kludge

Clocks and calendars may seem regular—all that repetitive ticking and such—but when it comes to programming, they're just a mess. Tenths of seconds are base 10, seconds and minutes are base 60, hours are base 24, weeks are base 7, months are base 12, days are base 365, and leap years are base 4 . . . so to speak. Or so to program.

You run into more of that confusion when the clock is exclusively software, but you can't get out of it even with a hardware real-time clock/calendar (RTCC) such as this. The ridiculous results are found in the display routine (lines 2700–3210).

After the screen display start position is identified the routine moves to the subroutine to read the MM58274 clock/calendar (lines 3290–3450), which in turn moves to the swap-to-clock segment (lines 3500–3540). The swap is required you remember, because the clock must share memory space with the disk. By writing 1 to address $FF58, the swap flip-flop accesses the clock, and then returns to GETCLK.

After storing the user's registers, the clock values are moved into a storage area pointed to by the Y register (lines 3310–3390). A little peculiarity follows. Naturally, it takes some time to read the clock, even in assembly language. During this time the clock may have begun a change cycle. If that change is at the hour or the day, perhaps, the software will get half of the old time and half of the new time. The result will be a very wrong mixture of time value.

A safeguard is provided, however; it is the "data changed" flag in the RTCC chip. If bit 3 of address 0 in the clock is high, then the clock has been updated . . . so the assembly language program has to re-read the clock for a correct value. Finally it can return to the display routine.

The actual displaying picks up at line 2730, where X is pointed to the screen display position, saved, and re-pointed to the day-of-week value in the clock storage area. Y points to the table of weekdays, beginning with Sunday. The value is plucked from the table and displayed, one character at a time, followed by a space (lines 2820–2920).

The numbers are then pulled from the clock storage area and displayed: the year, a space, the month, a slash, the date, a space, the hour, a colon, the minute, a colon, the second, a period, and the tenths of seconds. Each value (only four bits in size) is stripped of its excess bits and converted for VDG display. (The conversion is unusual because the VDG does not display characters according to their proper ASCII codes. For more on the VDG, refer to *The Color Computer Magazine,* August 1983).

Finally, 166 bytes, 903 machine cycles or 1.01 milliseconds later, the routine is ready to return to the program at hand. And remember of course, that when there is a continuous on-screen clock display, this is an interrupt service routine, adding a shade more time to the process. The overall result is about just under a 5 percent loss of computer speed.

Out the Converter

The output of a 12-bit value through the digital-to-analog converter is quite fast, suitable for professional sounding single-voice music synthesis or high-speed lab or motor control applications. The routine begins at line 2460.

Recall that the AD667 D/A converter is versatile, capable of interfacing with 4-, 8-, 12- and 16-bit microcomputer systems. It does this by using three independent 4-bit latches preceding a master 12-bit output latch, in addition to a 4-bit control latch. Therefore, the 12-bit data and control word can be installed in many combinations: as four 4-bit nybbles, as two 8-bit bytes, or as a complete 16-bit word.

The Color Computer is an 8-bit system. So at line 2480, 8 bits are placed into the D/A converter, and the remaining four bits—plus four control bits—are stored in the converter at 2510. The strobe routine (2610) pulses the D/A converter to accept the data being made available to it; at lines 2560–2570, those 12 bits are strobed through to the master latch and out the converter.

This is a very orderly subroutine; by sacrificing some of that orderliness, a faster routine can be created (later for this).

Cornering the Prey

There are times when I realize I've got a lot more to learn as a programmer. Writing the subroutine to input an analog value through the Data Gatherer by successive approximation brought that weakness into focus once again. In fact, only an intuitive sense tells me there must be an easier way. Readers? Help!

A	B	SX1	SY1	SX2	SY2	Channel Number	Select (Hex)	Channel (Hex)
x	x	0	0	0	0	None	00	—
0	0	0	0	0	1	0	01	00
0	1	0	0	0	1	1	11	01
1	0	0	0	0	1	2	21	02
1	1	0	0	0	1	3	31	03
0	0	0	0	1	0	4	02	04
1	0	0	0	1	0	6	22	06
1	1	0	0	1	0	7	32	07
0	0	0	1	0	0	8	04	08
0	1	0	1	0	0	9	14	09
1	0	0	1	0	0	10	24	0A
1	1	0	1	0	0	11	34	0B
0	0	1	0	0	0	12	08	0C
0	1	1	0	0	0	13	18	0D
1	0	1	0	0	0	14	28	0E
1	1	1	0	0	0	15	38	0F

Table 1. Output Arrangement

First, let me tell you what I did (clumsy but successful). The conversion routine begins at line 2030, where a loop is set up: X points to the location where the working offset will be stored, Y points to the location where the 12-bit "known" value will be stored, A accepts the count value and stores it, and D is established as the midpoint between low and high values (midway between 0 and 4095). With these parameters in place, the successive approximation begins at line 2110.

The first value is output through the D/A converter, and the status of the comparator is read (COMPIN) and stored for reference (LASTIN). Remember that the unknown value is discovered by comparing it with a known value, which is what the hardware voltage comparator does.

So, if the hardware voltage comparator reads 1, then the unknown value is greater than the known value at the output of the D/A converter. If the comparator reads 0, then the unknown is less than the known value. ROLA (rotate-left-accumulator, line 2150) forces the value of bit 7 into the carry flag. Based on the contents of the carry flag, the subroutine either adds half again to the known value (2270) or subtracts half again from the known value (2170).

The count is decremented (2340) and the process repeated 12 times until the known value homes in on the unknown value. Finally, depending on the final state of the carry flag, the result is adjusted for known = unknown, against known ► unknown. The final value is stored, and also returned in the D register.

What tells me there must be a better way? Something about the adding and subtracting tells me . . . there's just too much of it, and it feels clumsy. The pattern of bits in the known value seems very regular, but I can't put my finger on it. Do you see it? If so, let me know. Either a free Data Gatherer PC board or a free dinner in Roxbury, VT awaits the first person who can rewrite that routine using bit manipulation—and who can get a single conversion down to 1 mS or less!

And the Rest

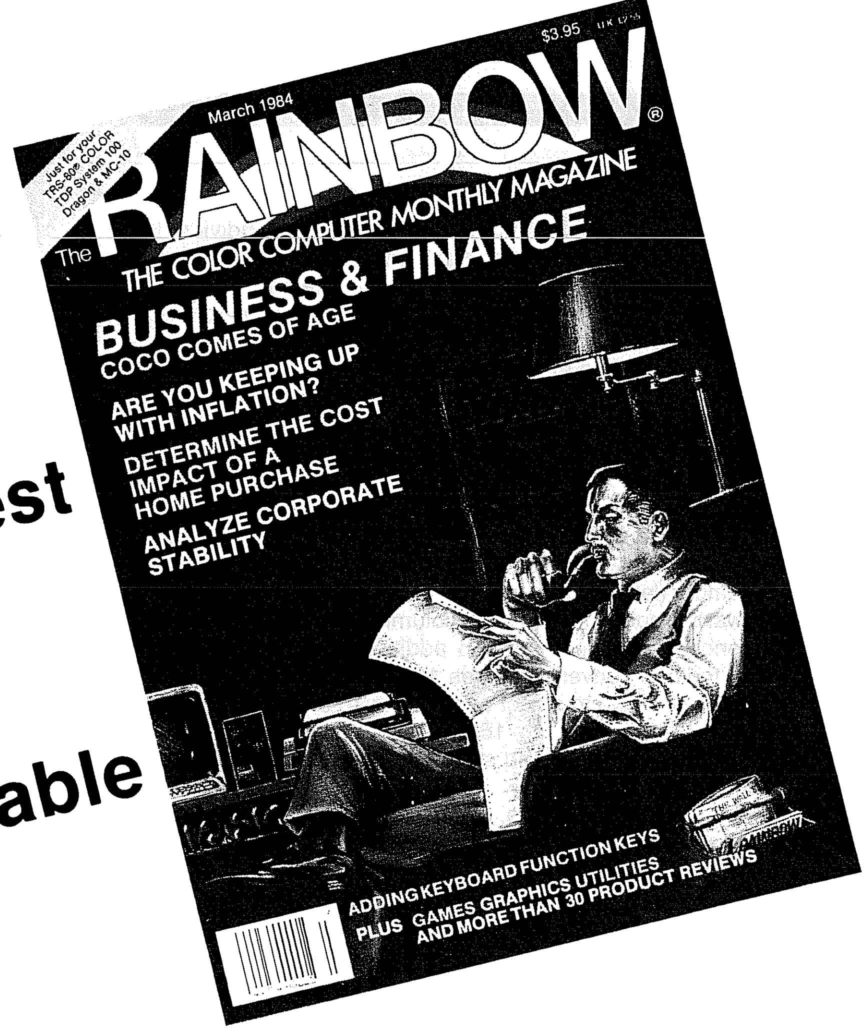
The rest of the A/D conversion section involves turning on the channel to be converted, and converting all 16 channels.

The channel selection routine is messy because of the nature of the analog multiplexer; Table 1 is the arrangement of output again.

The Biggest

The Best

The Indispensable



The latest news on the Color Computer grapevine is that more and more people are discovering **The Rainbow.®**

Now in its third year, **The Rainbow** has become the standard by which other Color Computer magazines are compared. And no wonder! **The Rainbow** towers above the crowd, now offering up to 300 pages each month, including as many as two dozen type-in-and-run program listings, a host of articles and in excess of 30 hardware and software product reviews.

We lead the pack in Color Computer publications and are devoted *exclusively* to the TRS-80® Color, TDP-100 and Dragon-32. We made our climb to the top by continually offering the best and the most by such well-known authors and innovators as Bob Albrecht and Don Inman, and games from top programmers like Robert Tyson, Fred Scerbo and John Fraysse. **The Rainbow** offers the most in entertainment and education, home uses, technical details and hardware projects, tutorials, utilities, graphics and special features like Rainbow Scoreboard and our CoCo Clubs section.

For only $28* a year, you get the keys to all the secrets locked in your CoCo!

Are you searching through the jungle of claims and clamor? Climb above it all. Look up. Find **The Rainbow.**

THE RAINBOW
(502) 228-4492

9529 U.S. Highway 42
The Falsoft Bldg.
P.O. Box 385
Prospect, KY 40059

THE RAINBOW is a publication of Falsoft, Inc.

YES! Sign me up for a year (12 issues) of **THE RAINBOW.**

Name ______________________

Address ______________________

City ____________ State ____________ ZIP ________

Payment Enclosed ☐

Charge Visa ☐ MasterCard ☐ American Express ☐

My Account# ______________________

Signature ____________ Card Expiration Date ________

In order to hold down non-editorial costs, we do not bill for subscriptions.

*Subscriptions to **the RAINBOW** are $28 ($31 after Jan. 1, 1985) a year in the United States. Canadian rate $35 U.S. ($38 after Jan. 1, 1985). Surface rate to other countries $65 U.S. ($68 after Jan. 1, 1985); air rate $100 U.S. ($103 after Jan. 1, 1985). All subscriptions begin with the current issue. Please allow up to 5-6 weeks for first copy. U.S. FUNDS only.

Prices subject to change.

Recall the Basic program used for testing the individual channel input? The Select column in the Table reveals where those mysterious looking data statements were derived from. Depending on the state of the X and Y select lines, the values of A and B determine which input are fed through the analog multiplexers. Notice that only one X or Y select line may be turned on at a time; the software must respect this or risk opening two input at once and feeding input into each other! (The initialization routine establishes channel 1 at power-up, just to be safe.)

The channel routine (line 1810) gets the value from MXCHAN (1 to 16), puts it in the range 0 to 15, and muscles it into the format shown above. Compare the adjusted channel number (last column) with the value needed to select the hardware properly (next-to-last column). You can follow the sequence of logical shifts and additions that perform the rather tedious conversion (lines 1840–1980).

Finally, the loop to select each of the 16 channels in turn is shown beginning at line 1660. This is a simple loop, storing the channel value, and storing the 16 input values in a block of memory.

Next time: Print routines, clock on and off routines, autoboot Basic from ROM, and application ideas. *(end)*

Program. Half of The Listing!

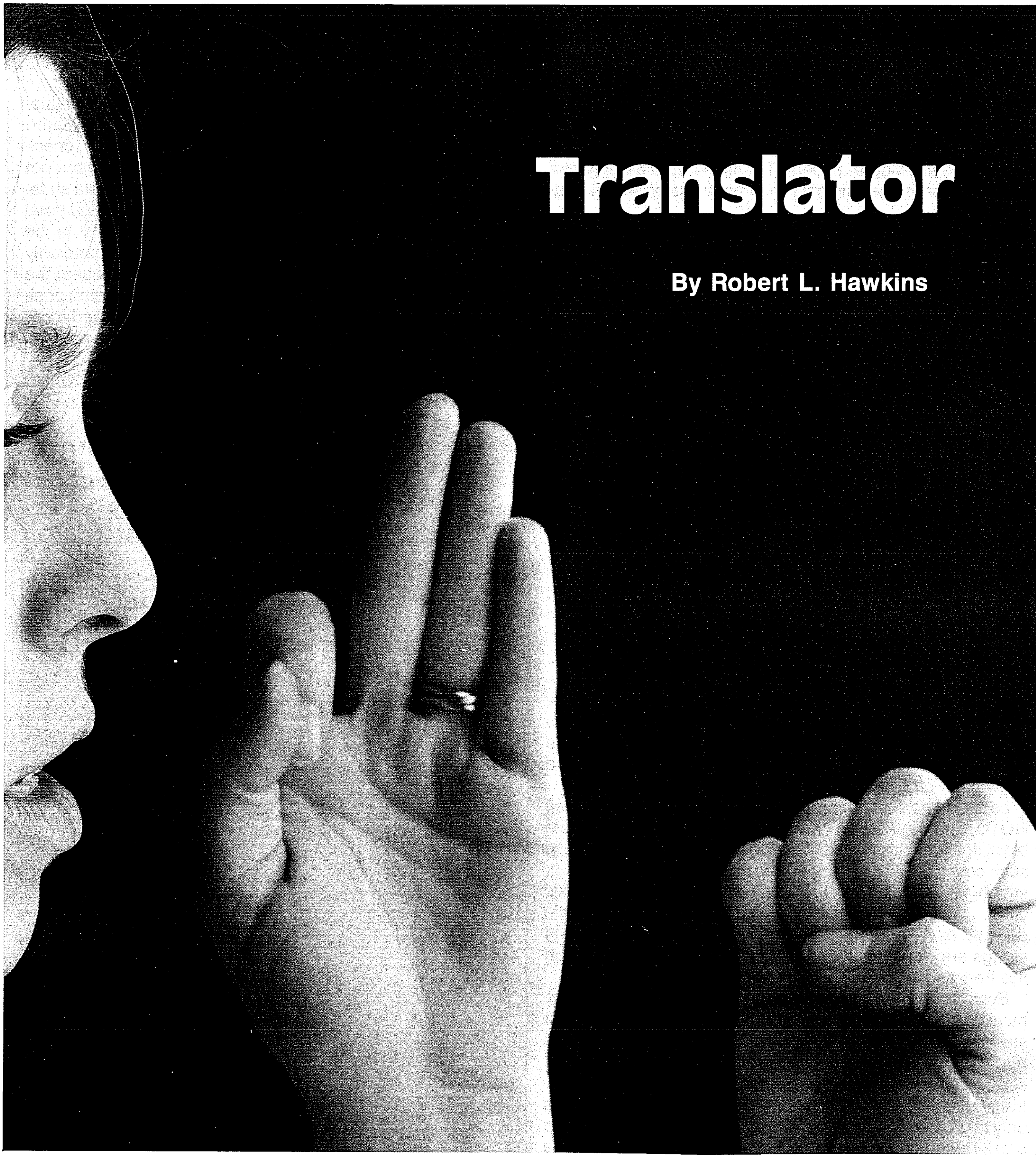
```
                00100 **************************
                00110 *    The Data Gatherer    *
                00120 *                         *
                00130 *   Copyright (c) 1984    *
                00140 *  Dennis Bathory Kitsz   *
                00150 *  All Rights Reserved.   *
                00160 *                         *
                00170 **************************
                00180 *
                00190 *
        7F82    00200 MVALUE  EQU   $7F82    * Final 12-bit value
        7F84    00210 OFFSET  EQU   $7F84    * Working offset status
        7F86    00220 VALHI   EQU   $7F86    * Value to output via DAC
        7F88    00230 STRSTR  EQU   $7F88    * Addr. of string to print
        7F8A    00240 SSTART  EQU   $7F8A    * Start of video screen
        7F8C    00250 DISPLC  EQU   $7F8C    * Clock display location
        7F8E    00260 MXCHAN  EQU   $7F8E    * Channel of mux to open
        7F8F    00270 HISTRB  EQU   $7F8F    * Hi DAC strobe pulse
        7F90    00280 LOSTRB  EQU   $7F90    * Lo DAC strobe pulse
        7F91    00290 COUNT   EQU   $7F91    * Number of output bits
        7F92    00300 LASTIN  EQU   $7F92    * Last comparator input
        7F93    00310 SAVE    EQU   $7F93    * DAC working area
        7F94    00320 MASK    EQU   $7F94    * Printer input mask
        7F95    00330 STASH   EQU   $7F95    * Working area for ADC
        7F96    00340 PRINTC  EQU   $7F96    * Character to print
        7F97    00350 INTVEC  EQU   $7F97    * Interrupt vector
                00360 *
        7F99    00370 INTJMP  EQU   $7F99    * Op sys vector point
        7F9B    00380 STOR32  EQU   $7F9B    * 16 2-bytes in from DAC
        7FBB    00390 CLKSAV  EQU   $7FBB    * 14-byte clock store/set
                00400 *
C000            00410         ORG   $C000    * First location in ROM
                00420 *
        FF40    00430 CLOCK   EQU   $FF40    * Clock $FF40-$FF4F
        FF50    00440 PORT    EQU   $FF50    * DAC Port $FF50-$FF52
        FF54    00450 COMPIN  EQU   $FF54    * Comparator input
        FF58    00460 SWAP    EQU   $FF58    * Disk/clock swap latch
        010D    00470 INTER   EQU   $010D    * Basic interrupt vector
                00480 *
C000    44      00490 AUTO    FCC   'DKITSZ'       * DK = autostart
        4B
        49
        54
        53
        5A
C006 20 2A      00500         BRA   IPL      * Autostart to IPL
                00510 *
C008    C032    00520 XIPL    FDB   IPL      * I
C00A    C04B    00530 XINIT   FDB   INIT     * N
C00C    C08A    00540 XRENIT  FDB   REINIT   * D
C00E    C111    00550 XCHANL  FDB   CHANNL   * I
C010    C135    00560 XCONVT  FDB   CONVRT   * R
C012    C10C    00570 XGETVL  FDB   GETVAL   * E
C014    C0F6    00580 XGET16  FDB   GET16    * C
C016    C18E    00590 XDACOT  FDB   DACOUT   * T
C018    C362    00600 XFSTAD  FDB   FASTAD   *
C01A    C331    00610 XFSTDA  FDB   FASTDA   * J
C01C    C262    00620 XCLKST  FDB   CLKSET   * U
C01E    C1BC    00630 XDISCK  FDB   DISCLK   * M
C020    C2FA    00640 XCLKON  FDB   CLKON    * P
C022    C30F    00650 XCLKOF  FDB   CLKOFF   *
C024    C230    00660 XGETCK  FDB   GETCLK   * V
C026    C281    00670 XPRNT1  FDB   PRINT1   * E
C028    C299    00680 XPRNTS  FDB   PRINTS   * C
C02A    C2BF    00690 XPRNTV  FDB   PRINTV   * T
C02C    C253    00700 XSWAPD  FDB   SWAPD    * O
C02E    C258    00710 XSWAPC  FDB   SWAPC    * R
C030    C051    00720 XCPYRT  FDB   COPYRT   * S
                00730 *
C032 8E 7F80    00740 IPL     LDX   #$7F80   * Protected Op Sys memory
C035 9F 74      00750         STX   <$74     * Maximum system memory
C037 9F 27      00760         STX   <$27     * Basic high memory
C039 9F 23      00770         STX   <$23     * Basic high memory
C03B 30 89 FF38 00780         LEAX  -200,X   * Clear 200 bytes
C03F 9F 21      00790         STX   <$21     * Into clear area
C041 1F 14      00800         TFR   X,S      * Set up stack for Basic
C043 BD AD19    00810         JSR   $AD19    * Observe Basic setups
C046 8D 03      00820         BSR   INIT     * Set up Op Sys from ROM
C048 16 038D    00830         LBRA  ATEST    * And go to Basic OK
                00840 *
C04B 8D 3D      00850 INIT    BSR   REINIT   * Initialize routine first
C04D 8D 02      00860         BSR   COPYRT   * Then copyright message
C04F 39         00870         RTS            * Back to calling program
                00880 *
C050 39         00890 STUFF   RTS            * For later expansion
```

```
                        00900 *
C051 10BE 7F8A          00910 COPYRT  LDY     SSTART  * Get screen starting point
C055 30   8D 03E5       00920         LEAX    SIGNON,PCR      * Point to sign-on
C059 31   A8 60         00930         LEAY    $60,Y   * Move three lines down
C05C 8D   01            00940         BSR     VIDEO   * And print message
C05E 39                 00950         RTS             * Back to calling program
                        00960 *
C05F A6   84            00970 VIDEO   LDA     ,X      * Get first character
C061 81   00            00980         CMPA    #$00    * Is it zero?
C063 27   24            00990         BEQ     OUTIPL  * If so, end of message
C065 81   0D            01000         CMPA    #$0D    * Is it carriage return?
C067 27   20            01010         BEQ     OUTIPL  * If so, end of message
C069 1F   89            01020         TFR     A,B     * Begin ASCII to VDG conv.
C06B C4   E0            01030         ANDB    #$E0
C06D 27   12            01040         BEQ     JDOIT
C06F C1   40            01050         CMPB    #$40
C071 27   0E            01060         BEQ     JDOIT
C073 C1   60            01070         CMPB    #$60
C075 26   04            01080         BNE     JNEXT
C077 84   3F            01090         ANDA    #$3F
C079 20   06            01100         BRA     JDOIT
C07B C1   20            01110 JNEXT   CMPB    #$20
C07D 26   02            01120         BNE     JDOIT
C07F 8B   40            01130         ADDA    #$40
C081 A7   A4            01140 JDOIT   STA     ,Y      * Store char. to screen
C083 30   01            01150         LEAX    1,X     * Move to next character
C085 31   21            01160         LEAY    1,Y     * Move to next screen posn.
C087 20   D6            01170         BRA     VIDEO   * And repeat till done
C089 39                 01180 OUTIPL  RTS             * Back to calling program
                        01190 *
C08A 17   01CB          01200 REINIT  LBSR    SWAPC   * Swap to clock mode
C08D 4F                 01210         CLRA            * Set A to zero
C08E B7   FF40          01220         STA     $FF40   * Set clock registers
C091 17   01BF          01230         LBSR    SWAPD   * Swap back to disk
C094 8E   FF50          01240         LDX     #PORT   * Point to DAC port
C097 86   38            01250         LDA     #$38    * Open port value
C099 A7   01            01260         STA     1,X     * Open port $FF50
C09B A7   03            01270         STA     3,X     * Open port $FF52
C09D A7   05            01280         STA     5,X     * Open port $FF54
C09F A7   07            01290         STA     7,X     * Open port $FF56
C0A1 86   FF            01300         LDA     #$FF    * Get all bits output
C0A3 A7   84            01310         STA     ,X      * All bits output $FF50
C0A5 A7   02            01320         STA     2,X     * All bits output $FF52
C0A7 A7   06            01330         STA     6,X     * All bits output $FF56
C0A9 86   3F            01340         LDA     #$3F    * Six bits output
C0AB A7   04            01350         STA     4,X     * Six bits output $FF54
C0AD 86   3C            01360         LDA     #$3C    * Close port value
C0AF A7   01            01370         STA     1,X     * Close port $FF50
C0B1 A7   03            01380         STA     3,X     * Close port $FF52
C0B3 A7   05            01390         STA     5,X     * Close port $FF54
C0B5 A7   07            01400         STA     7,X     * Close port $FF56
C0B7 86   01            01410         LDA     #$01    * Get channel #1 value
C0B9 B7   7F8E          01420         STA     MXCHAN  * Prepare channel #1 on
C0BC 86   3C            01430         LDA     #$3C    * Get DAC high strobe
C0BE B7   7F8F          01440         STA     HISTRB  * And put in reference
C0C1 86   34            01450         LDA     #$34    * Get DAC low strobe
C0C3 B7   7F90          01460         STA     LOSTRB  * And put in reference
C0C6 86   C0            01470         LDA     #$C0    * Get 6-bit print mask
C0C8 B7   7F94          01480         STA     MASK    * And put in reference
C0CB CC   0409          01490         LDD     #$0409  * Get clock screen pointer
C0CE FD   7F8C          01500         STD     DISPLC  * And put in reference
C0D1 CC   0400          01510         LDD     #$0400  * Get normal screen pointer
C0D4 FD   7F8A          01520         STD     SSTART  * And put in reference
C0D7 30   8D 0244       01530         LEAX    INTDIS,PCR      * Get int. vector
C0DB BF   7F97          01540         STX     INTVEC  * And put in reference
C0DE 17   0177          01550         LBSR    SWAPC   * Swap to clock mode
C0E1 4F                 01560         CLRA            * Set A to zero
C0E2 B7   FF40          01570         STA     CLOCK   * Make sure ctrl. reg. okay
C0E5 B6   FF4F          01580         LDA     CLOCK+15        * Point to set reg.
C0E8 84   0C            01590         ANDA    #$0C    * Mask in bits 2 & 3
C0EA 8A   01            01600         ORA     #$01    * Set bit 0
C0EC B7   FF4F          01610         STA     CLOCK+15        * Clock = 24 hrs
C0EF 17   0161          01620         LBSR    SWAPD   * Swap back to disk
C0F2 17   0205          01630         LBSR    CLKON   * Turn clock on
C0F5 39                 01640         RTS             * Back to calling program
                        01650 *
C0F6 86   10            01660 GET16   LDA     #$10    * Get 16 channels ready
C0F8 B7   7F8E          01670         STA     MXCHAN  * Start with channel 16
C0FB 8E   7F9B          01680         LDX     #STOR32 * Point X to storage
C0FE 34   10            01690 NEXT16  PSHS    X       * Save storage pointer
C100 8D   0A            01700         BSR     GETVAL  * Get one value from ADC
C102 35   10            01710         PULS    X       * Get storage pointer
C104 ED   81            01720         STD     ,X++    * Store value and bump
C106 7A   7F8E          01730         DEC     MXCHAN  * Get next channel
C109 26   F3            01740         BNE     NEXT16  * If not zero, continue
C10B 39                 01750         RTS             * Back to calling program
                        01760 *
C10C 8D   03            01770 GETVAL  BSR     CHANNL  * Turn on channel
C10E 8D   25            01780         BSR     CONVRT  * Do analog/digital conv.
C110 39                 01790         RTS             * Back to calling program
                        01800 *
C111 B6   7F8E          01810 CHANNL  LDA     MXCHAN  * Get value for channel
C114 4A                 01820         DECA            * Strip offset
C115 1F   89            01830         TFR     A,B     * Put into B for work
C117 C4   03            01840         ANDB    #$03    * Mask in bits 0 and 1
C119 58                 01850         LSLB            * And move...
C11A 58                 01860         LSLB            * ... more ...
C11B 58                 01870         LSLB            * ... more ...
C11C 58                 01880         LSLB            * ... until LSB is MSB
C11D F7   7F93          01890         STB     SAVE    * And save for later
C120 44                 01900         LSRA            * Divide by two ...
C121 44                 01910         LSRA            * ... and two again
C122 4C                 01920         INCA            * And bump up value
C123 C6   01            01930         LDB     #$01    * Start with bit 0 set
C125 4A                 01940 NEXT    DECA            * And begin movement
C126 27   03            01950         BEQ     GOTIT   * If done then mux in place
C128 58                 01960         LSLB            * Else move B over one
C129 20   FA            01970         BRA     NEXT    * And do it again
C12B FB   7F93          01980 GOTIT   ADDB    SAVE    * B + SAVE = Combination
C12E FA   7F94          01990         ORB     MASK    * And strip input bits
C131 F7   FF54          02000         STB     $FF54   * And turn on channel
C134 39                 02010         RTS             * Back to calling program
                        02020 *
C135 8E   7F84          02030 CONVRT  LDX     #OFFSET *
C138 108E 7F82          02040         LDY     #MVALUE * S
C13C 86   0C            02050         LDA     #$0C    * U
C13E B7   7F91          02060         STA     COUNT   * C
C141 CC   0800          02070         LDD     #$0800  * C
C144 ED   84            02080         STD     ,X      * E
C146 83   0001          02090         SUBD    #$0001  * S
```

```
C149 ED   A4      02100         STD     ,Y      * S
C14B FD   7F86    02110 AGAIN   STD     VALHI   * I
C14E 8D   3E      02120         BSR     DACOUT  * V
C150 B6   FF54    02130         LDA     COMPIN  * E
C153 B7   7F92    02140         STA     LASTIN  *
C156 49           02150         ROLA            * A
C157 24   13      02160         BCC     UGTK    * P
C159 EC   84      02170         LDD     ,X      * P
C15B 44           02180         LSRA            * R
C15C 56           02190         RORB            * O
C15D ED   84      02200         STD     ,X      * X
C15F EC   A4      02210         LDD     ,Y      * I
C161 A3   84      02220         SUBD    ,X      * M
C163 ED   A4      02230         STD     ,Y      * A
C165 7A   7F91    02240         DEC     COUNT   * T
C168 26   E1      02250         BNE     AGAIN   * I
C16A 20   11      02260         BRA     DONE    * O
C16C EC   84      02270 UGTK    LDD     ,X      * N
C16E 44           02280         LSRA            *
C16F 56           02290         RORB            * A
C170 ED   84      02300         STD     ,X      * /
C172 EC   A4      02310         LDD     ,Y      * D
C174 E3   84      02320         ADDD    ,X      *
C176 ED   A4      02330         STD     ,Y      * C
C178 7A   7F91    02340         DEC     COUNT   * O
C17B 26   CE      02350         BNE     AGAIN   * N
C17D B6   7F92    02360 DONE    LDA     LASTIN  * V
C180 49           02370         ROLA            * E
C181 25   08      02380         BCS     EXIT    * R
C183 EC   A4      02390         LDD     ,Y      * S
C185 C3   0001    02400         ADDD    #$0001  * I
C188 ED   A4      02410         STD     ,Y      * O
C18A 39           02420         RTS             * N
C18B EC   A4      02430 EXIT    LDD     ,Y      *
C18D 39           02440         RTS             *
                  02450 *
C18E FC   7F86    02460 DACOUT  LDD     VALHI   * Get value to send
C191 34   10      02470         PSHS    X       * Save X from caller
C193 8E   FF50    02480         LDX     #PORT   * Point to DAC output
C196 E7   84      02490         STB     ,X      * Put LSB into DAC
C198 8A   80      02500         ORA     #$80    * Set to strobe 3 nybbles
C19A A7   02      02510         STA     2,X     * And values are ready
C19C 8D   0D      02520         BSR     STROBE  * Strobe them into DAC
C19E 8A   F0      02530         ORA     #$F0    * Set all bits high
C1A0 A7   02      02540         STA     2,X     * And put them in place
C1A2 84   7F      02550         ANDA    #$7F    * Ready to do conversion
C1A4 A7   02      02560         STA     2,X     * Put convert command in
C1A6 8D   03      02570         BSR     STROBE  * And strobe through DAC
C1A8 35   10      02580         PULS    X       * Restore caller's X
C1AA 39           02590         RTS             * Back to calling program
                  02600 *
C1AB 34   06      02610 STROBE  PSHS    A,B     * Save caller's A & B
C1AD B6   7F8F    02620         LDA     HISTRB  * Get high strobe value
C1B0 A7   01      02630         STA     1,X     * Strobe DAC high
C1B2 F6   7F90    02640         LDB     LOSTRB  * Get low strobe value
C1B5 E7   01      02650         STB     1,X     * Strobe DAC low
C1B7 A7   01      02660         STA     1,X     * Strobe DAC high
C1B9 35   06      02670         PULS    A,B     * Restore caller's A & B
C1BB 39           02680         RTS             * Back to calling program
                  02690 *
```

```
C1BC FC   7F8C    02700 DISCLK  LDD     DISPLC  * Get screen display posn.
C1BF 8D   6F      02710         BSR     GETCLK  * Get clock into memory
C1C1 34   30      02720         PSHS    X,Y     * Stash caller's X & Y
C1C3 1F   01      02730         TFR     D,X     * Point X to display
C1C5 34   10      02740         PSHS    X       * And store it for use
C1C7 8E   7FBB    02750         LDX     #CLKSAV * Point X to GETCLK result
C1CA 31   8D 025B 02760         LEAY    DAYS,PCR     * ASCII day table
C1CE A6   84      02770         LDA     ,X      * Get first GETCLK value
C1D0 4A           02780         DECA            * Prepare 1-7 becomes 0-6
C1D1 C6   03      02790         LDB     #$03    * 3 ASCII chars. per day
C1D3 3D           02800         MUL             * 3*0 thru 3*6 for offset
C1D4 35   10      02810         PULS    X       * Restore X for use
C1D6 A6   A5      02820         LDA     B,Y     * Display
C1D8 31   21      02830         LEAY    1,Y     *  first
C1DA A7   80      02840         STA     ,X+     *  character
C1DC A6   A5      02850         LDA     B,Y     * Display
C1DE 31   21      02860         LEAY    1,Y     *  second
C1E0 A7   80      02870         STA     ,X+     *  character
C1E2 A6   A5      02880         LDA     B,Y     * Display
C1E4 31   21      02890         LEAY    1,Y     *  third
C1E6 A7   80      02900         STA     ,X+     *  character
C1E8 C6   60      02910         LDB     #$60    * Get VDG space value
C1EA E7   80      02920         STB     ,X+     * And display a space
C1EC 108E 7FBB    02930         LDY     #CLKSAV * Point to GETCLK result
C1F0 31   21      02940         LEAY    1,Y     * And bump to next value
C1F2 8D   33      02950         BSR     LICK    * Get, mask, display 10Y
C1F4 8D   31      02960         BSR     LICK    * Get, mask, display 1Y
C1F6 E7   80      02970         STB     ,X+     * Another space
C1F8 8D   2D      02980         BSR     LICK    * Get, mask, display 10M
C1FA 8D   2B      02990         BSR     LICK    * Get, mask, display 1M
C1FC 86   6F      03000         LDA     #$6F    * Get a VDG slash
C1FE A7   80      03010         STA     ,X+     * Display slash
C200 8D   25      03020         BSR     LICK    * Get, mask, display 10D
C202 8D   23      03030         BSR     LICK    * Get, mask, display 1D
C204 E7   80      03040         STB     ,X+     * Display another slash
C206 A6   A0      03050         LDA     ,Y+     * Get next character
C208 84   03      03060         ANDA    #$03    * Mask AM/PM indicator
C20A 8B   70      03070         ADDA    #$70    * Mask for VDG
C20C A7   80      03080         STA     ,X+     * And display 10H
C20E 8D   17      03090         BSR     LICK    * Get, mask, display 1H
C210 C6   7A      03100         LDB     #$7A    * Get a colon
C212 E7   80      03110         STB     ,X+     * And display it
C214 8D   11      03120         BSR     LICK    * Get, mask, display 10M
C216 8D   0F      03130         BSR     LICK    * Get, mask display 1M
C218 E7   80      03140         STB     ,X+     * Display another colon
C21A 8D   0B      03150         BSR     LICK    * Get, mask, display 10S
C21C 8D   09      03160         BSR     LICK    * Get, mask, display 1S
C21E 86   6E      03170         LDA     #$6E    * Get a VDG period
C220 A7   80      03180         STA     ,X+     * And display it
C222 8D   03      03190         BSR     LICK    * Get, mask, display 1/10S
C224 35   30      03200         PULS    X,Y     * Restore caller's X & Y
C226 39           03210         RTS             * Back to calling program
                  03220 *
C227 A6   A0      03230 LICK    LDA     ,Y+     * Get value from clock
C229 84   0F      03240         ANDA    #$0F    * Strip off high nybble
C22B 8B   70      03250         ADDA    #$70    * Mask for VDG display
C22D A7   80      03260         STA     ,X+     * Display the value
C22F 39           03270         RTS             * Back to calling program
                  03280 *
C230 8D   26      03290 GETCLK  BSR     SWAPC   * Swap to clock mode
```

```
C232 34   36              PSHS  X,Y,D    * Stash some registers
C234 8E   FF40    REDO1   LDX   #CLOCK   * Point to clock port
C237 108E 7FBB            LDY   #CLKSAV  * Point to storage area
C23B A6   84              LDA   ,X       * Reset "data changed" flag
C23D C6   0E              LDB   #$0E     * Number of registers
C23F A6   85      REDO2   LDA   B,X      * Get $0Eth register
C241 84   0F              ANDA  #$0F     * Mask in low nybble
C243 A7   A0              STA   ,Y+      * Put into storage
C245 5A                   DECB           * Done with all registers?
C246 26   F7              BNE   REDO2    * If not, then get next
C248 A6   84              LDA   ,X       * Read "data changed" flag
C24A 84   08              ANDA  #$08     * Mask in "d.c." bit
C24C 26   E6              BNE   REDO1    * If data changed, do again
C24E 35   36              PULS  X,Y,D    * Restore the stuff
C250 8D   01              BSR   SWAPD    * Swap to disk mode
C252 39                   RTS            * Back to calling program
                  *
C253 34   02      SWAPD   PSHS  A        * Save caller's A
C255 4F                   CLRA           * Set A to zero (disk mode)
C256 20   04              BRA   SWAPX    * Store in swap flip-flop
C258 34   02      SWAPC   PSHS  A        * Store caller's A
C25A 86   01              LDA   #$01     * Get A=1 (clock mode)
C25C B7   FF58    SWAPX   STA   $FF58    * Store in swap flip-flop
C25F 35   02              PULS  A        * Restore caller's A
C261 39                   RTS            * Back to calling program
                  *
C262 8D   F4      CLKSET  BSR   SWAPC    * Swap to clock mode
C264 34   36              PSHS  X,Y,D    * Store caller's stuff
C266 8E   FF40    REDO3   LDX   #CLOCK   * Point to clock registers
C269 10BE 7F88            LDY   STRSTR   * Point to setting address
C26D C6   0E              LDB   #$0E     * Number of clock registers
C26F A6   A0      REDO4   LDA   ,Y+      * Get first new value
C271 A7   85              STA   B,X      * Put into clock register
C273 5A                   DECB           * Decrement register count
C274 26   F9              BNE   REDO4    * Go back for next
C276 A6   84              LDA   ,X       * Else read "data changed" flag
C278 84   08              ANDA  #$08     * Mask in "d.c." bit
C27A 26   EA              BNE   REDO3    * If changed, do it again
C27C 35   36              PULS  X,Y,D    * Restore caller's stuff
C27E 8D   D3              BSR   SWAPD    * Swap back to disk mode
C280 39                   RTS            * Back to calling program
                  *
```



Translator

By Robert L. Hawkins

Fortran was the first of the high-level computer languages. It was designed primarily for scientific and mathematical computing, and in these areas a huge library of useful subroutines is available. The routines may be found in statistics, numerical methods, and data analysis textbooks, and as part of mainframe and minicomputer on-line libraries. Many routines, especially those written in the days when mainframe computers had the speed and memory size of current-day micros (or less!), were lovingly crafted for efficient use of limited memory, and are ideal for microcomputer use. The subroutines would have been written in Fortran II (up to about 1963) or, in most cases, Fortran IV (the standard version, from 1963 to about 1977).

An option for folks who can't implement Fortran is to translate the subroutines into Basic. Listing 1 performs the most tedious and error-prone parts of the job automatically. It takes a Fortran routine saved as an ASCII file as input, and adds Basic line numbers, translates logical and arithmetic If statements and GOTO statements, and transforms Do loops into For-Next loops. It translates some Fortran keywords and function names, and flags other statements for further translation (I/O statements, subroutine calls). The translated routine is written out as a Basic program in ASCII format.

Running

Create the original ASCII file with a word processor or by downloading the Fortran routine from a mainframe or minicomputer using a modem. The file should be only one routine; Fortran line numbers may be repeated in different routines.

The listing will ask for a five-digit line number to begin with, and Basic line numbering will increase in steps of 10 from this point.

The Fortran source will then load from cassette and be translated. You may send the translated program to tape or the printer, or just view it on the screen.

Listing 2 is the translated output of the sample Fortran source, provided for example.

The translated Basic version will be saved to tape. You will probably have to complete the translation process manually. Some Fortran statements are simply untranslatable: for example, there is no call subroutine in Basic. You will have to replace calls with GOSUBs, and resolve any variable name conflicts between the calling and the called routine. I/O statements will be flagged but not translated. Most Fortran programs assume that output goes to a printer with a 132 column width; most write statements will have to be rewritten for this reason, anyway. In any case, it is not considered good programming practice to have I/O statements in mathematical subroutines, so the problem may not occur.

Some untranslatable keywords, such as implicit, equivalence, and so on, will be ignored by Translator. Since they always occur at the beginning of a routine, they are easy to find. Some—especially equivalence—may render the entire routine untranslatable, because their effects may not be duplicatable in Basic. You have to have a pretty good knowledge of Fortran to handle the more difficult jobs.

Less common contructs, such as the seldom-seen assigned GOTO, and the Fortran 77 extensions, such as the elusive block if, are not translated. Older routines generally don't use such constructs—in fact, many don't even use the logical if, sticking strictly to the arithmetic version (translated to a Basic On . . . GOTO). Variable typing also is ignored, since Basic doesn't have double-precision or integer variables, and strings are rarely used in Fortran. Be sure to read through the Fortran source for potential problems.

Even if no untranslated keyword occurs, you are likely to have problems with variable names. Any variable name that starts with a Basic reserved word is illegal. Thus, FNAME is perfectly legal in Fortran, but not in Basic, since FN is reserved. Also TIME1 and TIME2 are different variables in Fortran, but the same to your Color Computer, since Basic looks only at the first two letters. It is useful to use a variable name cross-reference generator, which will find all the variable names in a program and list them alphabetically.

You can use the built-in Basic line editor to make the necessary changes, but a word processor or programming utility makes it much easier. Instead of changing each variable name individually, you can use a global search/change function.

Also watch out for Fortran integer variables (those which start with letters I through N, unless explicitly declared otherwise) being set equal to real variables or expressions. You should use the Basic fix function on the real expression to chop off the fractional part.

The speed or size of the translated program can be improved with a little more editing. For example, in most cases, an arithmetic If can be reduced to a simple If . . . Then statement, rather than the On . . . GOTO produced by Translator.

To add Fortran keywords to Translator's vocabulary, check the data statement in line 1320. Words to be flagged but not translated should be added to the beginning of the data statement. Also, add one to both of the numbers in line 1300 (total keywords and untranslated keywords). Keywords to be translated should be added to the end of line 1320, and only the first number in line 1300 increased. In both cases, the flag or the translation should go into the corresponding position in line 1340. Translation of constructs which need more than straight one-for-one replacement will have to be handled individually.

The translator is written for 32K. If you have 16K, you should remove all remarks and unnecessary spaces, change line 90 to 90 CLEAR 6000, and reduce NMAX in line 1100 to 200. Before CLOADing the Translator, PCLEAR0 (POKE 25,6 : NEW) to open up maximum memory.

In spite of the exceptions, the great majority of Fortran IV statements will be successfully translated. *(end)*

```
C TEST FILE FOR FORTOBAS
      SUBROUTINE STATS (Y, N, AVG, STDDEV, IERR)
C DIM STATEMENT SHOULD BE REMOVED FOR BASIC:
      DIMENSION Y(1)
      IER=0
      AVG=0.0
      STDDEV=0.0
C
C MAKE SURE N IS GREATER THAN ZERO:
      IF (N) 999, 999, 100
  100 DO 200 I = 1, N
      AVG=AVG+Y(I)
      STDDEV=STDDEV + Y(I)**2
200   CONTINUE
C
C "FLOAT" ISN'T NEEDED IN BASIC:
      AVG=AVG/FLOAT(N)
      STDDEV = SQRT(STDDEV/FLOAT(N) - AVG ** 2)
      GO TO 1000
C
C ERROR IN INPUT DATA:
 999  IER=1
      IF (N .EQ. 0) GOTO 950
C N LESS THAN ZERO:
      WRITE (6,601) N
601   FORMAT (1X, 'ERROR -- N = ', I5, ', LESS THAN
 0')
      GO TO 1000
C
C N = ZERO:
950   WRITE (6, 600) N
600   FORMAT (1X, 'ERROR -- N WAS ZERO')
1000  RETURN
C
C ########################
C SOME DELIBERATE ERRORS:
C UNDEFINED LINE NUMBERS:
      GOTO 3000
      DO 3000 K=1, 5, 2
C MISSING THIRD NUMBER:
      IF (N) 999, 999
C ########################
```

Figure 1. Sample Fortran Source

```
10 ''''''''''''''''''''''''''''
20 'FORTRAN-TO-BASIC TRANSLATOR'
30 '
40 '    ROBERT L. HAWKINS
41 ' 1286 HUNTER AVE., APT. C
42 '        P.O. BOX 8162
43 '  COLUMBUS, OHIO  43201
44 ''''''''''''''''''''''''''''
80 CLS 4 : PRINT "  fortran to basic translator" :
PRINT
90 CLEAR 16000
994 '          =*=
995 '*initialize*'
1000 TRUE=-1 : FALSE=0 : IO=-1 'IO=1 FOR DISK
1005 'LOGICAL-VALUED FUNCTIONS:
1010 DEFFN CM(X)=(LEFT$(A$,1)="C") 'COMMENT?
1020 DEFFN CON(X)=(MID$(A$,6,1)<>" ") 'CONTINUATION
?
1030 DEFFNUM(X)=((D$>="0"AND D$<="9") OR D$=" ") 'N
UMERIC OR BLANK?
1094 '          =*=
1095 'PROGRAM IS STORED IN Q$(0..NL).
1096 '
1100 NMAX=600 : DIM Q$(NMAX)
1110 NL=0
1115 '          =*=
1120 INPUT "  STARTING LINE NUMBER >= 10000"; LN :
IF LN<10000 THEN SOUND2,2 : GOTO 1120 'STARTING BAS
IC LINE#
1195 'LOGICAL OPERATIONS:
1200 DATA 8
1210 READ NOP
1220 DIM FOP$(NOP), BOP$(NOP)
1225 'FORTRAN OPS:
1230 DATA .LT.,.GT.,.EQ.,.LE.,.GE.,.NE.,.AND.,.OR.,
.NOT.
1235 'BASIC OPS:
1240 DATA <,>,=,<=,>=,<>," AND "," OR "," NOT "
1250 FOR I=0 TO NOP : READ FOP$(I) : NEXT I
1260 FOR I=0 TO NOP : READ BOP$(I) : NEXT I
1293 '          =*=
1295 'FORTRAN  TO BASIC KEYWORDS:
1300 DATA 17, 10
1310 READ NKEY, NSKIP '0..NSKIP ABORT FURTHER TRANS
LATION
1315 'FORTRAN:
1320 DATA " READ("," READ ("," WRITE("," WRITE (","
 FORMAT("," FORMAT (","  CALL EXIT "," CALL "," SUBR
OUTINE"," DIMENSION"," CONTINUE ","SQRT(","SQRT (",
"ALOG(","ALOG (","ATAN(","ATAN (","**"
1335 'BASIC:
1340 DATA " 'read("," 'read("," 'write("," 'write("
," 'format("," 'format("," STOP ", " SUBGO  "," <",
" DIM"," : "," SQR("," SQR("," LOG("," LOG("," ATN(
"," ATN(","^"
1360 DIM FK$(NKEY), BK$(NKEY)
1370 FOR I=0 TO NKEY : READ FK$(I) : NEXT I
1380 FOR I=0 TO NKEY : READ BK$(I) : NEXT I
1993 '
1994 '  =*=    =*=    =*=
1995 '*read FORTRAN INTO Q$(1..NL)*'
2000 PRINT
2010 INPUT "FORTRAN FILENAME"; NM$
2020 Q$(0)="C PROGRAM "+NM$
2030 PRINT : PRINT " OPENING FILE"
2040 OPEN "I",#IO,NM$
2045 'INPUT LOOP:
2050 LINEINPUT#IO,A$
2055 'CONTINUATION CARD? IF SO, CONCATENATE:
2060 IF (FNCON(0) AND NOT FNCM(0)) THEN Q$(NL)=Q$(N
L)+RIGHT$(A$,LEN(A$)-6) : GOTO 2090
2065 'NO -- A NEW LINE:
2070 PRINT Q$(NL)
2075 IF NL<NMAX THEN NL=NL+1 ELSE SOUND 2,2 : PRINT
 "TOO MANY LINES --" : INPUT" <ENTER> TO CONTINUE";
 Z$ : GOTO 3000
2080 Q$(NL)=A$
2090 IF NOT EOF(IO) THEN 2050
2100 PRINT Q$(NL)
2110 CLOSE#IO : SOUND 100,2
2120 CLS 5
2125 '          =*=
2995 '*add basic line#'s*'(Q$(0..NL)
3000 FOR N=0 TO NL
3010 A$=Q$(N) : LA=LEN(A$)
3015 'COMMENT?
3020 IF FNCM(0) THEN C$="REM "+RIGHT$(A$,LA-1) : GO
TO 3060
3025 'SPLIT FORTRAN LINE# (B$) FROM STATEMENT (C$):

3030 C1=5 : C2=7 : GOSUB 9050
3040 LL=VAL(B$) 'LINE# FIELD
3045 'PUT FORTRAN LINE# AT END OF LINE AS REMARK:
3050 IF LL THEN C$=C$+" '"+STR$(LL)
3055 'THE "5" ASSUMES 5-DIGIT BASIC LINE#'S
3060 Q$(N)=RIGHT$(STR$(LN),5)+" "+C$
3070 LN=LN+10 'INCREMENT BASIC LINE NUMBER
3080 PRINT Q$(N)
3090 NEXT N
3100 CLS 3
3993 '
3994 '   =*=   =*=   =*=
3995 '*translate*'(Q$(0..NL); Q$(0..NL))
3997 '
4000 FOR N=0 TO NL
4010 A$=Q$(N) : LA=LEN(A$)
4020 IF MID$(A$,7,3)="REM" THEN  4200 'REMARKS UNCH
ANGED
4022 '          =*=
4025 'TRANSLATE KEYWORDS:
4030 FOR I=0 TO NKEY : C1=INSTR(6,A$,FK$(I)) : IF C
1=0 THEN 4050 'SEARCH FOR KEYWORD
4040 C2=C1+LEN(FK$(I)) : C1=C1-1 : Z$=BK$(I) : GOSU
B 9000 : IF I<=NSKIP THEN 4200 'REPLACE WITH BASIC

4050 NEXT I
4052 '          =*=
4053 '"DO", "IF" AND "GO TO":
4055 '"DO" STATEMENT?
4060 COL=INSTR(6,A$," DO ")
4070 IF COL THEN GOSUB 5000 : GOTO 4200 'DO-TO-FOR
XLATOR
4075 '"IF" STATEMENT?
4080 COL=INSTR(6,A$," IF ") : IF COL=0 THEN COL=INS
TR(6,A$," IF(")
4090 IF COL THEN GOTO 6000 '"IF" XLATOR
4095 '"GOTO"?
4100 COL=INSTR(6,A$," GO") : IF COL=0 THEN COL=INST
R(6,A$,")GO")
4110 IF COL=0 THEN 4200
4120 COL=COL+3
4130 IF MID$(A$,COL,3)="TO " THEN COL=COL+3 ELSE IF
 MID$(A$,COL,4)=" TO " THEN COL=COL+4 ELSE COL=0
4140 IF COL=0 THEN 4200
4150 GOSUB 7000 'GOTO XLATOR
4200 PRINT A$
4210 Q$(N)=A$
4220 NEXT N
4990 GOTO 8000 'FINAL OUTPUT
4992 '
4993 '          =*=
4995 '*do-to-for XLATOR*'(A$,LA; A$,LA)
5000 C1=COL : C2=COL+3
5010 Z$="FOR" : GOSUB 9000 'REPLACE
5015 'GET DO-LOOP TARGET LINE:
5020 FL=VAL(RIGHT$(A$,LA-COL-3))
5025 'EXCISE IT:
5030 C1=COL+4 : C2=C1+1
5040 D$=MID$(A$,C2,1) 'SEARCH FOR NON-NUMERIC,NON-B
LANK
5050 IF FNUM(0) THEN C2=C2+1 ELSE 5100
5060 IF C2<LA THEN 5040 'KEEP LOOKING
```

```
5065 'IF YOU GET HERE, ERROR:
5070 EMSG$="LOST IN DO"
5080 GOSUB 8500
5090 RETURN
5095 'FOUND -- D$ IS FIRST CHAR OF DO VARIABLE NAME
:
5100 DV$=D$
5110 C3=C2+1
5120 D$=MID$(A$,C3,1)
5130 IF D$="=" THEN 5170 'END OF VARNAME
5140 DV$=DV$+D$ 'BUILD VARNAME
5150 C3=C3+1 : IF C3>LA THEN 5070 'LOST
5160 GOTO 5120
5165 'EXCISE DO-NUMBER:
5170 Z$="" : GOSUB 9000
5175 'CHANGE "," TO "TO":
5180 C1=INSTR(C1+1,A$,",")
5190 IF C1=0 THEN 5070 'NO COMMA -- ERROR
5200 C1=C1-1 : C2=C1+2
5210 Z$=" TO " : GOSUB 9000 'REPLACE
5215 'IF ANOTHER ",", => "STEP":
5220 C1=INSTR(C1+3,A$,",")
5230 IF C1=0 THEN 5300
5240 C1=C1-1 : C2=C1+2
5250 Z$=" STEP " : GOSUB 9000
5295 'NOW FIX TARGET LINE:
5300 SV$=A$
5310 GOSUB 7500 'FIND BL FROM FL
5320 IF EFLAG THEN EMSG$="DO-LOOP TARGET "+STR$(FL)
+" NOT FOUND" : GOSUB 8500 : RETURN
5330 A$=Q$(I) : LA=LEN(A$) 'TARGET LINE
5335 'CHECK IF PREVIOUS "NEXT" HAS BEEN ADDED:
5340 C1=INSTR(6,A$," : NEXT ") : IF (C1<COL AND C1)
 THEN COL=C1
5350 C1=COL : C2=COL+1
5360 Z$=" : NEXT "+DV$+" " : GOSUB 9000
5370 Q$(I)=A$
5380 A$=SV$ : LA=LEN(A$)
5390 RETURN
5995 '*if xlator*'(A$,LA,COL; A$,LA) "IF (EXPR) STA
TEMENT"
6000 C9=COL : C1=INSTR(COL,A$,"(")
6010 IF C1=0 THEN 6900 'ERROR
6015 'FIND MATCHING PARENS:
6020 C3=1
6030 FOR C4=C1+1 TO LA
6040 D$=MID$(A$,C4,1)
6050 IF D$=")" THEN C3=C3-1 ELSE IF D$="(" THEN C3=
C3+1
6060 IF C3=0 THEN 6100
6070 NEXT C4
6075 'IF YOU GET HERE,ERROR:
6080 EMSG$="MISMATCHED PARENS IN IF" : GOSUB 8500
6090 RETURN
6095 'C3,C4 BRACKET (EXPR):
6100 C3=C1
6115 'IF LOGICAL OPS USED, ASSUME LOGICAL IF:
6120 LFLAG=FALSE
6130 FOR I=0 TO NOP
6140 COL=INSTR(C3,A$,FOP$(I))
6150 IF COL=0 OR COL>C4 THEN 6200
6160 C1=COL-1 : C2=COL+LEN(FOP$(I))
6170 Z$=BOP$(I) : GOSUB 9000 'REPLACE
6180 C4=C4+LEN(Z$)-LEN(FOP$(I)) : LFLAG=TRUE
6190 GOTO 6140
6200 NEXT I
6205 'LOGICAL OR ARITHMETIC?
6210 IF LFLAG THEN C1=C4 : C2=C1+1 : Z$="THEN " : G
OSUB 9000 : GOTO 4100 'LOGICAL
6215 'ARITHMETIC: "IF (NUMBER) 100,200,300"
6220 C1=C9 : C2=C1+3
6230 Z$="ON 2+SGN" : GOSUB 9000 'REPLACE "IF"
6240 C4=C4+LEN(Z$)-2 'CLOSING PARENS
6250 C1=C4 : C2=C1+1
6260 Z$=" GOTO " : GOSUB 9000 'INSERT "ON ... GOTO"

6265 'CHANGE TO BASIC LINE#'S:
6270 C5=C4+LEN(Z$)
6280 C2=INSTR(C4,A$,",") : IF C2=0 THEN 6900
6290 GOSUB 7010 'REPLACE LINE#
6295 'GET NEXT #:
6300 C4=INSTR(C5,A$,",") : IF C4=0 THEN 6900
6310 C5=C4 : C4=C4+1
6320 C2=INSTR(C4,A$,",") : IF C2=0 THEN 6900
6330 GOSUB 7010
6335 'LAST ONE:
6340 C4=INSTR(C4,A$,",") : IF C4=0 THEN 6900
6350 C5=C4
6360 FOR C2=C4+1 TO LA : D$=MID$(A$,C2,1) : IF FNUM
(0) THEN NEXT C2 ELSE C2=C2-1 : GOTO 6380
6370 C2=LA+1
6380 GOSUB 7010
6390 GOTO 4200
6895 '*if error*'
6900 EMSG$="LOST IN IF" : GOSUB 8500
6910 GOTO 4200
6995 '*goto xlator*'(N,COL,A$; A$,LA)
7000 C5=COL-1 : C2=LA+1 'SKIP OVER " GO TO"
7010 FL=VAL(RIGHT$(A$,LA-C5)) 'TARGET FORTRAN LINE#

7020 GOSUB 7500 'FIND BL CORRESPONDING TO FL
7025 'IF LINE NOT FOUND, ERROR:
7030 IF NOT EFLAG THEN 7100
7040 EMSG$="LINE# "+STR$(FL)+" NOT FOUND"
7050 GOSUB 8500 'ERROR ROUTINE
7060 RETURN
7095 'TARGET LINE FOUND:
7100 C1=C5 'C2 SET EARLIER OR BY CALLING ROUTINE
7110 Z$=STR$(BL) : GOSUB 9000 'REPLACE LINE#
7120 RETURN
7495 '*find target line*'(FL; BL, EFLAG)
7500 EFLAG=TRUE : Z$=" '"+STR$(FL) : LZ=LEN(Z$)'TAR
GET STRING
7510 FOR I=1 TO NL
7520 COL=INSTR(6,Q$(I),Z$) : IF COL<>LEN(Q$(I))-LZ+
1 THEN COL=0 'DON'T CONFUSE "1000" FOR "100"!
7530 IF COL THEN EFLAG=FALSE : BL=VAL(Q$(I)) : RETU
RN 'FOUND
7540 NEXT I
7550 RETURN
7995 '*output*'(Q$(0..NL))
8000 CLS : PRINT "        TRANSLATED OUTPUT" : PRINT
 : IF IO>0 THEN IO$="dISK" : C$="SDPQ" ELSE IO$="cA
SSETTE" : C$="SCPQ"
8010 PRINT"OUTPUT TO:" : PRINT
8020 PRINT"       sCREEN ONLY" : PRINT
8030 PRINT"       ";IO$ : PRINT
8040 PRINT"       pRINTER" : PRINT
8045 PRINT"       qUIT" : PRINT
8050 PRINT : PRINT "YOUR CHOICE?";
8060 Z$=INKEY$
8070 Z$=INKEY$ : IF Z$="" THEN 8070
8080 I=INSTR(1,C$,Z$) : IF I=0 THEN SOUND 2,2 : GOT
O 8000
8090 ON I GOTO 8100, 8200, 8300 : CLS : PRINTTAB(8)
"== FINISHED ==" : END
8095 'SCREEN:
8100 PRINT
8110 FOR I=0 TO NL
8120 PRINT Q$(I)
8130 NEXT I
8140 PRINT : INPUT "<enter> FOR MENU"; Z$
8150 GOTO 8000
8195 'DISK/CASSETTE:
8200 CLS : PRINT IO$; : INPUT " FILENAME"; NM$
8210 PRINT
8220 OPEN"O", #IO, NM$
8230 FOR I=0 TO NL
8240 PRINT#IO, Q$(I) : PRINT Q$(I)
8250 NEXT I
8260 CLOSE#IO
8270 GOTO 8140
8295 'PRINTER:
8300 CLS
```

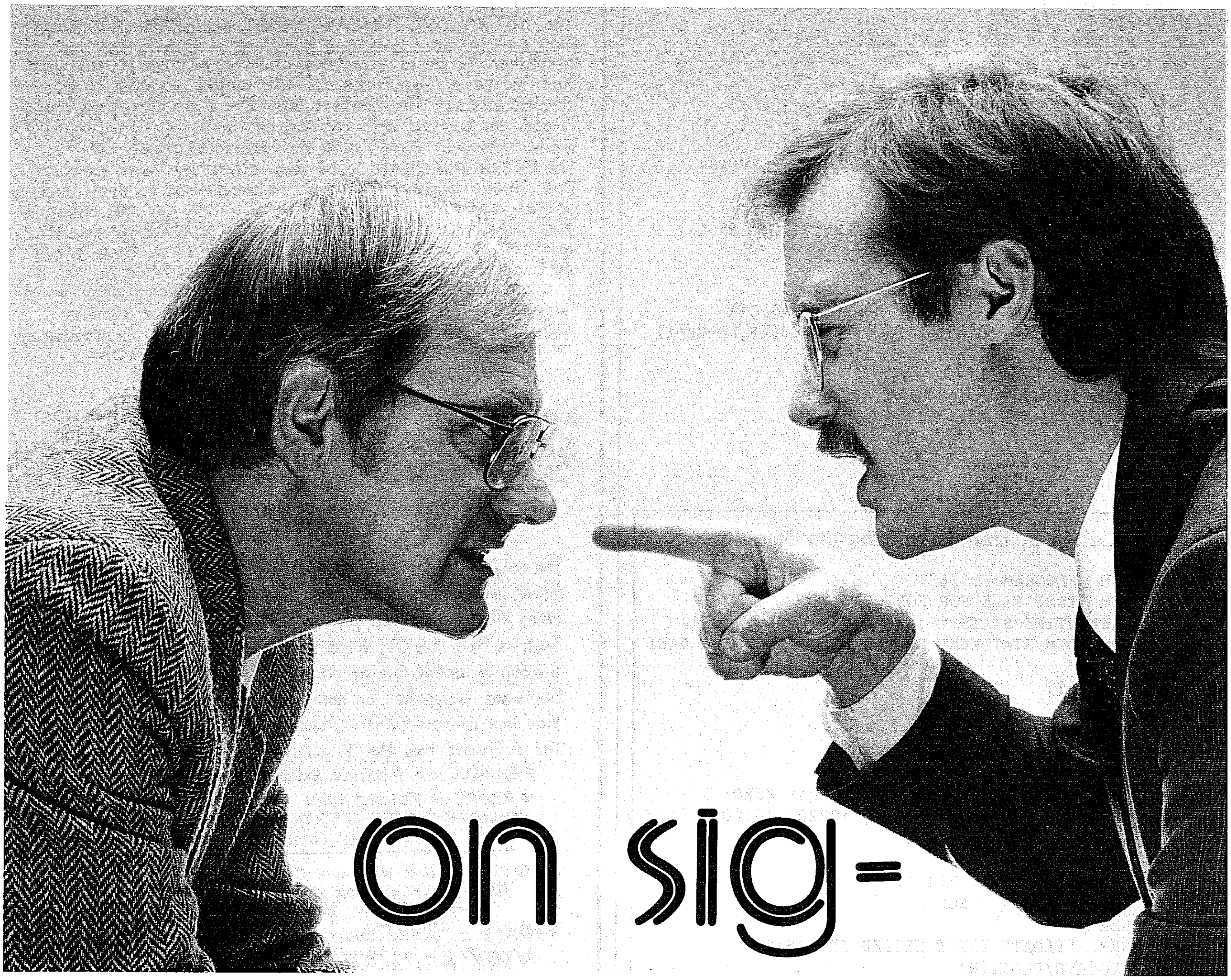
```
8310 FOR I=0 TO NL
8320 PRINT#-2, Q$(I) : PRINTQ$(I)
8330 NEXT I
8340 GOTO 8140
8350 '************************'
8495 '*error MESSAGE*'
8500 SOUND2,2
8510 A$=A$+"'****error**** "+EMSG$ : LA=LEN(A$)
8520 PRINT A$
8530 RETURN
8995 '*insert Z$ INTO A$*'(A$,LA,Z$,C1,C2; A$,LA)
9000 GOSUB 9050 : GOSUB 9100
9010 RETURN
9045 '*split A$*'(A$,LA,C1,C2; B$,C$)
9050 IF C1<1 THEN B$="" ELSE B$=LEFT$(A$,C1)
9060 IF C2>LA THEN C$="" ELSE C$=RIGHT$(A$,LA-C2+1)

9070 RETURN
9095 '*re-form A$*'(B$,Z$,C$; A$,LA)
9100 A$=B$+Z$+C$ : LA=LEN(A$)
9110 RETURN
```

Listing 2. Translated Program Sample

```
10000 REM  PROGRAM FORTEST
10010 REM  TEST FILE FOR FORTOBAS
10020 SUBROUTINE STATS (Y, N, AVG, STDDEV, IERR)
10030 REM  DIM STATEMENT SHOULD BE REMOVED FOR BASI
C:
10040 DIM Y(1)
10050 IER=0
10060 AVG=0.0
10070 STDDEV=0.0
10080 REM
10090 REM  MAKE SURE N IS GREATER THAN ZERO:
10100 ON 2+SGN (N) GOTO  10220, 10220, 10110
10110 FOR I = 1 TO  N ' 100
10120 AVG=AVG+Y(I)
10130 STDDEV=STDDEV + Y(I)^2
10140 :  : NEXT I  ' 200
10150 REM
10160 REM  "FLOAT" ISN'T NEEDED IN BASIC:
10170 AVG=AVG/FLOAT(N)
10180 STDDEV =  SQR(STDDEV/FLOAT(N) - AVG ^ 2)
10190 GO TO  10320
10200 REM
10210 REM  ERROR IN INPUT DATA:
10220 IER=1 ∸ 999
10230 IF (N = 0)THEN  GOTO  10300
10240 REM  N LESS THAN ZERO:
10250 'write(6,601) N
10260 'format(1X, 'ERROR -- N = ', I5, ', LESS THAN
 0') ' 601
10270 GO TO  10320
10280 REM
10290 REM  N = ZERO:
10300 'write(6, 600) N ' 950
10310 'format(1X, 'ERROR -- N WAS ZERO') ' 600
10320 RETURN ' 1000
10330 REM
10340 REM  #######################
10350 REM  SOME DELIBERATE ERRORS:
10360 REM  UNDEFINED LINE NUMBERS:
10370 GOTO 3000'****error**** LINE#  3000 NOT FOUND

10380 FOR K=1 TO  5 STEP  2'****error**** DO-LOOP T
ARGET  3000 NOT FOUND
10390 REM  MISSING THIRD NUMBER:
10400 ON 2+SGN (N) GOTO  10220, 999 '****error****
LOST IN IF
10410 REM  #######################
```



☆ **Fm: Roger Bouchard** (to A. Flexser)

Once again, what in heaven is wrong with the RAM hook? If need be you can use a disk routine by calling it by its command if you use the disk command table of pointers!

(A reply follows:)

Fm: A. Flexser (to R. Bouchard)

Most disk commands do not have RAM hooks attached that are useful entry points from ML programs.

To elaborate . . . I do agree that the use of the command dispatch table (via the RAM entry point at $137 is a technique that should be used when possible in preference to the more usual technique of checking for version number and using one of two possible jump addresses depending on version. But, the problem is that, more often than not, one does not want to jump into the ROM to a point that corresponds to the first byte of a Basic command routine. Usually, one wants to skip over a scan for arguments, or use a subroutine that is also used by one or more Basic commands. The point is that more of these general-purpose entry points should have been alloted their own (ROM) vectors.

★★**Fm: Marty** (to J. Ross)

True, John, but the point of having an operating system in the first place is to *not* force the programmer to write his or her own disk I/O, file handling, keyboard, screen, etc., routines. The solution you mentioned will work, but the fact that folks have to resort to it speaks of the abysmal lack of planning by Microsoft and Tandy in their miserable R/S DOS.

(The one answer follows:)

Fm: John Ross (to Marty)

I agree totally! But, since RS and Microsoft did not see fit to supply constant vector tables, and insist on moving things around, that approach is the *only* one that makes any sense! Otherwise you spend all your time changing versions for new ROMs, and answering hate mail!

☆☆**Fm: A. Flexser** (to Marty)

Yes, introducing those vectors in a "better late than never" fashion would be advantageous in limiting the number of versions future software might have to check for. Plus, if the same vectors were in the 09, it might lead to transportability between new CoCo software and that machine.

(One reply follows:)

Fm: Marty (to A. Flexser)

Very good point. I didn't mention that.

But that point is probably the single most cogent argument for their bothering at all to introduce disk file I/O vectors at this late date. Yes, very good point indeed. *Now* let's see what ol' Wayne can answer to that one! (Boy, do I ever get annoyed by folks who apologize for the blunders of others!)

☆☆**Fm: Wayne Day** (to Marty)

Simply because I'm tired of hearing people complain about the way it is . . .

Tandy, again and for the last time, told folks from the very beginning to not use the un-documented calls . . . and those programmers who have made life very miserable for a lot of otherwise innocent folks.

I'm interested mainly in the average non-techie user's viewpoint here . . . not someone who *knows* how it's "supposed" to be done, but rather the person that plunks down $25 to $100 for a program and then finds it doesn't work on their machine.

Again, I stand by my original statement, and it is one you can never change my mind about (at least until the hardware addresses change): The majority of compatibility problems existing in the CoCo today are because of programmers who were lazy, or inefficient, or simply unable to write their own code so that their product would be ROM-version independent.

You, I, nor any of the people on this Sig will *never* change Tandy's mind about declaring all ROM calls, or at least the vectors, to be sacred.

(One reply follows:)

Fm: Marty (to W. Day)

Guess you were serious in what you were writing, not just playing devil's advocate. Sorry to hear that. You are simply, totally, plumb dead wrong in the position you have taken. That's all there is to it, and it's really not much of a matter of opinion. I'd venture to guess that 95 percent or more of assembly language programmers for the CoCo would agree with me. If you've ever written or began to write a machine language program that did disk I/O of whole files on the CoCo and under other operating systems, you'd understand. Well, you're generally so right on most CoCo and computer-related matters that I guess you can afford the luxury of being totally, laughably wrong in this one area.

Be well.

(One reply follows:)

Fm: John Ross (to Marty)

I don't think Wayne is saying he agrees with their method, he is just saying that Tandy *warned* them, and they should have listened! Those that did not, have suffered for it, as have the people that buy their programs.

(OK folks. That's the end of that conversation thread. Now, think back a few sections:)

Fm: Marty (to S. Trevor)

What you said has merit, but if you look at the variety of customized disk ROMs on the market, you'll find that virtually all of them leave DSKCON, its vectors, and the disk file handling code intact in the same places that it was in the R/S DOS parent ROM. That's why I suggested checking for version type by looking at the DSKCON vector. I grant you that if programmers were not so vain and did not mess with the version number byte, things would be better. But most programmers *are* vain! Also, vanity aside, what about this possibility? Tandy comes out with a new disk ROM version number, but the file-handling routines are in the same place. In such an event, your program would give up and die with a gasp of "alien ROM!," whereas one checked for vectors would go merrily on its way. Perhaps a more rigorous version type check would consist of checking both the DSKCON vector and the first two bytes of the entry points to those subroutines that the program was to use. If those checked out, regardless of other mods, the odds would be good that it would have a friendly operating environment. What do you think of that idea? It's cumbersome, of course, but it might not be a bad idea.

(A reply follows:)

Fm: Sandy Trevor (to Marty)

OK, we'll consider modifying the version check.

Fm: A. Flexser (to S. Trevor)

Sandy, I've got to agree with Marty on this. So what if a version check of DSKCON results in Vidtex mistakenly thinking that it is dealing with (say) RS-DOS 1.0 when it is in fact dealing with a modified third-party ROM. If that modified ROM kept DSKCON in the same place as the RS-DOS ROM, chances are Vidtex will work OK. And if not, it'll just crash, which is exactly what it does now if that logo byte isn't in the right place.

(A reply follows:)

Fm: Sandy Trevor (to A. Flexser)

It shouldn't crash! It should work fine but without the disk commands. What Disk ROM causes Vidtex to crash?

(A reply follows:)

Fm: A. Flexser (to S. Trevor)

Anything (like JDOS, or ADOS in its initial versions) that doesn't have that logo byte where Vidtex expects it to be. It just locks up, as I recall. (Wayne, is that your recollection from the early ADOS?)

(A reply follows:)

Fm: Mike Ward (to A. Flexser)

I wonder why nobody seems to have any trouble with the Mouse? (Clara runs it on a CoCo II with 1.1 Disk ROMs.) You use ADOS and I clunk along with my old 1.0 ROMs . . .

(Two replies follow:)

Fm: Wayne Day (to M. Ward)

It *obviously* couldn't be superior programming! hehehe

(The reply is:)

Fm: Mike Ward (to W.Day)

Undoubtedly true! They're merely flabbergasted!

Fm: A. Flexser (to M. Ward)

Because you took the "lazy" way and used undocumented ROM calls, you layabout, you!

(The reply:)

Fm: Mike Ward (to A. Flexser)

That's true, and I also was careful to see which version of the ROMs are installed. If I can't figure it out, I assume 1.0.

(The reply:)

Fm: A. Flexser (to M. Ward)

Sounds good to me! *(end)*

Disk Drive Controllers: A Comparison

By Martin Goodman, M.D.

PRODUCT

CoCo II Disk Controller
Radio Shack
$90–$130

J&M Controller
J&M Systems
Albuquerque, NM
$120–$140

HDS Disk Controller
HDS

Since the introduction of the Color Computer there have been at least seven different Color Computer Floppy Disk Drive Controller boards available to the public. Of these, three have enjoyed extremely wide-scale sales, and two of those three are still in production. I'm going to concentrate on the two popular ones currently being sold, but I'll mention others.

What is a Floppy Disk Controller? It's that black box that plugs into the side of the CoCo (or into your system bus) at one end, and through a cable to your disk drive unit(s) at the other. In the case of the CoCo disk controller, this item is a circuit board that contains both the hardware needed for the computer to talk to the disk drive (disk controller chip and numerous small scale logic support chips) and the added software (usually Disk Extended Color Basic) needed to drive the hardware. That software is actually "firmware"—a ROM or EPROM chip. It's important to remember that while you often buy the two (disk operating system software and hardware) together, in a CoCo disk controller they are two separate items. That is, one can plug a different ROM or EPROM into the disk controller and have either Disk Extended Color Basic, JDOS, Spectrum Dos, ADOS or any of a number of different variant Disk Basic operating systems.

Warning: While a disk controller is meant to connect to its disk drives through a ribbon cable, it is *not* designed to connect to the computer through a ribbon cable. While a number of third party suppliers sell "Disk Extender Cables" and "Y cables," I urge you *not* to use such products! Both the lack of proper grounding and the added capacitance of the ribbon cable on the unbuffered CoCo's system bus will, on many systems, cause intermittent system crashes (typically during disk drive use). This is a lesson I've learned through bitter experience and extensive testing and polling of my fellow CoCo users. Heed this warning!

With one trivial exception, to be mentioned at the end of this article, all CoCo disk controllers are designed to hook up to any 35- or 40-track single-sided double-density disk drive. With appropriate exotic operating systems they can be all made to operate, to some degree or other, with double sided and/or 80-track drives, but such operation is for experienced system hackers only. Most CoCo users should stick with the Tandy standard, of 40-track single-sided double-density drives. I myself take this advice. Experienced OS-9 users and BBS system operators may be exceptions.

• Tandy was the first to introduce a double density disk controller for the Color Computer. This, the CoCo I disk controller, was the only game in town for quite a while. It worked reasonably well, even though in some respects its design was a bit clumsy. I've been using a CoCo I disk controller on my main CoCo system for the last two years, and it has been trouble free. My criticisms of the unit relate to the fact that some of them tended to pop their 74LS02 chips now and then, and that they sometimes required adjustments of the potentiometers on the oscillator for the data separator.

There is another problem with the old CoCo I disk controllers: when Tandy stopped making the old D, E, and NC (F) boards, and switched over to making the Color Computer 2, they altered the computer so it no longer provides +12 volts on its system bus. The CoCo I disk controller required +12 volts to operate its archaic Western Digital 1793 chip. So unless you modify your CoCo 2 (articles have been written about this) you can't use the old CoCo I controller directly with the new CoCo 2. If you are using a Tandy or PBJ multiple port bus, you will be able to use a CoCo I controller with a CoCo 2.

The CoCo I disk controller is no longer sold. Tandy has discontinued it in favor of its CoCo 2 disk controller. The CoCo 2 disk controller represents a distinct improvement over its predecessor. Its circuitry is cleaner and requires less frequent adjustments; because Tandy switched over to the Fujitsu MB8877A disk control chip, it does not require +12 volts. This means it works *equally* well on both the older and the newer model Color Computers. It is available from Tandy National parts, if you order the circuit board and the two halves of the plastic shell. Cost will be between $90 and $130. Overall, the CoCo 2 disk controller is a very reliable, well-designed and -built unit. I have one *very* serious criticism of it: despite insistent please from knowledgeable users and hackers over the last five years (this stems from the days of the Model I) Tandy still has *not* plated the connectors of its disk controller with gold. There is abundant evidence that the failure of Tandy to do so has resulted in system crash after awhile, due to oxidation of the contacts on the disk controller. A partial remedy consists of periodically removing the disk controller card from its shell and gently cleaning its contacts with a soft pencil eraser. I've "repaired" at least three Radio Shack store computers in just such a fashion.

The permanent fix consists of either sending the

Reviews

unit out to a qualified electrical gold-plating establishment (hard to find) or soldering on a product called the Gold Plug. This item is sold by EAP Corporation, POB 14, Keller, TX 76248, (817)498-4242. It requires a little hardware ability to install, but once installed it will fully cure problems relating to oxidation of the contacts. Cost of the package is around $18.

I should note in passing that Tandy, in a display of sublime technical stupidity, has recently started releasing ROMpaks with gold-plated connectors (ROMpaks don't really need gold-plated connectors) but still does not gold plate the connectors of the disk controller or the Multipak, both of which do need gold plating.

• By far the best known and most widely sold non-Tandy CoCo disk controller is made by J&M Systems of Albuquerque, NM. This unit came on the market just before the arrival of the Tandy CoCo 2 disk controller. It features competent design, a sturdy metal case, and gold-plated connectors. This last makes it a very desirable unit. I and many friends have used the J&M Systems controller extensively and have generally been very pleased with its reliability and ruggedness. The circuit board is well laid-out and manufactured in a high quality fashion. Like the CoCo 2 disk controller, it features a quartz crystal-controlled phase-locked loop data separator that requires no adjustment.

When friends of mine managed to blow out their unit (by unplugging it from the computer while the system was on) J&M provided extremely prompt repair services. Overall, it's a very nice unit, superior in some ways to the Tandy product, and I recommend it. It retails for around $120 to $140. I have a few complaints and cautions about the J&M product: J&M gave me a run-around when I tried to order a schematic from them. For months they stalled me . . . first refusing to send me one, then claiming they were in the process of printing a service manual. After about a half-dozen calls over a four month period (and a letter), I finally received a badly Xeroxed copy of the schematic and some other technical info, including spec sheets on the Synertec chip they use in their controller. They charged me $25 for those 10 pages.

The J&M controller uses a Synertec disk controller chip which is very similar to, but slightly different from, the Fujitsu chip used in the Tandy CoCo 2 disk controller. While this will make no difference in running normal software, some weird copy protection schemes may result in disks that won't load on systems with the J&M system controller, and some disk clone programs may not work properly with that controller. Also, the Synertec chip is much harder for the hacker to find.

The J&M disk controller is sold with a choice of disk operating systems on ROM or EPROM. You can order it with Tandy's Disk Extended Basic or with JDOS, an "enhanced" Disk Basic. *Don't* buy it with JDOS: order it with Tandy's standard Disk Basic. If you do not heed this warning you'll discover a number of valuable pieces of software that simply won't run right under JDOS. While JDOS is allegedly "compatible" with Tandy's Disk Extended Basic, don't believe it!*

• HDS Disk Controller: HDS (formerly Compukit) is advertising a disk controller of its own design. This unit will be available as an assembled and tested unit, as a kit, and as a bare-board with instructions. The company apparently has a sensible policy of making documentation on their product freely available. I had a chance to play with one of these units; it featured not one, but *two* 24-pin ROM sockets, and a jumper for selecting one ROM or the other. This makes it possible to quickly switch between two different operating systems (say JDOS and Tandy's Disk Extended Basic). This feature is of moderate interest to hardware hackers, but in my opinion virtually worthless to the average user. Unfortunately, both sockets are 24-pin. It would have been much better had HDS offered one of the two sockets as a 28-pin socket to fit 2764 EPROMs, which are much cheaper and more available than the 24-pin 68764's that the unit current requires.

I briefly tested the unit and it appeared to function well. While I did not have a schematic, the circuit seemed very similar to that of the Tandy CoCo 2 disk controller. I then proceeded to modify the board to accommodate a 28-pin 2764 in one of the slots so its owner could use both 2764 and 27128 EPROMs in it. In the course of doing this, I discovered the printed circuit board to be made in a significantly inferior fashion to that of the J&M and Tandy boards. It lacks lacquer and is much more easily damaged. The traces on the board seemed thinner and more fragile than those on the Tandy and J&M boards.

I'd previously modified CoCo I, CoCo 2 and J&M disk controller cards to take 28-pin sockets. Modifying this one proved five times harder, due to the flimsy construction of the board. Therefore, my first impression of this unit is that one should *not* buy it unless one is an inveterate kit builder (it *is* the only unit offered as a kit). The assembled unit may also be of interest to hackers, because unlike both the Tandy and J&M units, HDS sockets all chips on the board.*

• Modifying a controller for 28-pin sockets: CoCo 2 disk controllers are about the easiest to modify for 28-pin sockets, with J&M and CoCo I controllers just a little more difficult. The technique I use requires a 28-pin AUGAT-style socket. I clip off the thin part of pins 1, 2, 28, 27 and 26 socket, leaving the fat part of the pin under the socket as something to solder to and as a physical spacer under the socket. I then prepare each socket by soldering on 30 gauge wires as needed to these "stumps" of pins. I then desolder the 24-pin socket, place tape on top of the area where the socket was, make needed trace cuts on the board, solder in the modified 28-pin socket, then hook up loose ends, including running a wire from pin 26 of the socket to land 37 (A13 line) on the CoCo port edge-connector.

When modifying J&M controllers you'll have to fuss with the +5 volt supply, as there's a trace you must cut but also jumper to a further point. You'll see what I mean if you look carefully at it and if you know what you're doing.

The whole process is a bit delicate but not all that difficult. It takes me about an hour. I've done it about half a dozen times now. Pin-outs for the 68764 and

27128 are, of course, essential. There are about three pins that need to be reshuffled. As an added hint, I'll tell you that pins 1, 27, and 28 of the 28-pin socket are wired to each other and to the +5 line. Pin 2 of the 28-pin socket is connected to the trace that formerly went to pin 18 of the old 24-pin socket. Pin 23 of the 28-pin socket is connected to the trace that formerly went to pin 18 of the 24-pin socket. Pin 20 of the 28-pin socket goes to ground. Pin 26 of the 28-pin socket is jumpered to land 37 (A13) on the CoCo bus edge-connector. The 28-pin socket is inserted into the holes for the 24-pin socket so that pin 3 and 26 of the 28-pin socket line up with pins 1 and 24 of the holes for the 24-pin socket.

All unmentioned pins connect straight through . . . no further jumpers needed.

• Other units: I've seen or heard of three other disk controller units for the CoCo. One features a 28-pin socket so you can jumper to accept either 24- or 28-pin EPROMs. Another is from DSS (formerly Saturn). And if I'm not mistaken, Star Kits was once marketing a disk controller, though it may have been one of the above two. Both units seemed to work acceptably, but I really can't recommend either, if only because they're odd-ball items that offer little if anything in either price or performance over the more widely distributed controllers. Please note I'm not saying they're any worse than the more commonly sold units—merely that they did not appear to be to be significantly enough better to warrant buying an obscure item.

Hard-core CoCo trivia buffs will be amused to learn that the late Exatron Corp. briefly marketed a *single-density* disk controller and operating system for the CoCo before Tandy came out with its system. The only example of such a unit in captivity that I know of is at the home of Bob Rosen. Indeed, I viewed the other two less well-known CoCo disk controllers at Bob Rosen's "Museum of Weird and Obscure CoCo Hardware."

• Conclusions: there you have it. At least seven different disk controllers were designed and produced for the Color Computer. But only two are in current production *and* in wide circulation. Deciding between these two will be difficult. The J&M unit is a bit superior to the Tandy unit, and comparably priced. I would prefer that unit because of its gold-plated connectors. But the Tandy unit is not bad, especially if you solder on EAP Corp. Gold Plugs to it. Hackers will be disappointed to hear that all small-scale logic chips on both the CoCo 2 (Tandy) and the J&M disk controller are soldered, not socketed. Tandy repair is accessible around the country via your local R/S store. This may influence some to stick with the Tandy product; but as I write, the somewhat superior J&M product is serviced promptly and conscientiously by mail by J&M. I've several positive reports from satisfied J&M customers who needed repairs and no negative ones. *(end)*

**Since this review was written, both HDS and J&M have issued new controllers. Both use the new Western Digital Controller, a 28-pin second generation disk controller. All contacts on both of these controllers are now gold-plated. The J&M controller now supports 2764 and 27128 EPROMs; the new HDS controller supports 2764, 27128 and 27256 EPROMs.*

Space Shuttle

By Jeffrey S. Parker

PRODUCT
Space Shuttle
By John Frayesse
Tom Mix Software
4285 Bradford N.E.
Grand Rapids, MI 49506
(616) 957-0444
Requires 32K Extended Color Basic
***$28.95* Tape**
***$31.95* Disk**

"Houston, this is Discovery, we have achieved altitude and are on course. Velocity 12000."

"Roger, Discovery, we copy. Telemetered data is 5-10-10 on the plot board. Standby for main booster shutdown at 53000."

"10-4, Houston, that's a go. We have orbit. Satellite on tracking, Houston. My God, Houston, the stars sure are bright out here!"

If you have ever seen the pictures of the space shuttle in the magazines, or ever watched a launch on tv, then you have probably wondered what it would really be like to fly the Space Shuttle. Now, John Frayesse, a software author for Tom Mix Software, has given *you* that opportunity, right in your own living room, on your Color Computer.

The Space Shuttle simulator is just that, a simulation, not a toy or a game. While it doesn't cost the millions and millions of dollars that NASA's cost, and it will not enable you to walk into NASA and present your credentials as an astronaut, it will give you a good sense of what it is like to fly the space shuttle.

This program probably pushes the Color Computer to the very limits of its 32K capacity. The graphics display incorporates high resolution moving graphics and text. The display screen is usually running four separate data displays at the same time. These graphic displays are some of the most sophisticated and detailed displays created for the Color Computer.

The documentation to fly your simulator provides you with a mission plan in five parts, and guides you in ten carefully detailed pages through each of the phases of the flight plan necessary to successfully complete your mission: Phase 1 is Launch, in which you must achieve a specific orbit and course direction; Phase 2 is Park, in which you must successfully locate and place the shuttle in a "parking orbit" near a damaged communications satellite; Phase 3 is Fetch, in which you must open the shuttle bay doors, manipulate the robot arm to capture the satellite, bring it safely back into the shuttle, and close the bay doors; Phase 4 is Entry, the most difficult part of the simulation, in which you must guide the shuttle to a course, altitude, speed, and target destination; Phase 5 is Final Approach, in which the shuttle is now a supersonic glider, that you must glide safely to the runway, fighting crosswinds to make a precise touchdown.

If all these tasks are not enough of a challenge, the whole mission is flown against a time clock and a fuel gauge. You are given the weather conditions and the runway heading at the beginning of every flight, and these coordinates are different each time you fly. Are you still sure you want to be an astronaut?

The flight controls, like the screen displays, are intricate. The keyboard and one joystick are used, but the joystick has a different orientation for each phase of the mission. For example, the joystick controls the main engines during launch (phase 1), but when the satellite is being retrieved during phase 3, the joystick controls the robot arm.

The instrument panel is also complex, constantly changing during the flight to bring you the essential data you need regarding course, velocity, speed, and altitude. In addition, you are given a plotting scope, similar to radar, and event labels, which change as you proceed from one phase or subphase of the mission to the next. You are provided with fuel consumption data, a mission clock, a shuttle mode indicator, (to indicate which engines are on), and a reaction jet console to tell you which of the steering motors is being fired to move the shuttle in a given direction. Extra data as to runway heading and the range to the runway is provided during the landing phase. The view out the window of the shuttle also changes from phase to phase, showing you the launch, as you move through the clouds into space, the parking "windows" to approach the satellite, the view of the shuttle aft, as you watch the doors open and the arm operating, and finally, after blackout on reentry, the runway landing scene.

Space Shuttle is a very sophisticated demonstration of what the Color Computer can do. The program provides hours of challenge and fun, and some learning as well, as to what it takes to fly a real space rocket. Some of the best features of the program are its high resolution graphics and sound effects. The moving graphics display, and the multifunction instrument display, are some of the finest uses of animated graphics I have seen for this machine; they are a real pleasure to experience. The boredom or "shelf" factor of Space Shuttle is extremely low, because no two flights are alike, so the simulator does not lose its challenge even after repeated use. The debriefing, in which you receive your flight time, score, and other landing or ("sigh") crash data after every mission is excellent in helping you track your flight failures and successes.

There are only three weaknesses that I found in flying the Space Shuttle Simulator. The first is the use of the reset button to control the color calibration. The graphics are very nice, but having to reset and then run the program as much as twenty or more times before beginning can be frustrating indeed. A good deal of calibration is necessary because blue and red can become interchanged due to the use of multicolored high resolution graphics.

The second is the Fetch phase of the flight. This segment is used to give the pilot a guaranteed score of at least 100 points. If you fail before and after the Fetch, the shuttle computer puts the ship on automatic, and you are penalized points and time and aborted to the next phase of the flight. The Fetch phase does not have that option; the satellite must be retrieved to continue, and there is no abort. Retrieving the satellite is a relatively simple feat, and can become boring and repetitious. This step could be made more challenging by adding a timer, or by repositioning the satellite slightly differently each time, rather than in one given spot to the left or right of the shuttle.

Last but not least, in the actual Park phase, where you must put the shuttle in an orbit window near the satellite, you are required to use the up or down arrows on the keyboard to accelerate or deccelerate to match orbits with the satellite. The keyboard read routine in this part of the simulation is very slow, and sometimes you must wait several seconds to push the key again before the computer will scan and accept the command. This can cause you to fly right by the satellite, and abort this phase of the mission. A more rapid keyboard scanning here could help this problem, especially since the machine is busy with so many other simultaneous calculations.

All in all, Space Shuttle is an entertaining and rewarding program, with some very accurate and realistic details. While this is not a game, and I do not recommend it for young children or those unacquainted with other flight simulations, Space Shuttle provides hours of satisfaction to anyone who enjoys meeting difficult challenges. The displays and moving graphics are finely detailed, realistic and accurate, and are a pleasure to experience.

One final note is that the product is guaranteed to load for one year from purchase. As it happened, the copy I received on disk did not load. Tom Mix Software was both courteous and extremely quick to replace my copy. It is not only a pleasure to use the Space Shuttle Simulator, it is a pleasure to do business with such a responsive company. *(end)*

Trivia

By Mark Haverstock

PRODUCT
Trivia Fever
Professional Software, Inc.
P.O. Box 533
Needham, MA 02194
64K Extended Basic
***$29.95* Disk**

Q: What is gray or white, has 53 keys and plays trivia games?

A: Your Color Computer.

Trivia Fever is a computerized trivia game that provides a more random selection of questions than its manual counterparts, as well as some desirable automatic features. Up to eight players or teams may participate. A question book and tally sheets are also provided for non-computerized play.

Under its blue and gray cover lies a slickly executed OS-9 program. However, there's no need to buy an OS-9 operating system; side one of the disk includes an OS-9 boot and system disk with game instructions. On the reverse side reside the question files.

Complete instructions are given for running Trivia Fever on Disk Basic 1.0 and 1.1. On the newer 1.1 Disk Basic, typing DOS and the enter key will load and execute the program. Using Disk Basic 1.0, there are two alternatives. If you presently own the OS-9 operating system, merely insert the boot disk and run it. Otherwise you must create your own boot disk using the 21 line program described in the documentation: such a disk could just as easily have been provided by the manufacturer as a convenience to the user. Troubleshooting hints are enclosed in case the program does not load properly. My sample copy showed no hesitation in loading.

Instructions for playing Trivia Fever can be accessed by typing Y at the first prompt. If you want to review parts of the rules, these may be restarted by pressing I. Once the Trivia Master is chosen and the individuals or teams entered, the disk should be turned over and reinserted. At this point, play begins.

There are several features in Trivia Fever that enhance game play. A built-in timer counts down the amount of time a contestant has left to answer a question. Current scores can be handicapped in three different ways: 1) by the number of points necessary to complete a category; 2) random selection of question categories; and 3) by the amount of time given a specific player to answer a question.

The object of the game is to successfully complete questions in each of five subject categories. Questions may be chosen from three levels of difficulty. Once the level is chosen, the computer picks a random query. If a question repeats itself during the contest, it may be scratched by the Trivia Master and another can be chosen. Witty little remarks are made after each question is answered, such as "Don't get too excited, everybody gets that one."

Overall, Trivia Fever appears to be well organized and has clear instructions. It shares enough similar traits with another famous trivia game that an experienced player should be able to proceed with minimal instruction. The self-scoring timing and handicapping are desirable features because they handle some of the game's tedious tasks automatically. Still, the human intervention of the Trivia Master is important, as he or she becomes the moderator and final judge of the game events.

Looking beyond the basic package, there seem to be enough additional question disks available to keep a trivia buff occupied for quite awhile. An enclosed order blank lists ten more disks of trivia questions that can be ordered from PSI for $24.95 each. Volumes 2–4 are currently available. Topical disks include Trivia Fever Super Sports, Trivia Fever Entertainment, and Trivia What's in a Word, available at $39.95. Multiple Learning Disks, listed as "to be announced," contain various questions in academic areas. Perhaps parents and teachers will be able to cash in on the trivia game popularity for reviewing and memorizing facts.

I have one suggestion for PSI: make a do-it-yourself system disk that would allow the student, teacher or parent to enter their own categories and questions. *(end)*

Price Changes Occur On A Daily Basis. Please Call 1-800-343-8841

PRICE BREAKTHROUGH

Price Changes Occur On A Daily Basis. Please Call 1-800-343-8841

MEGADISK™ HARD DISK DRIVE SYSTEMS

TOLL FREE ORDERING 1-800-343-8841

For the **IBM/PC**, Tandy 1000, **TRS/80 Models I/III/IV/4P**, Compaq, Eagle, Sanyo, Tava, PC Workalikes, **Color Computers**, Apple/Franklin, Heath/Zenith, Max/80

Complete with Hardware, Cables, Software and Quikfit Installation

5 Megabytes Internal Mount	starting at	$ 449.95
10 Megabytes Internal Mount	starting at	599.95
20 Megabytes Internal Mount	starting at	1,199.95
5 Megabytes External System	starting at	649.95
10 Megabytes External System	starting at	849.95
20 Megabytes External System	starting at	1,299.95
3.3 Megabytes Kodak Backup System	starting at	599.95
Streaming Tape Backup System	starting at	899.95

DOS Systems Available: IBM/Heath — DOS, 1.0, 2.0, 2.1, 3.0, or later — Apple Franklin — DOS 3.3, Prodos TRS/80-LDOS, TRSDOS 6.x, Newdos/80, Dosplus, CP/M, COCO DOS, Max/80 LDOS, OS9

WARRANTIED FOR ONE FULL YEAR — PARTS AND LABOR — 24 HOUR SERVICE — CALL TOLL FREE — 1-800-343-8841

FLOPPY DISK DRIVES

Our Disk Drives are UL approved — Our Floppy Drive Cabinets and Power Supplies are Underwriters Laboratory Listed and have passed the required Federal Communications Part 15 Section B-EMI/RFI tests.
Warranty on all disk drives is one full years parts and labor. Warranty on floppy disk drive power supplies is five (5) years. In warranty or out of warranty service is 24 hour turn-a-round on all disk drives.

Tandon — Full Height Drives		
100-1	Single Sided 40 tk Bare	$128.00
	In Case with Power Supply	169.95
	Dual Drives in One Cabinet	329.95
100-2	Dual Sided 40 tk Bare	165.00
	In Case with Power Supply	209.95
	Dual Drives in One Cabinet	384.95
	Half High Drives	
65-1	Single Sided 40 tk Bare	125.00
	In Case with Power Supply	159.95
	Dual Drives in One Cabinet	309.95
65-2	**Dual Sided 40 tk Bare**	**145.00**
	In Case with Power Supply	**179.95**
	Dual Drives in One Cabinet	**349.95**
65-4	Dual Sided 80 tk Bare	165.00
	In Case with Power Supply	199.95
	Dual Drives in One Cabinet	359.95
TEAC Half High Drives		
55A	Single Sided 40 tk Bare	140.00
	In Case with Power Supply	169.95
	Dual Drives in One Cabinet	319.95
55B	Dual Sided 40 tk Bare	160.00
	In Case with Power Supply	199.95
	Dual Drives in One Cabinet	359.95
55F	Dual Sided 80 tk Bare	180.00
	In Case with Power Supply	219.95
	Dual Drives in One Cabinet	389.95
Apple/Franklin Disk Drives		
	35/40 Track in Case with Cable and Software	159.95
	Controller Card for Two Disk Drives	49.95
	Combination Price for Disk Drive and Controller	199.95
Commodore Disk Drives		239.95

COLOR COMPUTER DISK DRIVE SYSTEMS AND ADD IN PRODUCTS

40 Track Single Head Drive with Case, Power Supply, Cable Controller, Instruction Booklet, Diskettes	$279.95
Above with Dual Drives in One Cabinet	413.95
40 Track Dual Head with Case, Power Supply, Cable, Controller, Instruction Booklet, Diskettes	**289.95**
Above with Dual Drives in One Cabinet	**429.95**
128 Memory Upgrade Kit	129.95
Dual DOS Switch	19.95
With Second DOS System — JDOS, RSDOS, Micro DOS	48.00

TRS/80 HARDWARE

Model I Starter System — Delivered by UPS One Single Sided Disk Drive, Case, Cable, Power Supply, TRSDOS 1.3 and Manual	**$220.00**
Model III/IV Easy to Install Disk Drive Systems	309.95
Memory Upgrades — 4116 and 4164	starting at 1.50 ea.

ALL IN-STOCK ITEMS SHIPPED WITHIN 24 HOURS. SAME DAY SHIPPING PROVIDED BY REQUEST WITHOUT ANY EXTRA HANDLING CHARGES.

COMPUTERS

All of our computers come with a full one year Warranty — Parts and Labor — 24 Hour Service on all in stock parts.

IBM/PC-256K Two Floppy Drives, One Year Warranty	$1,895.00
With Monitor Card and High Resolution Monitor	2,195.00
IBM/PC-256K 5 Megabyte Drive, One Floppy Drive	2,495.00
With Monitor Card and Monochrome Monitor	2,695.00
10 Megabyte System	2,695.00
With Monitor Card and Monochrome Monitor	2,995.00
IBM Lookalike Monitor 800h x 650v resolution w/cable	199.00
TRS/80 Model IV Computer 128K Dual Drive RS232	1,095.00

PRINTERS

Dot Matrix	
Star Micronics	
Gemini X-Series Parallel 120 CPS	starting at $259.95
Delta 10/15 160 CPS	starting at 424.95
Radix 10/15 200 CPS	starting at 649.95
Panasonic 1090	249.95
Daisy Wheel	
Silver Reed 440 80 Column 12 CPS	315.95
550 132 Column 19 CPS	439.95
770 132 Column 36 CPS	895.00
Olympia 132 Column 14 CPS with Form and Tractor	399.95
Star Micronics 100 Column 18 CPS	352.95
Apple/Franklin Printer Interface w/Graphics and Cable	84.95
Printer Cables	starting at 19.95
Printer Paper — Microperf Edge 1000 Sheets	16.95

ELECTRICAL

Surge Protectors — Line Filters — SL Waber — 6 Outlets with Switch	$ 39.95
Uninterruptable Power Supplies — Panamax 250 Watt-15 Minute	399.95

MODEMS

Volksmodem 300 Baud	$ 59.95
Signalman Mark X Audodial	123.95
Mark XII 1200/300 Baud Autodial	284.95

ADD IN BOARDS FOR THE IBM

STB — Rio Plus 128K	$299.95
Super Rio 128K	219.95
Graphix Plus	283.00
Graphics Plus II	355.00
Parallel Printer Port	74.95
Serial Port	84.95
Monochrome Board with Parallel Printer Port	210.00
Quadram — Quadboard 512 128K	269.95
Quadcolor I	215.95
Quadcolor II	439.95

MISCELLANEOUS

Diskettes in 10 Pack	**from $ 12.95**
Twoprint Switches	from 99.95
Disk Drive Cables	from 16.00
Maintenance Cleaning Kits	12.00
Parallel Printer Buffers 16K	149.95
Floppy Disk Drive Cables	
1 Drive	16.00
2 Drives	18.95
Heath/Zenith 2 Drive Cables — Shielded	39.95

SOFTWARE SUPPORT, INC.

1 Edgell Road, Framingham, MA 01701 (617) 872-9090 Telex-383425

Hours: Mon. thru Fri. 9:30 am to 5:30 pm (E.S.T.) Sat. 10 am to 3:30 pm

SERVICE POLICY — Our Professional Technical Staff Is Available To Assist You Monday Through Saturday.
WARRANTIES — Disk Drives — One Full Year Parts And Labor. Floppy Disk Drive Power Supplies — Five (5) Years.
SERVICE — 24 Hour Turn-A-Round On All In-Stock Parts. Price Changes Occur On A Daily Basis. Please Call

Toll Free 1-800-343-8841

For Our Latest Price Saving Specials.

DYNACALC®

CoCo's Best & Fastest Spreadsheet System

ACCLAIMED BY THE EXPERTS

"DYNACALC is my choice for a CoCo spreadsheet."
Dan Downard, RAINBOW, September, 1984.

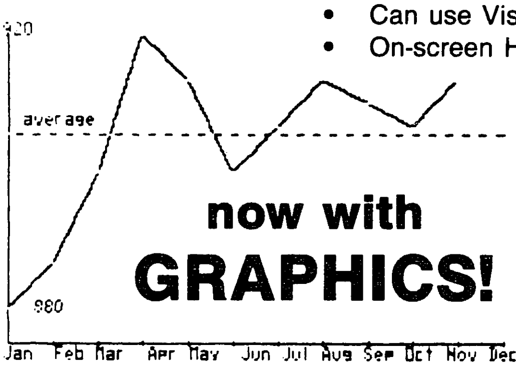
"Eat your heart out, Lotus 1-2-3!"
Scott Norman, HOT CoCo, October, 1984.

Built-in Features:

- 51 x 24 Display with Lower Case
- Super-fast Smart Screen Refresh
- Auto-Repeat Keyboard Driver
- Keysaver (TM) repeats last command x times
- Disk Operating System (works just like ROM DOS)
- Easy communication with BASIC/DOS programs
- Two-way communications with PRO-COLOR-FILE ★Enhanced★
- Outputs to ASCII Word Processors like Telewriter-64
- Fast 16-Digit Arithmetic with Scientific Functions
- Summation, Mean, and Standard Deviation Functions
- Logical Functions with String & Numeric Comparison
- String locate command to navigate large worksheets
- Sort full or partial worksheet by columns or rows
- Line, Bar, Hi/Lo/Close, Circle Graphs
- Full Graphics captioning and overlay facility
- Graphics Drivers for all popular Printers
- Joystick/Mouse Driver for Cursor Movement
- Works with any ROM versions, even JDOS
- 33k Available Worksheet Space
- Up to 256 Columns or 256 Rows
- Can use VisiCalc worksheets & training materials
- On-screen Help Messages

$99.95

RAINBOW
CERTIFICATION SEAL



FOR 64K DISK SYSTEMS

available from

COMPUTER SYSTEMS CENTER

NEW ADDRESS

42 Four Seasons Center #122
Chesterfield, MO 63017
(314) 576-5020
10 to 6 Mon.-Fri.

CSC

or your local DYNACALC dealer

Price US$99.95
Outside North America add $10 postage

CANADA- CDN $129.95
RGS MICRO INC.
759, VICTORIA SQUARE 405
MONTREAL H2Y 2J3
TEL.: (514) 287-1563
ORDER LINE ONLY ★ ★ ★
QUEBEC-ONTARIO-MARITIMES
800-361-5338
WESTERN CANADA 800-361-5155

MasterCard
VISA

DYNACALC Reg. U. S. Pat. Off.
Lotus and 1-2-3 are trademarks of Lotus Development Corp.
PRO-COLOR-FILE is a trademark of Derringer Software
Telewriter-64 is a trademark of Cognitec
VisiCalc is a trademark of VisiCorp.