FORTH Dimensions

Volume 6, Number 3

September/October 1984
$2.50

Continued Fractions

Re-Defining a Colon Word

Write Like a Fox

Forth-79 Programs → Forth-83

More Debugging

Introducing 3 New 68000 FORTH Systems

Multi-FORTH™ Under CP/M™-68K

- Available Now
- 32-bit Multitasking FORTH System
- Snapshot features allow creation of executable CP/M programs (in under 5 seconds from memory)
- Full access to CP/M files and facilities
- Inline Macro Assembler
- Line, Screen Editors, other utilities
- Introductory price.....**$695.00** (Regularly $895.00)

MacFORTH™

- Stand Alone Programming on Macintosh
- User defined MENUS
- User defined WINDOWS
- Compatible with Macintosh User Interface
- Mac File Compatability
- Interactive COMPILER & INTERPRETER
- User Access to TOOLBOX
- 32-bit FORTH System
- Large Installed Base with Local User Groups
- Available Now.....**$149.00**

Multi-FORTH™ Under UNIX™

The Speed and Flexibility of FORTH with the Structure and Utilities of UNIX™

- Comprehensive 32-bit FORTH Environment Under UNIX
- Full access to UNIX File Features and Shell from FORTH (ls, grep, rm, cat, etc.)
- Multi-FORTH Core Resident Real time Multitasker
- Inline Macro Assembler
- Ability to snapshot memory image as a loadable application
- Available in July on the Perkin Elmer 7300 (soon on other UNIPLUS Hosts)
- Color Graphics on the Perkin Elmer version
- Introductory price.....**$1295.00**

68000 FORTH Systems also available on HP Series 200 and Motorola VME10

For more information contact

CREATIVE SOLUTIONS

4701 Randolph Rd. Ste.12 Rockville, Maryland 20852

(301)984-0262

UNIX is a registered trademark of AT&T • CP/M is a registered trademark of Digital Research

FORTH Dimensions
Published by the
Forth Interest Group
Volume VI, Number 3
September/October 1984

Editor
Marlin Ouverson

Production
Jane A. McKean et al.

Forth Dimensions solicits editorial material, comments and letters. No responsibility is assumed for accuracy of material submitted. Unless noted otherwise, material published by the Forth Interest Group is in the public domain. Such material may be reproduced with credit given to the author and the Forth Interest Group.
Subscription to *Forth Dimensions* is free with membership in the Forth Interest Group at $15.00 per year ($27.00 foreign air). For membership, change of address and/or to submit material for publication, the address is: Forth Interest Group, P.O. Box 1105, San Carlos, California 94070.

Symbol Table

Simple; introductory tutorials and simple applications of Forth.

Intermediate; articles and code for more complex applications, and tutorials on generally difficult topics.

Advanced; requiring study and a thorough understanding of Forth.

Code and examples conform to Forth-83 standard.

Code and examples conform to Forth-79 standard.

Code and examples conform to fig-FORTH.

Deals with new proposals and modifications to standard Forth systems.

FORTH Dimensions

FEATURES

DEPARTMENTS

FOR TRS-80 MODELS 1, 3 & 4
IBM PC, XT, AND COMPAQ

The MMSFORTH System. Compare.

- The speed, compactness and extensibility of the MMSFORTH total software environment, optimized for the popular IBM PC and TRS-80 Models 1, 3 and 4.
- An integrated system of sophisticated application programs: word processing, database management, communications, general ledger and more, all with powerful capabilities, surprising speed and ease of use.
- With source code, for custom modifications by you or MMS.
- The famous MMS support, including detailed manuals and examples, telephone tips, additional programs and inexpensive program updates, User Groups worldwide, the MMSFORTH Newsletter, Forth-related books, workshops and professional consulting.

mmsFORTH

A World of Difference!

- Personal licensing for TRS-80: $129.95 for MMSFORTH, or "3+4TH" User System with FORTHWRITE, DATAHANDLER and FORTHCOM for $399.95.
- Personal licensing for IBM PC: $249.95 for MMSFORTH, or enhanced "3+4TH" User System with FORTHWRITE, DATAHANDLER-PLUS and FORTHCOM for $549.95.
- Corporate Site License Extensions from $1,000.

If you recognize the difference and want to profit from it, ask us or your dealer about the world of MMSFORTH.

MILLER MICROCOMPUTER SERVICES
61 Lake Shore Road, Natick, MA 01760
(617) 653-6136

the Authors

Robert Berkey is a systems software engineer at Dysan Corporation. He is the treasurer of the Forth Standards Team, participated on the fig-FORTH implementation team and is the author of the Alpha Micro fig-FORTH listing.

Nathaniel Grossman is a professor of nathematics at UCLA where he teaches oth graduate and undergraduate stulents. He uses Forth for personal and emi-professional purposes, and enjoys eading, thinking and fussing over a mall model railroad.

John D. Hall is the National Chapter Coordinator for the Forth Interest Group.

Michael Ham is manager of customer atisfaction at Dysan Corporation, and ists documentation design as one of his esponsibilities of interest. He designed a national data base of continuing education records, and wrote a college earch program in Forth.

Henry Laxen is Vice-President of Research and Development for Paralise Systems, Inc. He worked on the operating system for the Panasonic/ Quasar HHC, the world's first handeld computer. He will soon participate n a tango competition, and has a cat named Sophie who sounds like a bird.

William F. Ragsdale was the foundng President of the Forth Interest Group. As the author of *fig-FORTH Installation Manual and Model*, his work has been translated to run on eleven processors.

Ed Schmauch is a research chemist with Conoco,Inc. where he investigates methods of controlling corrosion during the production of petroleum products. He has developed on a portable computer various real-time Forth programs which were in turn used as debugging aids for other computer systems.

Letters to the Editor

Value Needn't Be Pricey

Dear FIG:

I would like to thank Henry Laxen and Michael Perry for their F83 implementation. It is receiving considerable support in the Minnesota FIG Chapter. Its inclusion of complete source code is a great benefit few vendors offer. Its many enhancements and low price make it an unbeatable system. The understandable lack of support could largely be compensated for if a user network of some kind develops. Suggestions, anyone?

I must comment on Bill Ragsdale's comments on the price of a software package as an indicator of its value (V/6). While it is a clue, I don't think the conclusion drawn ("...Forth...has only one-fifth the value [of] a C system.") is justified. Price is also an artifact of the market system. The simplicity and elegance of Forth makes it easier to implement with fewer development resources, which results in cheaper implementations being possible. I hope vendors don't conclude that if they only charge more they will sell more, clearly a fallacious conclusion. Personally, I think software should be priced near the cost of the medium on which it resides, and programmers should be remunerated from the proceeds of some other financial mechanism.

Fred H. Olson
1221 Russell Avenue N.
Minneapolis, Minnesota 55411

A Cure is in Sight

Editor:

I am very pleased with the new format for *Forth Dimensions*. It certainly makes things easier to read and to find articles at my level.

I have one suggestion. In your column "Ask the Doctor" could you make the "Rx" stand out in some manner. I find that sometimes I am into the answer before I realize I left the question.

Keep up the good work.

Sincerely,

David H. Lawson
219 N. Vanderhurst
King City, California 93930

We aim to please — this month's column format should be a bit easier to read...-Ed.

Sixth Annual
Forth National Convention

November 16–17, 1984

Hyatt Palo Alto

4290 El Camino Real, Palo Alto, CA 94306 USA

Learn about Forth and make your life easier. The convention will show you how!

- **Exhibits**
- **Speakers**
- **Tutorials**
- **Vendor Meetings**
- **Panel Discussions**
- **Equipment Demonstrations**
- **Discussion Groups**
- **Worldwide FIG Meeting**
- **Banquets**
- **Awards**

Forth is for everyone. The Forth computer language is used in video games, operating systems, real-time control, word processing, spread sheet programs, business packages, DBMS, robotics, engineering and scientific calculations and more.

Coverage of Forth applications, Forth-based instruments, Forth-based operating systems, and more.

Speak at the convention. Those wishing to participate and be speakers and/or panelists are urged to contact the program coordinator immediately. (Telephone the FIG hotline 415/962-8653.)

PROGRAM

FRIDAY, November 16		**SATURDAY, November 17**	
EXHIBITS	**Noon — 6 pm**	**EXHIBITS**	**9 am — 5 pm**
11:30 am	**Registration**	**10 am**	**Forth Resources**
1 pm	**Forth Systems**	**11 am**	**Education**
2 pm	**Data Base Developments**	**Noon**	**Lunch**
3 pm	**Forth-Based Products**	**1 pm**	**Forth Chips and Computers**
4 pm	**Forth-Based Products**	**2 pm**	**Business Applications**
5 pm	**32 Bit Systems**	**3 pm**	**Forth Chapters**
6 pm	**Exhibits Close**	**4 pm**	**Forth-83 Standard, FORML Preview**
		5 pm	**Exhibits Close**

BANQUET

7 pm Saturday — Reservation and payment required — $30.00

Convention preregistration is $10.00; or $15.00 at the door. Special convention room rates are available at the Hyatt Palo Alto. Telephone direct to Hyatt reservations by calling (800) 228-9000 and request the special Forth Interest Group Convention rates for November 16th and 17th.

The Forth Convention is sponsored by the Forth Interest Group (FIG). The Forth Interest Group is a non-profit organization of over 4800 members and 50 chapters worldwide, devoted to the dissemination of Forth-related information. FIG membership of $15.00/year ($27.00 overseas) includes a one-year subscription to **FORTH Dimensions**, the bimonthly publication of the group.

- -

☐ Yes! I will attend the Forth Convention.

☐ Number of pre-registered admissions ____ × $10.00 each $ ________

☐ Number of Banquet Tickets ____ × $30.00 each ________

☐ Yes! I want to join FIG and receive **FORTH Dimensions** ($15.00 US, $27.00 foreign) ________

TOTAL CHECK TO FIG $ ________

☐ I want to exhibit; please send exhibitor information.

Name ______________________________

Address ______________________________

Company ______________________________

City ______________________ State ________ Zip ____________

Phone (   ) ______________________

Return to: **Forth Interest Group**, P.O. Box 1105, San Carlos, CA 94070 • **415/962-8653**

Editorial

Credit Where It's Due

Forth Dimensions is evolving along with the Forth Interest Group, and it requires the care and labor of a fairly large group of people, mostly FIG board members, attendees at the monthly FIG business meetings, correspondents and other volunteers. It seems a good idea to let you know how it all manages to come together.

We receive letters and articles (but never enough!) in the mail. These are collected by Shepherd Associates (who is also responsible for all the advertising content) and are passed to me along with press releases, requests for reprint privileges and other miscellaneous mail for the editor. Often included among this material are requests for specific information about a Forth system: from how to save blocks to diskette on an Apple system, to which headerless compiler is right for a given application. Unfortunately, except for items answered in Bill Ragsdale's "Ask the Doctor" column, we cannot always give the kind of individual attention we'd like. The best source of such information is often the vendor, a users group, the local FIG chapter, the FIG hotline, the FIG Tree or any class on Forth programming.

Material under consideration for publication is taken to Forth experts who offer technical and literary criticism. Ray Duncan and Kim Harris have often assisted graciously in this task, although a considerably larger number contributes as needed. We look for sound programming technique, clarity of both the code and the text, interesting new solutions which demonstrate the problem-solving process, small applications which illustrate one or more Forth techniques, and good tutorials. Frequently frowned at are articles which simply re-hash an old subject with little new to offer. The reviewers also usually shy away from code presented in dialects of Forth that few readers can use directly, or which contains undocumented non-standard words and extensions. Sometimes the reviewers will like the basic concepts of a piece but will make suggestions to the author and ask him to consider revising the piece. Well-written tutorials for beginners are rarely received, but are almost always received enthusiastically.

Articles accepted for publication are tentatively scheduled for a particular issue. In the interests of timeliness, *Forth Dimensions* does not work as far ahead as some periodicals, but we are always about two issues ahead of our readers in terms of the ones we are putting together. When you receive this issue we will be preparing Volume VI, Number 4 for the printer, Number 5 will be waiting in the wings and we will be planning and reviewing articles for Number 6.

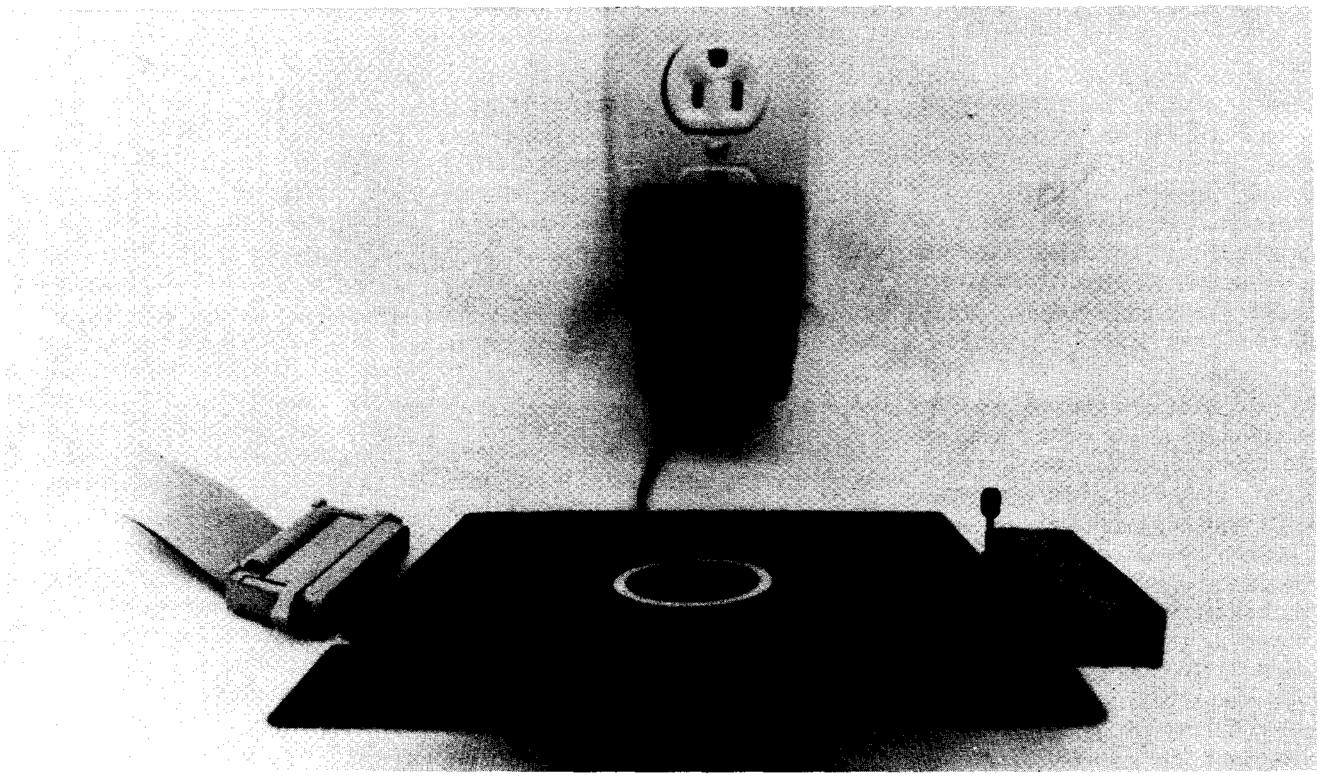
Jane McKean is the one responsible for taking the abstract concept of the new design of *Forth Dimensions* and turning it into the concrete form you hold in your hands. After an article's spelling is corrected, the capitalization fixed and all questions answered, it is transmitted to Jane's computer via modem or is handed to her as hard copy. She then enters formatting codes and uploads it to a nearby commercial typesetter. The typeset articles come back on long pages called galleys, and are proofread by yours truly before a final version is run off. Jane then tries to make the brew we've concocted fit within the given length of the magazine, a labor sometimes only requiring a shoe horn, but often calling for a crowbar wielded with surgical skill (my eyes are sometimes bigger than the plates). If you can imagine a combination of a crossword and fifteen-tile puzzle with elements of Risk and Rubik's Cube, you have a taste of what it's like to do the physical production of a magazine.

There is one more inspection of the entire issue before it goes to the printer. Labels are generated by C.J. Street & Associates, who provides our mailing list service; and the issue is printed and bound by Technical Publishing just as editorial and production are starting to put the next issue together.

In a nutshell, that is how each issue gets put together. Of course, I haven't mentioned our faithful columnists who endure my preaching about the significance of deadlines. (I really *do* need it by Friday, gang, no kidding!) The names John Hall, Henry Laxen, Bill Ragsdale and Robert Reiling appear regularly. Not mentioned often enough are the many other authors whose work also is freely given to improve the body of Forth literature and tools. Perhaps the best way to thank them is to return the favor!

—*Marlin Ouverson*
Editor

THE MOST ADVANCED VERSION OF FORTH IS NOT ON THIS 5 1/4" DISK.



IT'S UNDERNEATH IT!

When you pay $300 for some languages all you get is a $3.00 disk. When you buy the NMIX-0012 RSC-FORTH System from New Micros, Inc. you not only get the language, you get a complete computer system. It has features that you can't expect from a disk based language, like an RS 232 serial interface, 40 individually programmable input/output lines (5 parallel ports), two counter/timers, RAM, ROM and a EEPROM/EPROM programmer. For that matter, disks don't usually come with their own power supplies, and almost none are packaged in rugged metal cases! You would expect a disk to come with hundreds of pages of documentation — our system certainly does. With a disk, you have to tie up your whole computer to run the program. After storing your program in our system, it can be dedicated to a task without the support of your PC, or you can communicate with it over the same serial channel used to program it. On top of all this our FORTH has advanced features, like built-in target compilation, CASE statements, PROM programming words and, believe it or not, disk access functions. The NMIX-0012 is 100 millimeters on a side, smaller than a disk, priced at $290 complete. The "100 Squared"™ is the logical choice! Substantial quantity discounts.

ORDER NOW FROM: New Micros, Inc.
808 Dalworth
Grand Prairie, Texas 75050
(214) 642-5494

Operations

Have you ever wondered how the Forth Interest Group accomplishes all its member service activities? Dedicated volunteers perform many functions; for example, the Board of Directors and Officers are all volunteer positions. But this group cannot provide all the support needed to keep the Forth Interest Group running smoothly; therefore, they have entered into agreements with specialists who are able to perform the ongoing daily tasks that are necessary.

Early this year the Board selected an association management firm, Shepherd Associates, to perform association services for the Forth Interest Group. For the past several years, Martens and Associates had performed this service, but because of other commitments they wanted to stop. Shepherd Associates is experienced in association work and now handles the daily activities of the Forth Interest Group, doing such things as entering *Forth Dimensions* subscription renewals and new subscriptions, filling orders for publications and products, and answering the hot line. They help the Chapter Coordinator, John Hall, do mailings to chapter coordinators throughout the world. They also solicit advertising for insertion in Forth Interest Group publications. In July, the Board extended the initial contract to the end of 1984.

The editor of *Forth Dimensions*, Marlin Ouverson, is one of the professionals supporting the Forth Interest Group. He is responsible for editing *Forth Dimensions* and making certain that it is ready for the printer on schedule. You probably have noticed some of the new-look features that Marlin has added to *Forth Dimensions*. Unique department headings, the thermometer to indicate the technical level of an article, and symbols to indicate the Forth dialect, for example.

Spreading the word about the Forth Interest Group activities is publicist Linda Kahn. She is in contact with magazine editors regularly in order to alert them to current events and new information about Forth. If you have an article about Forth, Linda can probably suggest a publisher who would be interested in it. She also sends a regular stream of press release material to the press.

These are some of the support functions that are contracted by the Forth Interest Group. They keep things going.

—*Robert Reiling*
President

P.S. Don't forget the upcoming events: the Sixth Annual Forth Interest Group Convention November 16-17, 1984 at the Hyatt Palo Alto in Palo Alto, California; and the FORML Conference to be held at Asilomar, Pacific Grove, California, November 23-25, 1984.

The Programmer's Quiz

Question:

Name one 32 bit Realtime Operating / Development System for which 128 kb of RAM and Floppy Disk is an enormous computer.

Answer:

4xFORTH for the MC68000, a complete ROM based operating system with

multi-user & tasking,

dynamically selectable system's device,

error checking assembler,

ring buffered input,

ram disk, and much more.

4xFORTH which meets the '83 Forth Standard, except with 32 bit variables,

4xFORTH, a realtime tool for the professional programmer.

by

The Dragon Group, Inc
148 Poca Fork Road
Elkview, WV 25071

304/965-5571

Long Divisors and Short Fractions

Nathaniel Grossman
Los Angeles, California

Readers of *Starting Forth, Forth Tools* and *Forth Dimensions* understand the power of the scaling operation ***/** and will value the convenience of simple rational approximations to thirty-two-bit floating-point constants. *Starting Forth* [5, p. 122] contains a short table of useful approximations. *Forth Tools* [2, p. 104] and *Forth Dimensions* [6] contain similar tables.

For example, $\sqrt{2} \cong 1.4142135624$. *Starting Forth* suggests the approximation $19601/13860 \cong 1.4142135642$. The method to be explained in this article obtains the approximation 19601/13860 as well as better approximations, among them $47321/33461 \cong 1.4142135621$, which has an error $\cong 3 \times 10^{-10}$. This method, part of the theory of *continued fractions* and well known to mathematicians, furnishes simple, iterative algorithms for grinding out short, rational approximations to long and complicated rational numbers and even to other classes of numbers such as square roots of integers. Implementation of these algorithms on specific computers may run into difficulties when the arithmetic overflows the capacity of the built-in calculator routines. If, however, the numerical givens and the iterative schemes are carefully analyzed, the difficulties are seen to be removable.

This article presents both an analysis to show that the continued fraction algorithm can be implemented in double-precision integer Forth and, best of all, an implementation in Forth-83. By loading the implementation, any Forth user can unleash continued fraction power.

The rest of this article is in two parts. To get the full power from the algorithms, the standard Forth arithmetic arsenal must be extended by a word **UD/MOD** that expects unsigned double dividend and divisor and returns their unsigned double quotient and remainder. (Thus, **UD/MOD** is a superword of **/MOD**.) At the same time, it is convenient to introduce a new class of stack manipulators that I call *long pair* words. These words, distinguished by the prefix **LP**, move pairs of adjacent double numbers around the stack. The **LP** words and **UD/MOD** have an independent value outside the continued fraction calculations. They can be viewed as quadruple-precision words of a sort, but I decided not to use David Beers' quadruple precision words [3] because I did not need triple and quadruple addition and subtraction or signed multiplication and division. These matters compose Part I.

Part II begins with a short introduction to continued fractions in just enough detail to justify their use in constructing rational approximations. Both strengths and weaknesses are included. This leads into a discussion of the specific algorithm and of its implementation in Forth.

Part II requires Part I, but the first part can stand alone as a useful addition to the Forth dictionary. You may be wavering about reading either or both. In that case, turn first to the Intermezzo between the two parts. You will find there a single, stand-alone screen. Load it and read the description; run it and you will get a concise demonstration of the power of the full continued fraction program.

Part I
Long Pairs and Long Division

Although the Forth Required Word Set and the Double Number Extension Word Set are rich in arithmetic words, no words are included to carry out the division of one (signed or unsigned) double integer by another. Such words are needed occasionally. The Forth way to fill such a need is clear: extend the dictionary with new words. Indeed, two tries at writing double division words are available in back issues of *Forth Dimensions*. L.H. Bieman [4] wrote screens of double precision mathematics words, but an accompanying commentary on the words by Robert L. Smith indicates that Bieman's word **D/** for division of double by double can produce errors of several units in the least significant digit of the quotient. Because I need full accuracy in the quotient, Bieman's **D/** is of no help to me. (I continue to use his word **U*/** with gratitude.) And, as I have already mentioned, David Beers' quadruple words [3] are too lush for my needs. Therefore, I have written a word **UD/MOD** to fill my needs exactly.

A Long Division Algorithm

The goal is a word **UD/MOD** that expects unsigned double dividend and divisor and returns the unsigned double quotient and remainder from the division.

Of course, the literature contains many descriptions of multi-precision division algorithms. I read the appropriate section in Knuth's standard treatise [8] but I did not feel that the algorithm Knuth recommends would be best for implementing in Forth. Nevertheless, I could make good use of some of the many goodies with which Knuth lards his text. The algorithm I implemented is improved by a long division algorithm (for use on geared mechanical calculators) presented in a mathematical handbook [1, p. 21]. Algorithms with such provenance usually require an additional logical co-processor —the human brain! It is toward doing away with human intervention that the tidbits from Knuth are applied.

Suppose that u and v are integers, with v positive, u non-negative, and both less than 2^{32} (double integers). It is required to divide u by v and return the quotient and remainder as unsigned double integers.

Knuth advises using the division of a two-digit integer by a two-digit integer as a model. His algorithm requires extension of the dividend to a three-digit integer, and I want to avoid this higher precision extension. Now, thirty-two binary digits seem more than two, but they are not really much more from the right point of view. Introduce a *superbase* $b = 2^{16} = 65336$. Any double integer may then be written as $u = u_0 * b + u$, where $0 \leq u_0 < b$ and $0 \leq u_1 < b$; u_0 and u_1 are the "superdigits." Similarly, $v = v_0 * b + v_1$ and $u/v = (u_0 * b + u_1)/(v_0 * b + v_1)$.

$$
\begin{aligned}
\frac{u}{v} &= \frac{d * u}{w} \\
&= \frac{d * (u_0 * b + u_1)}{w_0 * b + w_1} \\
&= \frac{d}{w_0 * b} * \frac{u_0 * b + u_1}{1 + (w_1 \,/\, w_0 * b)} \\
&= \frac{d}{w_0 * b} * \{u_0 * b + u_1\} * \left\{1 - \frac{w_1}{w_0 * b} + E\right\} \\
&= \frac{d}{w_0 * b} * \left\{u - \frac{u_0 * w_1}{w_0} - \frac{u_1 * w_1}{w_0 * b} + u * E\right\}
\end{aligned}
$$

Figure One

The problem would be essentially done if Forth, which can talk in so many bases, could manipulate in base b. But it cannot.

The next step is analytic. There is a simple formula: $(1 + x)^{-1} = 1 - x + \text{error}$, where if $0 \leq x < 1$ then $0 \leq \text{error} < x^2$. Before we use this formula, we are going to scale the fraction u/v for a reason that will be easier to explain a few paragraphs farther along. We want to multiply dividend and divisor by a scaling integer d that makes dv close to $b^2 = 2^{32}$, so close that $\frac{1}{2}b^2 \leq dv < b^2$. The exact choice of d is not crucial: I take Knuth's suggestion, selecting $d = [b/(v_0 + 1)]$ (where [x] is the floor of x, the largest integer not bigger than x). Let $w = dv$ and write $w = w_0 * b + w_1$; now $\frac{1}{2}b \leq w_0 < b$ and $u/v = (d * u)/w$. (We do *not* multiply d and u together immediately to form a new dividend because there is no guarantee that d * u will not stretch to more than thirty-two bits.)

With this preparation, the analytic unfolding of the division is easy (see figure one).

Now we will show that the last two terms in the braces can be omitted if we want only the *quotient* [u/v]. This is where the scaling is important. We know that $1 \leq d \leq b$, $0 \leq w < b$, $0 \leq u_1 < b$, and $\frac{1}{2}b \leq w_0$. Also, $0 \leq E < (w_1/w_0 * b)^2$.

Then

$$
\begin{aligned}
0 &\leq \left\{\frac{d}{w_0 * b}\right\} * \left\{\frac{u_1 * w_1}{w_0 * b}\right\} \\
&\leq \left\{\frac{b}{\frac{1}{2}b^2}\right\} * \left\{\frac{b^2}{\frac{1}{2}b^2}\right\} \\
&= \frac{4}{b}.
\end{aligned}
$$

and

$$
\begin{aligned}
0 &\leq \left\{\frac{d}{w_0 * b}\right\} * u * \left\{\frac{w_1}{w_0 * b}\right\}^2 \\
&\leq \left\{\frac{b}{\frac{1}{2}b^2}\right\} * b^2 * \left\{\frac{b^2}{\frac{1}{2}b^2}\right\} \\
&= \frac{8}{b}.
\end{aligned}
$$

Thus

$$
[u/v] = \left\{\frac{d}{w_0 * b}\right\} * \left\{u - \frac{u_0 * w_1}{w_0}\right\}
$$

because the two terms neglected are small (and of opposite signs, which will not be needed here). Having the quotient in hand, we find the remainder simply as u − v * [u/v].

The actual calculation of [u/v] is now easy using the arithmetic operations built into Forth, but the terms must be evaluated in the correct order to ensure full precision. First calculate the bracket as a double integer—Bieman's **U*/** is useful for this—then multiply/divide by d/w_0, again using **U*/**. Division by the superbase $b = 2^{16}$ is then performed by dropping the least significant sixteen bits of the penultimate result.

Implementation of Long Division

The algorithm described above does not call for a complicated implementation. There is one place to take care. Even though the divisor is called thirty-two bits wide, the sixteen high bits may be zero. If this happens, the number w_0 will be zero and the algorithm will call on division by zero. To avoid zero divisions, the algorithm lays down a double track.

The screens are composed in Forth-83, specifically MicroMotion MasterFORTH. The screens should run in Forth-79 if **UM*** is replaced by **U*** and **UM/MOD** by **U/MOD**. MasterFORTH contains a file system and I have placed these screens in the file LONGWORDS.

Screen #2: These are the long pair—**LP**—words for moving pairs of double numbers around the stack. They are analogues of one-cell and two-cell stack words, and their actions should be clear from the stack diagrams.

Screen #3: Here are Bieman's useful words **T***, **T/** and **U*/**.

Screen #4: If the divisor in fact fits into sixteen bits, then [u/v] can be computed with no further extension of the dictionary by the word **NARROW_UD/ MOD**. The word **US>D** converts an unsigned single to a double, and it is necessary because **S>D** reads the sixteenth bit as a − (minus) sign and extends to a double accordingly. Note the double constant **SUPERBASE**.

Screen #5: The actual long division begins on this screen. The word **UD/ MOD_TUCK** disassembles the high and low parts of the dividend and divisor and saves those cells needed later. Then **UD/MOD_DENSCALE** calculates the number d, which **SCALE_DEN** uses to scale the denominator. The quotient, at least if w needs thirty-two bits, is a consequence of **WIDE_QUOT**, which carries out the algorithm worked out above.

Screen #6: **WIDE_UD/MOD** carries out the full thirty-two-bit division operation and returns quotient and remainder.

Screen #7: Enter double numbers, dividend first and then the divisor, and execute **UD/MOD**. The stack holds the remainder inside the quotient. Example: 3141592654. 1000000000. **CF D. D.** prints first 3 and then 141592654.

(See "Intermezzo," page 15.)

```
SCR # 2
 \ LONGWORDS                                        FORTH 83  09MAR84NG
 \ For manipulating groups of double numbers
 : LPSWAP      \ D1 D2 D3 D4 --- D3 D4 D1 D2
    >R >R 2SWAP   >R >R 2SWAP   R> R> R> R> 2SWAP
    >R >R 2SWAP R> R>   ;
 : LPDUP       \ D1 D2 --- D1 D2 D1 D2
    2DUP >R >R   2OVER R> R>   ;
 : LPOVER      \ D1 D2 D3 D4 --- D1 D2 D3 D4 D1 D2
    >R >R >R >R   LPDUP   R> R> R> R>   LPSWAP   ;
 : LPROT       \ D1 D2 D3 D4 D5 D6 --- D3 D4 D5 D6 D1 D2
    >R >R >R >R LPSWAP   R> R> R> R> LPSWAP   ;
 : LPDROP      \ D1 D2 ---
    DROP DROP DROP DROP   ;



```

```
SCR # 3
 \ LONGWORDS                                        FORTH 83  10MAR84NG
 \ words for manipulating double numbers
 \ double-triple words after L. H. BIEMAN, FORTH Dimensions, V-1
 : UM/     \ UD UN --- UN   divide UD by UN and drop remainder
    UM/MOD SWAP DROP   ;
 : T*      \ UD UN --- UT
    DUP ROT UM* >R >R
            UM*
    0 R> R> D+   ;
 : T/      \ UT UN --- UD
     >R   R@ UM/MOD SWAP
    ROT 0 R@ UM/MOD SWAP
    ROT   R> UM/MOD SWAP DROP
    0 2SWAP SWAP D+   ;
 : U*/     \ UD UN UN --- UD
    >R T* R> T/   ;
```

```
SCR # 4
 \ LONGWORDS                                        FORTH83   10MAR84NG
 \ words for dividing double by double

 : NARROW_UD/MOD
 \ UDividend UNdivisor --- UDremainder UDquotient
    DROP >R 2DUP R@                    \ shuck high cell of divisor
    1 SWAP U*/                         \ UDquotient
    2SWAP 2OVER R> 1 U*/ D- 2SWAP ; \ UDrem UDquot

 VARIABLE NUMH  VARIABLE DENH  VARIABLE DENL  VARIABLE DENSCALE
 2VARIABLE NUM  2VARIABLE DEN
 65536. 2CONSTANT SUPERBASE

 : US>D    \ convert unsigned 16bit to 32bit
    0   ;

```

```
SCR # 5
\ LONGWORDS                                      FORTH 83  10MAR84NG
\ words for dividing of double by double

: UD/MOD_TUCK    \ UD UD ---        save parts of num and den
  2DUP DEN 2! DENH ! DENL !
  2DUP NUM 2! NUMH ! DROP  ;
: UD/MOD_DENSCALE    \      --- UN   for scaling-up den
  SUPERBASE  DENH @ 1+ UM/
   DENSCALE !   ;
: SCALE_DEN     \   ---    multiply denominator by scale factor
  DEN 2@ DENSCALE @ 1 U*/
  DENH ! DENL !    ;
: WIDE_QUOT    \     --- UD   if divisor needs more than 16 bits
  NUM 2@ NUMH @ US>D
   DENL @ DENH @ U*/ D-
  DENSCALE @ DENH @ U*/ SWAP DROP  ;
```

```
SCR # 6
\ LONGWORDS                                      FORTH 83  10MAR84NG
\ words for dividing double by double

: ?NARROW_DIVISOR     \ D --- flag   is divisor <  65536 ?
  DUP 0=    ;
: WIDE_REM    \    ---  UD   remainder in wide division
  DUP NUM 2@ ROT DEN 2@
  ROT 1 U*/ D- ROT US>D ;
: WIDE_UD/MOD     \ UDdividend UDdivisor --- UDrem UDquot
  UD/MOD_TUCK
  UD/MOD_DENSCALE
  SCALE_DEN
  WIDE_QUOT
  WIDE_REM    ;
```

```
SCR # 7
\ LONGWORDS                                      FORTH 83  10MAR84NG
\ division of double by double

: UD/MOD    \ UDdividend UDdivisor --- UDremainder UDquotient
  ?NARROW_DIVISOR
  IF
   NARROW_UD/MOD
  ELSE
   WIDE_UD/MOD
  THEN   ;
: UDMOD     \ UDdividend UDdivisor --- UDremainder
  UD/MOD. 2DROP   ;
: UD/       \ UDdividend UDdivisor --- UDquotient
  UD/MOD  2SWAP 2DROP   ;
```

C64-FORTH/79 New and Improved for the Commodore 64

C64-FORTH/79™ for the Commodore 64-$99.95

- New and improved FORTH-79 implementation with extensions.
- Extension package including lines, circles, scaling, windowing, mixed high res-character graphics and sprite graphics.
- Fully compatible floating point package including arithmetic, relational, logical and transcendental functions.
- String extensions including LEFT$, RIGHT$, and MID$.
- Full feature screen editor and macro assembler.
- Compatible with VIC peripherals including disks, data set, modem, printer and cartridge.
- Expanded 167 page manual with examples and application screens.
- "SAVE TURNKEY" normally allows application program distribution without licensing or royalties.

(Commodore 64 is a trademark of Commodore)

TO ORDER

-Disk only.
-Check, money order, bank card, COD's add $1.65
-Add $4.00 postage and handling in USA and Canada
-Mass. orders add 5% sales tax
-Foreign orders add 20% shipping and handling
-Dealer inquiries welcome

PERFORMANCE MICRO PRODUCTS

770 Dedham Street,
Canton, MA 02021
(617) 828-1209

Sixth FORML Conference
Forth Modification Laboratory

November 23–25, 1984

CALL FOR PAPERS

FORML is a technically advanced conference of Forth practitioners. The topics to be discussed will affect the future evolution of Forth. All conference participants are encouraged to write a paper for oral or poster presentation.

Topics Suggested for Presentation

Forth in 64K and beyond	**Forth on the 32 bit machine**
Forth expert systems	**Forth in the future**
Forth on the Macintosh	**Forth for robot control**
Forth advanced applications	**Forth programming style**

Registration and Papers

Complete the registration form, selecting accommodations desired, and send with your payments to FORML. Include a 100 word abstract of your proposed paper. A complete author's packet will be sent to you. The deadline for abstracts and paid registrations is October 15, 1984. Completed papers are due November 1, 1984.

About Asilomar

Asilomar is an ideal conference location. It is situated on the tip of the Monterey Peninsula overlooking the Pacific Ocean. Asilomar occupies 105 secluded acres of forest and dune. The secluded setting and clustered meeting and accommodation areas make it ideal for group meetings. Asilomar's excellent meals are complemented by Asilomar's homemade bread and pastries. Accommodations are excellent and deluxe rooms have been reserved for FORML attendees. Sweeping ocean views are available from decks or balconies. Asilomar is a Unit of the California State Park System.

Registration Form

Complete and return with check made out to:
FORML, P.O. Box 51351, Palo Alto, CA 94303

Name ______________________________

Company ______________________________

Address ______________________________

City ______________ State __________ ZIP __________

Telephone (day) ____________ (evening) ______________

I have been programming in Forth for: (years) _____ (months) _____

Accommodations Desired

Prices include coffee breaks, wine and cheese parties, use of Asilomar facilities, rooms Friday and Saturday nights, and meals from lunch Friday through lunch Sunday. Conference participants receive notebooks of papers presented.

Conference attendees, share a double room:
number of people ______ × \$250 = \$ __________

Attendees in single room (limited availability):
number of people ______ × \$300 = \$ __________

Non-conference guests:
number of people ______ × \$200 = \$ __________

Total Enclosed \$ ________

Options: Vegetarian meals? _____
Non-smoking roommate? _____

FORML, P.O. Box 51351, Palo Alto, California 94303, U.S.A.

Intermezzo
Short Continued Fraction

This interlude is a sampler. You can load one screen and see a cut-down version of the continued fraction program in action. If you like what you see, you can plunge into Part II—explanations, screens, and all.

Begin by loading figure two, the screen #8 of the file NUMBTHY. It is overstuffed and has no comments, but they will be found in abundance in Part II. Enter 31416 10000 **SCF** and execute. You will soon be viewing three columns headed **PARTIAL_QUOT**, **NUMERATOR** and **DENOMINATOR**. Ignore the first column for the moment and inspect the second and third. They list the numerator and denominator of a sequence of fractions, one or more of which you may recognize. If you enter a pair as <numerator> 10000 <denominator> **&#42;/**, execute, and interpret the result as an integer followed by four decimal places, you will surely recognize the sequence of fractions as giving approximations to a well-known number.

You can repeat with 14142 10000 **SCF** and 6931 10000 **SCF** to generate rational numbers approximating $\sqrt{2}$ and ln 2.

Return now to 31416 10000 **SCF** for a closer look. The partial quotients in the left column fit together to create continued fractions:

3	3/1
$3+\frac{1}{7}$	22/7
$3+\cfrac{1}{7+\cfrac{1}{16}}$	355/113
$3+\cfrac{1}{7+\cfrac{1}{16+\cfrac{1}{11}}}$	3927/1250

Notice also that 3927/1250 is just the fraction 31416/10000 reduced to lowest terms.

The word **SCF** accepts integers p and q no bigger than 32767 and returns the partial quotients and corresponding fractions (called convergents) of the continued fraction for p/q.

The goal of Part II is a word **CF** that would accept, say, 3141592654. and 1000000000. and return a sequence of fractions affording increasingly better approximations to the floating point number 3.141592654. From this sequence, we can select one whose numerator and denominator each fit into sixteen bits and test it using **U&#42;/** to see if the quotient approximates 3.141592654 to the desired precision.

If you see possibilities from this demonstration, continue on to Part II.

```
SCR # 8
\ NUMBTHY                                    FORTH 83   17MAR84NG
\ short version of continued fraction
VARIABLE SB_BIN  VARIABLE SQ_BIN  VARIABLE SR_BIN
: SCF                        \ N1 N2 --- continued fraction of N1/N2
  CR ." PARTIAL_QUOT"  ."   NUMERATOR"  ."   DENOMINATOR"
  1 0 2SWAP  0 1 2SWAP
  BEGIN
   SWAP OVER /MOD DUP S>D CR 12 D.R
   SB_BIN ! SR_BIN ! SQ_BIN !
   2SWAP 2OVER
   SB_BIN @ * ROT + DUP S>D 11 D.R
   ROT ROT SB_BIN @ * + DUP S>D 13 D.R
   SWAP SQ_BIN @ SR_BIN @ DUP 0=
  UNTIL
  6 0 DO DROP LOOP   ;

```

Figure Two

DASH, FIND & ASSOCIATES

Our company, DASH, FIND & ASSOCIATES, is in the business of placing FORTH Programmers in positions suited to their capabilities. We deal only with FORTH Programmers and companies using FORTH. If you would like to have your resumé included in our data base, or if you are looking for a FORTH Programmer, contact us or send your resumé to:

DASH, FIND & ASSOCIATES
808 Dalworth, Suite B
Grand Prairie TX 75050
(214) 642-5495

Committed to Excellence

Part II
Continued Fraction

If you have not read the previous Intermezzo, you may wish to do so now to get an idea of where this part will go. Then return here to continue reading.

Every real number can be approximated to arbitrary precision by rational numbers. If the real number is itself rational, then it is its own approximation to arbitrary precision, but in that case there are still approximations by rational numbers with smaller denominators. We must carefully distinguish the ideal set of real numbers from machine numbers, those entities that a given computer uses to mimic the real numbers and, even more so, machine arithmetic from the ideal of real number arithmetic. Most computer-users believe they are doing real arithmetic and, if the computer is well-designed, they will be reasonably safe in that belief.

In carrying out actual arithmetical computations, we never calculate other than with rational numbers. Irrational numbers such as $\sqrt{2}$ and π must be represented by rational surrogates. For example, we may represent π as the decimal approximation 3.1416, that is, the rational number 31416/10000. Computation with decimal fraction approximations proceeds by familiar repetitive, simple steps. However, decimal fraction representatives are not the only forms in which to present a rational number and are not always the most efficient for specific calculations.

A decimal number $n_0.n_1n_2n_3\ldots$, possibly non-terminating, is a conventional abbreviation for an infinite sum $\Sigma_{k=0}^{\infty} n_k 10^{-k}$, each n_k being an integer between 0 and 9 inclusive. Another type of representation is by a (possibly) infinite product, for example Wallis' product

$$\frac{\pi}{2} = \frac{2}{1}\frac{2}{3}\frac{4}{3}\frac{4}{5}\frac{6}{5}\frac{6}{7}\cdots$$

(I have used a product decomposition in an earlier Forth algorithm [7].) Product representations are useful in forming products of two numbers, but they are not well-adapted for addition and subtraction.

Our goal in this part is an implementation of continued fraction expansions of double precision rational numbers. Continued fractions—to be described presently—correspond to real numbers in a (nearly) one-to-one fashion, but they are not convenient for *any* of the usual arithmetic operations. They can, however, be used to produce approximating rational numbers—optimal in a sense to be described—which can be substituted for real numbers in machine computations of given precision and upon which arithmetic can be carried out as usual. If the original real number is irrational, the associated continued fraction is unique. If the original number is rational, the continued fraction is *almost* unique: there are two such associated continued fractions differing only trivially. In the rational case, the continued fraction engenders an infinite sequence of rational numbers approximating the given rational to increasing and, finally, to infinite precision. The decimal fraction form of one of these approximants may agree with that of the original to the precision required and yet the approximant may have drastically smaller numerator and denominator than the original fraction.

Continued Fractions

Here is the place to define continued fractions and describe their properties useful to us. In fact we will describe a subclass, the *simple* continued fractions. (The word "simple" is merely a technical term.) A *simple continued fraction* is a symbol

$$f = b_0 + \cfrac{1}{b_1 + \cfrac{1}{b_2 + \cfrac{1}{b_3 + \ldots}}}$$

$$= [b_0;\ b_1,\ b_2,\ b_3, \ldots].$$

The first version gives an idea of the meaning of the symbol. The second version is a conventional representation for the sake of the typographer. The first version indicates that f is the limit of a sequence of rational fractions (called convergents).

$$b_0 = b_0/1$$

$$b_0 + \frac{1}{b_1} = (b_0 * b_1 + 1)/b_0$$

$$b_0 + \cfrac{1}{b_1 + \cfrac{1}{b_2}} = \frac{b_0 * b_1 * b_2 + b_0 * b_1 + b_2}{(b_1 * b_2 + b_1)}$$

and so on. By conventions. a *partial quotient* b_k can equal 0 only if all the following partial quotients are 0. In this case the convergents stabilize and f, the limiting value, is a rational number if all the partial quotients are integers. If the sequence of partial quotients never terminates, and if the partial quotients are integers, the limiting value f always exists and it is an irrational number. (It can be shown that the sequence of integer partial quotients is eventually periodic exactly when f is a number of the form $(P + Q\sqrt{D})/R$, where P, Q, R, and D are integers, $R \neq 0$, and D is positive and not a square.) For example, the continued fraction [1; 1, 2, 3] —the trailing 0 entries are suppressed — gives the sequence of rationals 1/1, 2/1, 5/3, 17/10, and the last is the value of the continued fraction. (Note that [1; 1, 2, 3] = [1; 1, 2, 2, 1], illustrating the trivial non-uniqueness of terminating fractions.)

Proofs of the following assertions will be found in almost every book whose title contains one of the strings "Theory of Numbers" or "Number Theory." The notation is keyed to the standard reference [1, 3.10.1] with one minor change that will be pointed out later.

If $f = [b_0; b_1, b_2, b_3, \ldots]$, then the finite section $f_n = [b_0; b_1, b_2, \ldots, b_n]$ is called the *nth convergent*. When all the b_k's are integers (and this is the only case we consider from now on), the nth convergent simplifies into a rational number of the form $f_n = A_n/B_n$, where the numerator and denominator may have any common divisors removed. The limit $f = \lim A_n/B_n$ always exists. If f is irrational, the partial quotients are uniquely determined. If f is rational, they are determined except for the last: the last partial quotient can be decreased by 1 and a further partial quotient 1 appended. (For example, [1; 1, 2, 3] = [1; 1, 2, 2, 1].)

The convergents bracket the value of the continued fraction according to the inequalities

$$f_{2k} < f_{2k+2} < f < f_{2k+1} < f_{2k-1}$$

Furthermore,

$$|f - A_n/B_n| < 1/B_n^2,$$

and there is no better approximation to f by a fraction A/B with $B < B_n$.

If we set $A_{-1} = 1$, $A_{-2} = 0$, $B_{-1} = 0$, and $B_{-2} = 1$, then the value of the A_k and B_k can be found for $k \geq 0$ from the partial quotients by the recurrences

$$A_k = b_k * A_{k-1} + A_{k-2}$$

$$B_k = b_k * B_{k-1} + B_{k-2}$$

Conversely, given f, the partial quotients can be generated from the convergents by the formula

$$b_k = [B_{k-1}/(A_{k-1} - f * B_{k-1})].$$

Euclidean Algorithm

As we are aiming only for continued fractions of rational numbers, we can take an important path along the Euclidean algorithm. If u and v are integers, u non-negative and v positive, then the Euclidean algorithm consists of the sequence of divisions

$$\begin{aligned} u &= b_0 * v + r_1 \\ v &= b_1 * r_1 + r_2 \\ r_1 &= b_2 * r_2 + r_3 \\ r_2 &= b_3 * r_3 + r_4 \\ r_{n-2} &= b_{n-1} * r_{n-1} + r_n \\ r_{n-1} &= b_n * r_n \\ r_{n-1}/r_n &= b_n. \end{aligned}$$

In this presentation, the Euclidean algorithm states exactly that $u/v = [b_0; b_1, b_2, \ldots, b_n]$!

We can, therefore, implement the Euclidean algorithm and draw from it both greatest common divisor and continued fractions.

Two remarks will be useful. It should be clear from inspection of the Euclidean algorithm that it will terminate after finitely many steps. How many steps will be required? G. Lamé proved that the algorithm will require no more than about five times the larger of the numbers of decimal digits in the two terms u and v. When the terms u and v are double precision integers of at most thirty-two bits, this means that the Euclidean algorithm will cycle at most fifty times or, in other words, that the continued fraction for u/v will contain at most fifty non-zero partial quotients.

What numbers will stretch the algorithms to their limits? Further inspection of the Euclidean algorithm shows that this will happen if the numbers u and v produce the sequence of partial quotients 1; 1, 1, 1,.... When the ones run on forever to produce $f = [1; 1, 1, 1,\ldots]$, then $f = (1 + \sqrt{5})/2$, the *golden section* (or its reciprocal). Furthermore, the convergents to f can be calculated by the recurrences $A_k = A_{k-1} + A_{k-2}$, $A_{-2} = 0$, $A_{-1} = 1$, and $B_k = B_{k-1} + B_{k-2}$, $B_{-2} = 1$, $B_{-1} = 0$. Thus, both A_k and B_k run along the sequence of *Fibonacci numbers* F_k defined by the recurrence $F_k = F_{k-1} + F_{k-2}$, $F_1 = 1$, $F_2 = 1$.

Implementation of the Algorithm

The algorithm as I have implemented it calls upon the word **UD/MOD** and its satellites defined in Part I, as well as the long pair (**LP**) words. All of the calculations are in double precision and the screens are essentially the expansion to double precision of the single precision screen presented during the Intermezzo.

Again the screens have been composed in MicroMotion MasterFORTH, an implementation of Forth-83 with file system. These screens are from my file NUMBTHY. They should run in Forth-79 if the word **ABORT"** in screen #3 is adjusted. Here is a description of the screens.

Screen #2: For convenience, the greatest common divisor, calculations for which are embedded in the continued fraction algorithm, is given without encumbrances by the word **DGCD** acting on double integers. The Fibonacci numbers are useful for many purposes—including generation of demonstration continued fractions of maximal stretch—so the word **FIB** is included to list them as far as **D.** will print them.

Screen #3: The numbers A_k, B_k, and b_k all can be genuinely thirty-two-bit integers. As the recurrence formulas for generating the convergents call on products $b_k * A_{k-1}$ and $b_k * B_{k-1}$, it is conceivable that *quadruple* precision multiplication could be required. In fact, this worst case can never happen, and this is clear from a close look at the recurrence formulas. Each of the A_k, B_k and b_k is non-negative, so that $b_k * A_{k-1} < A_k \leq u$ and $b_k * B_{k-1} < B_k \leq u$. Therefore, b_k and A_{k-1} cannot be both wider than sixteen bits and the same is true of b_k and B_{k-1}: at most one of the factors can be double width. I wrote the word **CF*** to exploit this special circumstance. It calculates the product of UD1 and UD2 using Bieman's multiplication/division **U*/**. Because **U*/** is not commutative, **CF*** orders the factors and issues an error message if both are wider than sixteen bits.

Screens #4, 5, and 6: The actual calculation begins here in the mode of the Euclidean algorithm. I've written short words with long names in a try at cutting down on comment lines. The words should be easy to follow through on screen #6, where the stack at **BEGIN** holds B_{k-2}, A_{k-2}, B_{k-1}, A_{k-1}, r_{k-1}, r_k.

Screen #7: If I have chosen word names well, all that remains to explain is that the last line of **CF** cleans the stack. (I could have used the Forth-83 word **CLEAR**, but that might clear other numbers deeper in the stack.)

Using CF

The word **CF** ideally is used interactively. For example, suppose we are to calculate integer products $[x\pi]$, where x is a double integer and $[x\pi]$ is to be a double integer. The evaluation will be carried out by finding a rational u/v approximating π, then computing $[x\pi]$ by the sequence x u v **U*/**. If x is much wider than sixteen bits, it is important to approximate π in such precision that low precision digits in the product are not distorted.

We know that $\pi \cong 3.141592654$, so enter 3141592654. 1000000000. **CF.** The latest convergent both of whose elements fit into sixteen bits each is 355/113. This gives $\pi \cong 3.1415929\ldots$, good to six rounded places.

We can do better. The next entry is $104348/33215 \cong 3.141592654$, the exact rounded value to nine decimal places. But

104348 needs more than sixteen bits. Even so we can obtain [xπ] by the procedure

x 52174 33215 U*/ 2DUP D+.

Observe that the expansion given here for 355/113 = [3; 7, 15, 1], while **SCF** gives 355/113 = (3; 7, 16]. This is an instance of the only almost uniqueness of the continued fraction for rational numbers, and arises because only finitely many digits of π are being manipulated.

References

1. Abramowitz, M., and I.A. Stegun, *Handbook of Mathematical Functions,* National Bureau of Standards Applied Mathematics Series, 55. (Re-printed by Dover Publications.)
2. Anderson, Anita, and Martin Tracy, *Forth Tools,* Volume One, MicroMotion, Inc., 1984.
3. Beers, David A., "Quadruple Word Simple Arithmetic," *Forth Dimensions,* IV/1, p. 17.
4. Bieman, L.H., "Double-Precision Math Words," *Forth Dimensions,* V/1, p. 16.
5. Brodie, Leo, *Starting Forth,* Prentice-Hall, Inc., 1981
6. Corry, Robert T., "Closer Approximations," *Forth Dimensions,* IV/4, p. 4.
7. Grossman, Nathaniel, "Fixed-Point Logarithms," *Forth Dimensions,* V/5, p. 11
8. Knuth, D.E., *The Art of Computer Programming,* Volume Two, Addison-Wesley Publishing Co., 1973.

```
SCR # 2
\ NUMBTHY                                        FORTH 83  15MAR84NG
\ double number greatest common divisor and fibonacci numbers

: DGCD    \ D1 D2 --- gcd of D1 and D2
  BEGIN
   2SWAP 2OVER UDMOD 2DUP D0=
  UNTIL     2DROP D.   ;

: FIB   \ N --- first N fibonacci numbers, N < 47
  >R 0. 1. R> 1+ 1 CR
  DO
   SPACE SPACE I .
   2DUP D. SPACE 126 EMIT
   2SWAP 2OVER D+
  LOOP   ;


SCR # 3
\ NUMBTHY                                        FORTH 83  14MAR84NG
\ product of double factors when one is really single

: CF*     \ UD1 UD2 --- UD
  ?DUP                          \ is UD2 wider than 16 bits?
  IF                                \ yes, so see if UD1 is
   2SWAP ?DUP                       \  also wider than 16 bits
   IF    CR ABORT" OVERFLOW"        \ both wide => all done
   ELSE  1 U*/                  \ but at least one is narrow
   THEN                         \  permits multiplication
  ELSE  1 U*/                   \ with 32 bit product
  THEN   ;


SCR # 4
\ NUMBTHY                                        FORTH 83  15MAR84NG
\ words for continued fraction --- see scr #7

2VARIABLE B_BIN   2VARIABLE Q_BIN   2VARIABLE R_BIN

: .CF_HEADING
  CR ." PARTIAL_QUOT" ."    NUMERATOR" ." DENOMINATOR"   ;

: INIT'IZE_RECURRENCE
  1. 0. LPSWAP 0. 1. LPSWAP   ;
: GET_PARTIAL_QUOTIENT
  2SWAP 2OVER UD/MOD   ;

: .PARTIAL_QUOTIENT
  CR 2DUP 12 D.R   ;
```

```
SCR # 5
\ NUMBTHY                                      FORTH83  15MAR84NG
\ words for continued fraction --- see scr #7

: SAVE_GCD_ELEMENTS
   B_BIN 2!  R_BIN 2!  Q_BIN 2!    ;

: INIT'IZE_TO_CALC_CONVERGENT
   LPSWAP LPOVER   ;

: GET_NUMERATOR
   B_BIN 2@ CF* 2ROT D+    ;

: .NUMERATOR
   2DUP 12 D.R    ;




SCR # 6
\ NUMBTHY                                      FORTH 83 15MAR84NG
\ words for continued fraction --- see scr #7

: GET_DENOMINATOR
   2ROT 2ROT B_BIN 2@ CF* D+ ;

: .DENOMINATOR
   2DUP 12 D.R    ;

: REINIT'IZE_RECURRENCE
   2SWAP Q_BIN 2@ R_BIN 2@    ;

: ?END_GCD_ALGORITHM
   2DUP D0=    ;




SCR # 7
\ NUMBTHY                                     FORTH 83  15MAR84NG
\ continued fraction algorithm

 : CF
    .CF_HEADING   INIT'IZE_RECURRENCE
    BEGIN    \ euclidean gcd algorithm loop
     GET_PARTIAL_QUOTIENT   .PARTIAL_QUOTIENT
     SAVE_GCD_ELEMENTS      INIT'IZE_TO_CALC_CONVERGENT
     GET_NUMERATOR          .NUMERATOR
     GET_DENOMINATOR        .DENOMINATOR
     REINIT'IZE_RECURRENCE  ?END_GCD_ALGORITHM
    UNTIL
    6 0 DO 2DROP LOOP    ;
```

Re-Defining a Colon Word

E.H. Schmauch
Ponca City, Oklahoma

Although Forth offers may significant advantages over BASIC, there is one characteristic of BASIC which makes it easier for me to use in program debugging. To change any line in a BASIC program, all I do is re-enter that line and the program is ready to run. In Forth, I can interactively re-define any word; however all words which already reference this word will still reference the old version. To have the new version of the re-defined word referenced, I must re-load all words which reference the re-defined word. As programs become more complicated, the successive editing and re-loading of screens begins to resemble the old Fortran program development cycle, something I would like to avoid.

Forth purists may insist that proper planning and bottom-up development will eliminate the type of problem I am describing; however, I found myself wishing to re-define a Forth word, when higher-level words have been developed, sufficiently often to warrant the development of **RE:** and **RE;** which are shown in screens 42 and 43. By using **RE:** and **RE;** instead of : and ; I can re-define a word and previously defined words will reference the new version. This allows me to interactively debug the word. When I am satisfied with the modifications, I edit the screen and re-load everything just once, instead of once for each test run during debugging.

```
SCR #42
  ( REDEFINING A COLON WORD -- CRC=53915 -- EHS 06AUG83 FORTH-79 )

  ' : CFA @ CONSTANT DOCOL

  : RE-CREATE    ( --- )
    -FIND NOT IF
      HERE COUNT TYPE ."  NOT FOUND" ABORT
    THEN
    DROP DUP CFA @ DOCOL = NOT IF
      HERE COUNT TYPE ."  NOT A COLON WORD" ABORT
    THEN
    HERE SWAP ! ;

  : RE-EXIT    ( --- )
    R> R> 2DROP ;
  -->


SCR #43
  ( REDEFINING A COLON WORD -- CRC= 4982 -- EHS 06AUG83 FORTH-79 )

  : RE:    ( --- )
    SP@ CSP !
    CURRENT @ CONTEXT !
    RE-CREATE DOCOL , ] ;

  : RE;    ( --- )
    ?CSP
    COMPILE RE-EXIT
    [COMPILE] [ ;
    IMMEDIATE
```

RE: and **RE;** do not overwrite the old definition, but rather compile a new parameter field at the top of the dictionary. (The parameter field is the list of words in the definition.) At run time, execution is directed to the new parameter field. **RE:** is identical to : except **RE-CREATE** is used instead of **CREATE.** (For definitions of : and ; consult *All About Forth* by Glenn Haydon.) **RE-CREATE** finds the word and replaces the value in the PFA with the current top of the dictionary. **RE:** then puts the address of **DOCOL** at the top of the dictionary and enters the compile mode. Now the first word in the old definition is a colon word at the top of the dictionary. The word at the top of the dictionary will only have code and parameter fields, not name or link fields. Next the re-definition is compiled at the top of the dictionary. When the re-definition is complete, **RE;** is used instead of ;.

RE; is identical to ; except it compiles **RE-EXIT** instead of **EXIT**. **RE-EXIT** serves the same function as **EXIT** except it pops the return stack an extra time. **EXIT** would transfer program execution to the third byte of the parameter field of the original definition. The extra pop of the return stack in **RE-EXIT** causes program execution to properly return to the point after the word was called. **SMUDGE** is not used in **RE:** or **RE;** since the word being re-defined, in general, will not be on top of the dictionary.

There is a performance penalty with words re-defined with **RE:** and **RE;** since **DOCOL** must be executed twice for each execution of the re-defined word. Once I am satisfied with the definition, I will edit the screen and re-load everything, eliminating the performance penalty. **RE:** and **RE;** can also be used to easily change colon words in the Forth kernel without meta-compilation.

RE: and **RE;** demonstrates one of the greatest advantages of Forth over other languages: the ease with which a characteristic of a Forth system can be modified in Forth.

SELECTED PUBLICATIONS

The FORTH Interest Group Order Form (*on the reverse side of this page*) has 11 newly added publications selected by the FIG Publications Committee:

All About FORTH	Threaded Interpretive Languages
Beginning FORTH	Understanding FORTH
FORTH Encyclopedia	The Journal of FORTH Applications and Research, V. 1, #1
FORTH Fundamentals, Volume 1	The Journal of FORTH Applications and Research, V. 1, #2
FORTH Fundamentals, Volume 2	Dr. Dobb's Journal, 9/84

Thinking FORTH (Soft and Hard cover)

Here are brief descriptions of 4 of them:

THINKING FORTH,
A Language and Philosophy for Solving Problems
by Leo Brodie

The best-selling author of STARTING FORTH (Prentice Hall, 1981) is back again! — this time with the first guide to using FORTH to program applications. This book captures the *Philosophy* of the language to show users how to write more readable, better maintainable applications.

Both beginning and experienced programmers will gain a better understanding and mastery of such topics as:

- FORTH style and conventions
- decomposition
- factoring
- handling data
- simplifying control structures
- and more.

And, to give you an idea of how these concepts can be applied, Thinking FORTH contains revealing interviews with real-life users and with FORTH's creator, Charles H. Moore.

To program intelligently,you must first *think* intelligently, and that's where Thinking FORTH comes in.

Leo Brodie is a writer, programmer, consultant, teacher and world-renowned authority on FORTH.

BEGINNING FORTH
by Paul M. Chirlian

Here's a clear, self teaching introduction to FORTH. It starts with the very basic ideas you need to know to begin programming, then builds to the most complex FORTH programming procedures.

FORTH Fundamentals,
Volume 1: Language Usage
by C. Kevin McCabe

A complete guide to the two major versions of FORTH, fig-FORTH and FORTH-79. The book gives you nontechnical descriptions of FORTH words and programming methods, and it explores the language's internal operation and use of memory.

FORTH Fundamentals,
Volume 2: Language Glossary
by C. Kevin McCabe

Organized by core FORTH word names, this comprehensive fig-FORTH and FORTH-79 glossary gives you all the applicable vocabularies and pronounciation. Each word is fully defined, with notes on the differences between the two FORTH versions.

FORTH INTEREST GROUP
MAIL ORDER FORM

NAME ____________________

COMPANY ____________________

STREET ____________________

CITY ____________________ STATE/PROV ____________ ZIP ____________

COUNTRY ____________________ TELEPHONE (   ) ____________________

	PRICES US/FOREIGN AIR	
Membership in the FORTH Interest Group & Volume 6 of FORTH Dimensions	$15/27	______
Volume 1 FORTH Dimensions	15/18	______
Volume 2 FORTH Dimensions	15/18	______
Volume 3 FORTH Dimensions	15/18	______
Volume 4 FORTH Dimensions	15/18	______
Volume 5 FORTH Dimensions	15/18	______
BOOKS ABOUT FORTH		
All About FORTH	$25/35	______
Beginning FORTH	17/21	______
FORTH Encyclopedia	25/35	______
FORTH Fundamentals, V. 1	16/20	______
FORTH Fundamentals, V. 2	13/16	______
Starting FORTH (Soft Cover)	18/22	______
Starting FORTH (Hard Cover)	23/28	______
Thinking FORTH (Soft Cover)	16/20	______
Thinking FORTH (Hard Cover)	23/28	______
Threaded Interpretive Languages	23/28	______
Understanding FORTH	3/5	______
REFERENCE		
FORTH 83 Standard	$15/18	______
FORTH 79 Standard	15/18	______
CONFERENCE PROCEEDINGS		
FORML Proceedings 1980	$25/35	______
FORML Proceedings 1981 (2 V.)	40/55	______
FORML Proceedings 1982	25/35	______
Rochester Proceedings 1981	25/35	______
Rochester Proceedings 1982	25/35	______
Rochester Proceedings 1983	25/35	______
JOURNAL OF FORTH APPLICATIONS AND RESERACH		
Journal of FORTH Research V. 1 #1	$15/18	______
Journal of FORTH Research V. 1 #2	15/18	______
REPRINTS		
Byte Reprints	$3.50/5	______

	PRICES US/FOREIGN AIR	
Popular Computing 9/83	$3.50/5	______
Dr. Dobb's 9/81	3.50/5	______
Dr. Dobb's 9/82	3.50/5	______
Dr. Dobb's 9/83	3.50/5	______
Dr. Dobb's 9/84	3.50/5	______
HISTORICAL DOCUMENTS		
Kitt Peak Primer	$25/35	______
fig-FORTH Intallation Manual	15/18	______
ASSEMBLY LANGUAGE SOURCE LISTINGS		
1802	$15/18	______
6502	15/18	______
6800	15/18	______
6809	15/18	______
68000	15/18	______
8080	15/18	______
8086/88	15/18	______
9900	15/18	______
ALPHA MICRO	15/18	______
Apple II	15/18	______
ECLIPSE	15/18	______
IBM/PC	15/18	______
NOVA	15/18	______
PACE	15/18	______
PDP-11	15/18	______
VAX	15/18	______
Z80	15/18	______
T-Shirt Size:______	$10/12	______
Poster (BYTE Cover)	3/5	______
Handy Reference Card	FREE	______

SUBTOTAL ________

CA Residents Add 6½% Sales Tax ________

TOTAL ________

☐ VISA ☐ Mastercard # ____________________ Expiration Date ____________

$15 Minimum On VISA/Mastercard Orders.
All Prices Include Shipping.

Make Check or money order payable in US funds drawn on a US Bank to: FIG.
PAYMENT MUST ACCOMPANY ALL ORDERS (Including Purchase Orders).

OFFICE USE ONLY

By ____________ Date ____________ MO ______ TO ______ PU ______ Auth No ____________

Shipped By ____________ Date ____________ Weight ____________ UPS ______ USPS ______

Hold ____________ Date ____________ Weight ____________ UPS ______ USPS ______

ORDER PHONE: (408) 277-0668
FORTH INTEREST GROUP • P.O. BOX 8231 • SAN JOSE, CA 95155

Think Like a User

Write Like A Fox

Michael Ham
Scotts Valley, California

Good design consists in large part of anticipating the user's inclinations and accommodating them, at the same time ensuring that the user is as friendly to your program as your program is to the user. This article describes the word **DIGITS**, written to conform to this precept. **DIGITS** collects numeric data of a specified number of digits. **3 DIGITS**, for example, will allow entry of at most three numeric digits and will ignore all keys except for numerals, backspace and the enter (or return) key – and those keys allowed by **FIX**.

A computer will follow its instructions exactly but users are not so cooperative. It is better to observe users and then make your program conform to their habits and expectations than to attempt the reverse. **FIX** is a word to accommodate the users.

Whenever a user can validly complain, "The computer should have known what I meant," the design is bad. For example, typists generally use the lower-case letter L for the number one; if an L is typed in the context of numeric entry, "one" is clearly intended. The word **L->1** thus accommodates the user in this regard. Similarly, the letter O and the numeral zero not only share the same form, they also occupy adjacent keys on the keyboard. If the user is a hunt-and-peck typist, it is likely that the letter O in a numeric context is meant as a zero and **O->0** again accommodates the user.

If the program displays the numeric entry field (e.g., the numbers are being entered in an inverse video rectangle that defines the maximum number of characters), then some users will automatically space over instead of entering leading zeroes. That is, if they are to enter the number seven in a three-digit field, they normally will want to enter the seven in the units position and will press the space bar twice to move the cursor over. Rather than fight this natural tendency, it is easy to interpret spaces as zeroes. "Space, space, seven" can be accepted and displayed as "007". (Of course, if they simply enter the seven (in the hundreds place) and then hit enter, the program should accept the number as seven, not as 700.) If the entry field does not have a visibly defined length, delete **SP->0** from the definition of **FIX**.

Note that **FIX** is unobtrusive. Users who restrict themselves to the numeric keys will never know that **FIX** is present and even those who use **FIX** will be unaware of it. For them, the computer is simply doing what they would expect.

BS? and **CR?** test for backspace and carriage return. This was written using Forth Technology's Forth/level 2, which interprets the backspace key as ASCII 12. Most Forths use another value, typically 8. To find out what your Forth does, type the sequence

KEY . <enter> <backspace>

The number displayed is the key-code for your backspace key. Use it in place of "12" in the definition of **BS?**.

BACK moves the cursor back one position by decrementing the contents of **CURSOR**, which in Forth/level 2 determines the cursor position. In Forth/level 2, **EMIT**ting a control code to the screen displays a character instead of triggering the specified function. In particular, that system's **8 EMIT** (which in some Forths

Words to collect clean numeric data

```
 0   ( Numeric Input   1 of 2                  Michael Ham    6/28/84 )
 1   : BS?    ( n - f )  12 = ;
 2   : CR?    ( n - f )  13 = ;
 3
 4   : BACK    -2 CURSOR +! ;
 5   : BSP     BACK SPACE BACK ;
 6   : SP->0  ( n - n )  DUP 32 = IF DROP 48 THEN ;
 7   : L->1   ( n - n )  DUP 76 = OVER 108 = OR IF DROP 49 THEN ;
 8   : O->0   ( n - n )  DUP 79 = OVER 111 = OR IF DROP 48 THEN ;
 9   : FIX    ( n - n )  SP->0  L->1  O->0  ;
10   : OK?    ( n f - 0 or n 1 )   IF 1 ELSE BELL DROP 0 THEN ;
11   : #?     ( n - n f )  DUP 47 >  OVER 58 <  AND ;
12   : #BSCR? ( n - n f )  #?  OVER BS? OR  OVER CR? OR ;
13
14   : GET#BSCR ( - ascii )   BEGIN KEY FIX #BSCR? OK? UNTIL ;
15   : GET#BSCR ( - ascii ) 0 BEGIN DROP KEY FIX #BSCR? UNTIL ;


 0  ( Numeric Input   2 of 2                  Michael Ham    6/28/84 )
 1
 2  ( n = max # of digits to collect; m = # of digits entered)
 3  : DIGITS ( n - d m )  DUP 1+
 4     0 DO BEGIN  GET#BSCR  DUP BS?
 5     ( bksp:)     IF DROP  I IF BSP  R> 1- >R  ELSE BELL THEN  0
 6                  ELSE DUP  PAD I +  C!  DUP CR?
 7     ( cr:  )       IF DROP  1  LEAVE
 8     ( nmbr:)       ELSE OVER I =  IF DROP BELL  0  ELSE EMIT  1
 9                  THEN THEN THEN   UNTIL  LOOP  DROP ( n)
10      0 0  PAD 1-  CONVERT  PAD - ;
11
12  ( NOTE:  Delete  1  in line 7 if LEAVE is 83-Standard. )
13
14
15
```

acts as a backspace and thus could be used as the definition of **BACK**) prints a rectangular blob. Once you have defined **BACK** you can then define **BSP**, which moves the cursor back one position and erases the character above it.

#? checks to see whether the value on the stack is in the range of the ASCII values of decimal digits. **#BSCR?** extends **#?** to allow ENTER and backspace as valid characters also. I factored out **#?** for separate definition because it is useful in other contexts to check for purely numeric values.

OK? is a general-purpose word for dropping bad input with a beep, sounded by **BELL** (substitute your Forth's equivalent). A beep can be useful if the user will be doing head-down data entry. If the user is probably going to be looking at the screen, however, I prefer to ignore invalid keystrokes without the beep: unnecessary noise is distracting in an office. I thus give two versions of **GET#BSCR**; the second version, on line fifteen of the first screen listing, will not beep. Instead, the **DROP** immediately after the **BEGIN** will get rid of invalid input after the **UNTIL** has eaten the flag left by **#BSCR?**. The zero preceding **BEGIN** is to give **DROP** something to drop the first time through the loop.

You pick the version of **GET#BSCR** that you want to use. Both will loop until the user enters an acceptable value. From the point of view of the user, invalid keys (for example, most of the alphabetic keys) simply don't work.

The above words are used to define **DIGITS**, which itself requires a number: the maximum number of digits to be allowed in the number the user will enter. **DIGITS** leaves *two* numbers on the stack: the count of digits the user actually typed and, beneath that, the entered number (as a double-precision number occupying two cells on the stack). By having the count of digits typed, your program can check whether a number was input or whether the enter key was pressed immediately, with no digits entered.

The trick in writing **DIGITS** was seeing that to collect a five-digit number, for example, the loop must allow for *six* repetitions – but the sixth time through the loop, enter and backspace are the only valid input.

DIGITS first duplicates the number of digits allowed. It uses one copy to determine when it is in the last cycle of the loop; it adds one to the other copy and uses that as the limit for the loop. **DIGITS** refuses to backspace on the first cycle of the loop and refuses numeric input on the last cycle (when only enter or backspace are allowed). For these two errors, I decided that a beep was warranted because these keys are normally acceptable; it is only under these special circumstances that they are invalid. If you don't want the beep, eliminate the word **BELL**.

The **DO** loop in **DIGITS** contains a **BEGIN-UNTIL** construct – the routine escapes the **BEGIN-UNTIL** only when either a valid digit is collected or enter is pressed.

If **GET#BSCR** delivers a backspace, the value is dropped from the stack; then if the backspace is invalid (trying to backspace past the beginning of the number) a beep is sounded or, if the backspace is valid, the backspace is performed and the loop index is picked up, decremented by one and put back. By setting back the index by one, the routine will collect again the previous digit. In either case, the backspace leaves a zero on the stack so that the routine will not escape the **BEGIN-UNTIL** but will go back to **GET#BSCR**.

If the input was *not* a backspace, it is stored in the scratch area **PAD**. The index is used to locate the proper byte within **PAD** so that the ASCII string representation of the number is built digit by digit. Note that the ASCII value for enter (13) will be stored as the last character of the string.

If **GET#BSCR** delivers an ASCII 13 (the enter key), that value is stored in the string, then dropped from the stack and the **DO** loop is exited with **LEAVE**.

If **GET#BSCR** delivers the ASCII value for a number – the next digit of the entered number – a check is made to see whether this is the last time through the loop (when only a backspace or enter will be accepted). If so, the digit's ASCII value is dropped from the stack, the computer beeps and zero is left on the stack so that the routine will repeat the **BEGIN-UNTIL** loop; if not, **EMIT** displays the digit on the screen and a one is left on the stack to escape the **BEGIN-UNTIL** and go through the next cycle of the **DO** loop.

Extraneous digits – digits that were backspaced over or entered (incorrectly) on the last cycle through the **DO** loop —are left in the string in **PAD**. They can be left there because they will be overlaid either by the correct digit or by the ASCII 13 stored when enter is pressed. The ASCII 13 marks the end of the valid digits.

When the routine escapes the **DO** loop, it drops the number that was kept on the stack to check the index. **CONVERT** converts a number expressed as an ASCII string into numeric representation, adding it to a double-precision number on the stack. The two zeroes (which amount to one double-precision zero) are put on the stack so that **CONVERT** will be adding to zero, and **PAD 1-** provides the correct address for **CONVERT**. **CONVERT** leaves on top of the stack the address of the first non-numeric character it encountered in the string (which will be the ASCII 13 from the enter). By subtracting **PAD** (the address of the beginning of the string), the top of the stack will show the number of digits entered.

If fewer than five digits are collected, the number will be a single-precision number. **DIGITS** can then be followed by **2DROP** to get a single-precision number – without showing the number of digits collected. If you want a single-precision result but also want to know the number of digits entered (so you can distinguish an entered zero from no entry), follow **DIGITS** with **SWAP DROP**.

Once you have received the result of **DIGITS**, you can subject it to any other edits your application might demand – maximum or minimum legal values or the like. **DIGITS** cooperates with the user, but also delivers a clean number for you to work with.

SUPER FORTH 64®

By Elliot B. Schneider

TOTAL CONTROL OVER YOUR COMMODORE-64™
USING ONLY *WORDS*

MAKING PROGRAMMING FAST, FUN AND EASY!

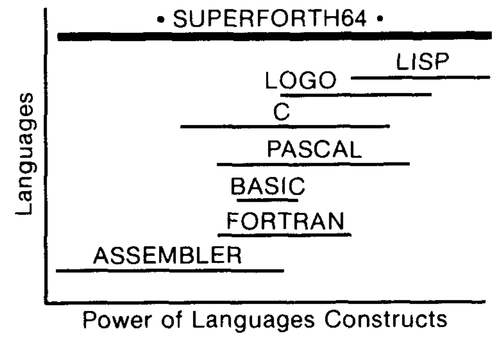
MORE THAN JUST A LANGUAGE...
A complete, fully-integrated program development system.

Home Use, Fast Games, Graphics, Data Acquisition, Business, Music
Real Time Process Control, Communications, Robotics, Scientific, Artificial Intelligence

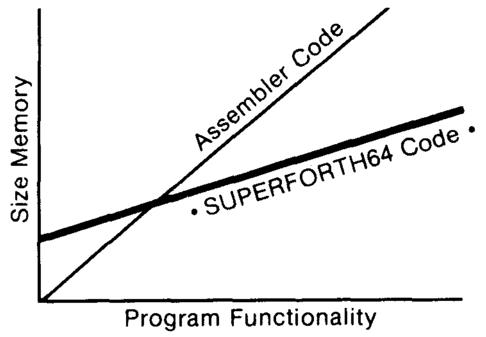
A Powerful Superset of MVPFORTH/FORTH 79 + Ext. for the beginner or professional

- 20 to 600 x faster than Basic
- 1/4 x the programming time
- Easy full control of all sound, hi res. graphics, color, sprite, plotting line & circle
- Controllable SPLIT-SCREEN Display
- Includes interactive interpreter & compiler
- Forth virtual memory
- Full cursor Screen Editor
- Provision for application program distribution without licensing
- FORTH equivalent Kernal Routines
- Conditional Macro Assembler
- Meets all Forth 79 standards+
- Source screens provided
- Compatible with the book "Starting Forth" by Leo Brodie
- Access to all I/O ports RS232, IEEE, including memory & interrupts
- ROMABLE code generator
- MUSIC-EDITOR
- SPRITE-EDITOR
- Access all C-64 peripherals including 4040 drive and EPROM Programmer.
- Single disk drive backup utility
- Disk & Cassette based. Disk included
- Full disk usage — 680 Sectors
- Supports all Commodore file types and Forth Virtual disk
- Access to 20K RAM underneath ROM areas
- Vectored kernal words
- TRACE facility
- DECOMPILER facility
- Full String Handling
- ASCII error messages
- FLOATING POINT MATH SIN/COS & SQRT
- Conversational user defined Commands
- Tutorial examples provided, in extensive manual
- INTERRUPT routines provide easy control of hardware timers, alarms and devices
- USER Support

SUPER FORTH 64® *is more powerful than most other computer languages!*



SUPER FORTH 64® *compiled code becomes more compact than even assembly code!*



A SUPERIOR PRODUCT in every way! At a low price of only

$96

Free Shipping in U.S.A.

CALL:
(415) 961-4103
MOUNTAIN VIEW PRESS INC.
P.O. BOX 4656, MT. VIEW, CA 94040
Dealer for
PARSEC RESEARCH
Drawer 1776, Fremont, CA 94538
AUTHOR INQUIRIES INVITED

Ordering Information: Check, Money Order (payable to MOUNTAIN VIEW PRESS, INC.), VISA, MasterCard, American Express. COD's $5.00 extra. No billing or unpaid PO's. California residents add sales tax. Shipping costs in US included in price. Foreign orders, pay in US funds on US bank, include for handling and shipping $10.

® PARSEC RESEARCH (Established 1976)

Commodore 64 & VIC-20 TM of Commodore

Upgrading Forth-79 Programs

Robert Berkey
Palo Alto, California

Given a set of code developed under the 79-Standard, several choices exist concerning translating to Forth-83. The approach considered in this article is to integrate the application with the new standard, and the changes needed are reviewed. After upgrading, each word used within the application will then be used in a Forth-83 manner.

The discussion is based on having a 79-Standard program needing translation onto a Forth-83 system. Actual programs may use system-dependent extensions but the standard word set provides a basis from which to work. A general knowledge of the changes to Forth-79 is expected for following the discussion; these changes have been documented in previous issues of *Forth Dimensions*. Only the required word set of Forth-79 is considered, not the extension word sets. Users upgrading fig-FORTH applications will also need to consider the differences between fig-FORTH and Forth-79.

Integrating the Code

The largest programming task is to examine the output of each flag (eight exist in the standard, and a typical application uses additional flag producers), and determine whether its output is used in other than a zero or non-zero way. If in doubt, **NEGATE** or **ABS** can be inserted in-line following the flag result. **ABS** will work with either a 79flag or an 83flag, whereas **NEGATE** will be slightly faster; but don't ignore efficient alternatives. The phrase **79flag** + can be re-coded **83flag** -. **79flag** * is the same as **83flag AND**. The phrase

```
flag IF + ELSE DROP THEN
```

is the same as

```
83flag AND +
```

NOT must be examined to see if its input is a pure flag. If in doubt, it should be changed to **0=**. Indeed, it is simplest to globally change each **NOT** to **0=**, although I find this to be stylistically undesirable. Uses of **LEAVE** should be examined for code that would have executed following the **LEAVE** and before the loop terminates; such code includes **+LOOP** itself, since **+LOOP** removes a value from the stack. Note that the only word needed after the 83-Standard **LEAVE** is **THEN**. For example,

```
IF (a) LEAVE ELSE (b) THEN
```

can be re-coded as

```
IF (a) LEAVE THEN (b)
```

Wherever ' is used, the ticked word must be followed with >**BODY**. Cases of ' used in a colon definition must be changed to **[']** and, as just mentioned, >**BODY** must be added after the ticked word.

My own experience in doing a major conversion was to successfully examine all the flags, **NOT**s, tick-values and **LEAVE**s and to forget to convert the few cases involving **PICK** and **ROLL**. Some modifications are less common but can be routinely checked for. **U*** must be re-named **UM*** and **U/MOD** must be re-named **UM/MOD**. **FIND** translates directly by replacing its occurrences in colon definitions with the following phrase:

```
32 WORD FIND  0= IF  DROP 0  THEN
```

If **FIND** is being interpreted on a load screen it can usually be replaced with ' (tick).

Several words have different immediate flags. **COMPILE LEAVE** gets changed to **[COMPILE] LEAVE**. **FORTH**, when used in a colon definition, must be replaced with **[FORTH]**. Likewise, **EDITOR** is no longer immediate and should be replaced with **[EDITOR]**. The phrases **[COMPILE] FORTH** and **[COMPILE] EDITOR** should have the **[COMPILE]** removed for stylistic reasons.

The formerly state-smart words present special problems when preceded with **[COMPILE]**. Consider the phrase **[COMPILE]** '. Will the word containing the **[COMPILE]** ' be used (1) when **STATE** is true, (2) when **STATE** is false, or (3) both? For case (1) use **[COMPILE] [']**. For case (2) remove the **[COMPILE]**. For any of the three cases a state-smart phrase can be substituted:

```
STATE @ IF  [COMPILE] [']  ELSE  '
  THEN  [COMPILE] ."
```

can have the following phrase substituted:

```
STATE @ IF [COMPILE] ." ELSE
  34 WORD COUNT TYPE THEN
```

[COMPILE] LITERAL can have the following phrase substituted:

```
STATE @ IF [COMPILE] LITERAL THEN
```

The considerations from here until the end of the paper will become increasingly unusual, but any bug in an algorithm is too much of a bug, so here goes. This material is also of increasing interest to work intended to be transportable across all standard systems.

A number of other changes affect few programs. These include division producing negative quotients when the remainder is not zero, unusual **DO** inputs, the trailing character left by **WORD** and trailing null characters left by **EXPECT**. None of these was a problem in the application I converted.

SCR, **LIST**and **EMPTY-BUFFERS** are no longer required by the standard but this is unlikely to be a problem, both because standard programs don't typically use these words and because standard systems have them around anyway. If they are unavailable in the **FORTH** vocabulary check in the **EDITOR** vocabulary. If available in the **FORTH** vocabulary they can be depended upon to conform with the Controlled Reference Words of Forth-83.

QUERY is no longer required but is typically implemented. It is regulated by the Controlled Reference Word Set but it should be noted that the Forth-83 **QUERY** sets >**IN** and **BLK** to zero. The word **?** is no longer supported but is easily defined:

```
: ? ( a -- ) @ . ;
```

Block buffers may be a subtle problem because in Forth-83 changes can be written to disk even if not **UPDATE**d although I don't know of any systems that exhibit this property. The phrase

```
SAVE-BUFFERS EMPTY-BUFFERS
```

can be re-coded **FLUSH**. Forth-79 algorithms that make other use of **EMPTY-BUFFERS** or that write to block buffers without updating are candidates for re-designing.

Some usages that are now non-standard will nonetheless work on some implementations; included in this group are (1) ticking a **CONSTANT** and modifying its contents, (2) using **COMPILE** in conjunction with , (comma) to put a literal value into the threaded code as with

```
COMPILE [ 0 , ]
```

and (3) using the output of **KEY** without stripping the high bits.

If the application is to be portable, each of these cases should be reviewed and eliminated. For the case of ticking a **CONSTANT** and modifying the contents, it may be useful to define a new class of words:

```
: VALUE  ( -- 16b ) ( creating: 16b --)
   CREATE , DOES> @ ;
```

If any **CONSTANT** which is being ticked is re-defined using **VALUE**, the problem is resolved. A similar solution exists for **KEY**.

```
: ASKEY  ( -- char )  KEY 127 AND ;
```

This, of course, requires altering references to **KEY** to become **ASKEY**. If your application is dependent on structures of the class

```
COMPILE [ 0 , ]
```

no standard mechanism is available for translation. The significance here is that the threaded code of a colon definition may be kept outside of the Forth address space. In practice there will be alternate programming approaches available or, for any particular system, a simple system-dependent word (or words) providing equivalent functions.

Multi-programming has been specified in Forth-83. If the application will run on only one system and that system is not multi-tasking, then there is no problem. But if the application is to be portable, multi-programming must be considered. A likely problem area is typing from block buffers.

Yet another problem can exist if the requirements of the standard program exceed the capacity of the standard system. Systems can be considered standard with as few as 2000 bytes of application dictionary, sixty-four bytes of data stack, forty-eight bytes of return stack and thirty-two mass storage blocks.

Another area to be considered is non-standard practices that worked on the older system but that will not run on a Forth-83 system. One potential problem is confusing **I** and **R@**. Similarly, the function **I'** might have been used to extract the loop limit. For many Forth-83 systems the loop limit can be retrieved as a function of the top two elements of the return stack, but the specific function varies widely among systems. Additional unknown effects will involve vocabularies. **CONTEXT** and **CURRENT** are no longer in the required word set; in Forth-83 they are in the System Extension Word Set. In fig-FORTH, **CONTEXT** @ could be used to preserve the entire search order. In Forth-83 systems with a vocabulary stack, **CONTEXT** @ may preserve only the first vocabulary in the search order. **FORGET** now uses the compilation vocabulary, not the **CONTEXT** search-order.

The standard permits loop stacks, and the loop stack need support only three concurrent loops in the program. The number of loops in use at any one time is not something a Forth programmer would normally consider.

Some non-standard techniques will work only on an indirect threaded code system, while others will work only on a subset of indirect threaded code systems; in either case these techniques would not have worked on certain Forth-79 systems and won't work on some Forth-83 systems. Here is an example of problems involving non-standard techniques. Consider the following definitions:

```
: 79DEFINER CREATE COMPILE
   [ HERE 6 - @ , ]
  FIND ,  FIND ,  COMPILE EXIT
  DOES> EXECUTE  8 * ;
```

A superficially Forth-83 implementation of the same idea follows:

```
: NEST (  apf -- ) 2+ >R ;
: 83DEFINER  CREATE  ['] NEST @ ,
   ' , ' ,  COMPILE EXIT
  DOES>  NEST  8 * ;
```

These are used in the form:

```
79DEFINER MUNCH  PAD @
83DEFINER MUNCH  PAD @
```

On a post-increment indirect threaded system, **MUNCH** will fetch the value at **PAD** and multiply it by eight.

But the above definitions comprise a rogues' gallery of non-standard practices. The definitions use the following Forth-83 non-standard practices.

- Reaching into the header of the Forth definition and knowing what is there.
- Requiring that the code field of a colon definition be exactly two bytes. (With a direct threaded system the code field might well be three, four or more bytes.)
- Leaving the return stack unbalanced in a colon definition.
- Assuming knowledge of whether the system has a pre-increment or a post-increment IP.
- Assuming that the threaded code is kept in the Forth address space, i.e., assuming that execution of **COMPILE** will modify the value of **HERE** by two (**HERE** might in fact be modified by zero, one, two, three or four).

• Assuming that the compilation address is a Forth address. (A compilation address might be any sixteen-bit value that is somehow contained in the threaded code and understood by the address interpreter. In addition, the compilation address is not necessarily the address of a code field.)
• Assuming that the sixteen-bit compilation address is the only component of the threaded code. (A jump-subroutine system would have an additional one or two bytes; a byte-token system would only have one byte.)
• Assuming that the address used by the address interpreter is a Forth byte address. (It could be a cell address or another number, such as an 8086 segment.)
• Assuming that the address used by the address interpreter is kept on the return stack and is only one cell deep. (The part of the return stack accessible with **R>** may well have a return address of zero, one or two cells in length.)
• Assuming knowledge of the physical relationship of the address of the code field and the address of the parameter field. (**>BODY** converts the compilation address to the address of the parameter field for words whose parameter fields are known to exist in the Forth address-space — **CREATE** and **VARIABLE** and user-defined words that execute **CREATE** — but no mechanism is standardized for going from the parameter field to the compilation address or from the parameter field to the address of the code field.)

Because of the actual structure involved, these definitions could be standardized in this way:

```
: EXECUTE2  ( apf --- )  DUP >R  @
EXECUTE  R> 2+ @ EXECUTE ;
: 83DEFINER  CREATE  ' ,  ' ,
DOES>  EXECUTE2  8 * ;
```

This article has shown one approach to translating a Forth-79 program onto a Forth-83 system. The customized changes described above produce efficient and compact code. The next issue of *Forth Dimensions* presents another approach, a program interface that minimizes man-hours required to translate at the cost of efficiency and size of the program.

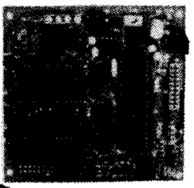
YOU WON'T WANT THIS BOARD BECAUSE:

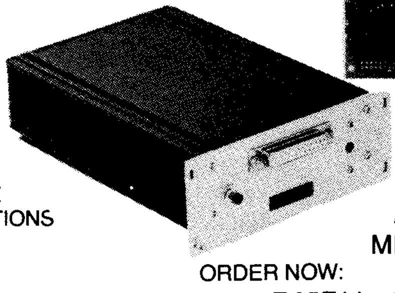
It is a low cost development system; it comes in a metal case; it has its own 9 VAC power pak; it has a high level language and operating system built in, that software-supports disk functions; it has an RS232 serial port; it has two 8 bit parallel ports (five 8 bit ports for $330); it has two 16 bit multi-function counter/timers; it has edge sensitive lines; it has a 2 kbyte CMOS RAM; it has a 2 kbyte EEPROM; it has in circuit PROM programming capability; it has a prototyping area and room in the box for other cards.
It is reliable . . . it is rugged . . . it is powerful.

NOT EVEN BECAUSE: it can auto-start a user program on power up and is priced at $90 in OEM quantity and configuration.

YOU WANT IT BECAUSE:
IT WILL SAVE YOU TIME!

Dedicated applications can quickly be developed and installed . . . like . . .
The electronic to compass-to-computer interface: development time - 4 hours.
Or the acoustic ranging unit with false echo rejection: 5 days.
Or the electronic scale-to-business system, RS232 link: under 2 hours.





THE
"100 Squared"™
BASED ON THE
R65F11
FORTH-BASED
MICROCOMPUTER

COST EFFECTIVE ENOUGH TO KEEP IN STOCK FOR THOSE
QUICK INTERFACING FIXES OR DEDICATED CONTROL APPLICATIONS

808 DALWORTH
SUITE A
GRAND PRAIRIE, TEXAS 75050
214/642-5494

ORDER NOW:

R65F11	$290
R65F12	$330

THE FORTH SOURCE™

MVP-FORTH

Stable - Transportable - Public Domain - Tools
You need two primary features in a software development package . . . a stable operating system and the ability to move programs easily and quickly to a variety of computers. MVP-FORTH gives you both these features and many extras. This public domain product includes an editor, FORTH assembler, tools, utilities and the vocabulary for the best selling book "Starting FORTH". The Programmer's Kit provides a complete FORTH for a number of computers. Other MVP-FORTH products will simplify the development of your applications.

MVP Books - A Series

- ☐ **Volume 1,** *All about FORTH* by Haydon. MVP-FORTH glossary with cross references to fig-FORTH, *Starting FORTH* and FORTH-79 Standard. 2nd Ed. $25
- ☐ **Volume 2,** *MVP-FORTH Assembly Source Code.* Includes CP/M®, IBM-PC®, and APPLE® listing for kernel $20
- ☐ **Volume 3,** *Floating Point Glossary* by Springer $10
- ☐ **Volume 4,** *Expert System* with source code by Park $25
- ☐ **Volume 5,** *File Management System* with interrupt security by Moreton $25

MVP-FORTH Software - A Transportable FORTH

- ☐ **MVP-FORTH Programmer's Kit** including disk, documentation, Volumes 1 & 2 of MVP-FORTH Series (*All About FORTH*, 2nd Ed. & Assembly Source Code), and *Starting FORTH*. Specify ☐ CP/M, ☐ CP/M 86, ☐ CP/M+, ☐ APPLE, ☐ IBM PC, ☐ MS-DOS, ☐ Osborne, ☐ Kaypro, ☐ H89/Z89, ☐ Z100, ☐ TI-PC, ☐ MicroDecisions, ☐ Northstar, NEW ☐ Compupro, ☐ Cromenco, ☐ DEC Rainbow, ☐ NEC 8201, ☐ TRS-80/100 $150
- ☐ **MVP-FORTH Enhancement Package** for IBM-PC/XT Programmer's Kit. Includes full screen editor, NEW MS-DOS file interface, disk, display and assembler operators. $110
- ☐ **MVP-FORTH Cross Compiler** for CP/M Programmer's Kit. Generates headerless code for ROM or target CPU $300
- ☐ **MVP-FORTH Meta Compiler** for CP/M Programmer's kit. Use for applicatons on CP/M based computer. Includes public domain source $150
- ☐ **MVP-FORTH Fast Floating Point** Includes 9511 math chip on board with disks, documentation and enhanced virtual MVP-FORTH for Apple II, II+, and IIe. $450
- ☐ **MVP-FORTH Programming Aids** for CP/M, IBM or APPLE Programmer's Kit. Extremely useful tool for decompiling, callfinding, and translating. $200
- ☐ **MVP-FORTH PADS (Professional Application Development System)** for IBM PC, XT or PCjr or Apple II, II+ or IIe. An integrated system for customizing your FORTH programs and applications. The editor includes a bi-directional string search and is a word processor specially designed for fast development. PADS has almost triple the compile speed of most FORTH's and provides fast debugging techniques. Minimum size target systems are easy with or without heads. Virtual overlays can be compiled in object code. PADS is a true professional development system. Specify Computer. $500
- NEW ☐ **MVP-FORTH Floating Point & Matrix Math** for IBM with 8087 or Apple with Applesoft on Programmer's Kit or PADS. $85
- NEW ☐ **MVP-FORTH Graphics Extension** for IBM or Apple on Programmer's Kit or PADS. $65
- NEW ☐ **MVP-FORTH MS-DOS** file interface for IBM PC PADS $80
- NEW ☐ **MVP-FORTH Expert System** for development of knowledge-based programs for Apple, IBM, or CP/M. $100

FORTH CROSS COMPILERS Allow extending, modifying and compiling for speed and memory savings, can also produce ROMable code. Specify CP/M, 8086, 68000, IBM, Z80, or Apple II, II+ $300

Ordering Information: Check, Money Order (payable to MOUNTAIN VIEW PRESS, INC.), VISA, MasterCard, American Express. COD's $5 extra. Minimum order $15. No billing or unpaid PO's. California residents add sales tax. Shipping costs in US included in price. Foreign orders, pay in US funds on US bank, include for handling and shipping by Air: $5 for each item under $25, $10 for each item between $25 and $99 and $20 for each item over $100. All prices and products subject to change or withdrawal without notice. Single system and/or single user license agreement required on some products.

FORTH DISKS

FORTH with editor, assembler, and manual.

- NEW ☐ **APPLE** by MM, 83 $100
- ☐ **ATARI®** valFORTH $60
- NEW ☐ **CP/M®** by MM, 83 $100
- ☐ **HP-85** by Lange $90
- ☐ **HP-75** by Cassady $150
- NEW ☐ **IBM-PC®** by LM, 83 $100
- ☐ **NOVA** by CCI 8" DS/DD $175
- NEW ☐ **Z80** by LM, 83 $100
- NEW ☐ **8086/88** by LM, 83 $100
- NEW ☐ **68000** by LM, 83 $250
- ☐ **VIC FORTH** by HES, VIC20 cartridge $50
- ☐ **C64** by HES Commodore 64 cartridge $40
- ☐ **Timex** by HW $25

Enhanced FORTH with: F-Floating Point, G-Graphics, T-Tutorial, S-Stand Alone, M-Math Chip Support, MT-Multi-Tasking, X-Other Extras, 79-FORTH-79, 83-FORTH-83.

- NEW ☐ **APPLE** by MM, F, G, & 83 $180
- ☐ **ATARI** by PNS, F,G, & X. $90
- NEW ☐ **CP/M** by MM, F & 83 $140
- ☐ **Multi-Tasking FORTH** by SL, CP/M, X & 79 $395
- ☐ **TRS-80/I or III** by MMS F, X, & 79 $130
- ☐ **Timex** by FD, tape G,X, & 79 $45
- NEW ☐ **C64** by ParSec. MVP, F, 79, G & X $96
- ☐ **Victor 9000** by DE,G,X $150
- ☐ **FDOS** for Atari FORTH's $40
- ☐ **Extensions** for LM Specify IBM, Z80, or 8086
  - ☐ Software Floating Point $100
  - ☐ 8087 Support (IBM-PC or 8086) $100
  - ☐ 9511 Support (Z80 or 8086) $100
  - ☐ Color Graphics (IBM-PC) $100
  - ☐ Data Base Management $200
- ☐ **fig-FORTH Programming Aids** for decompiling, callfinding, debugging and translating. CP/M, IBM-PC, Z80 or Apple. $200

FORTH MANUALS, GUIDES & DOCUMENTS

- ☐ **ALL ABOUT FORTH** by Haydon. See above. $25
- ☐ **FORTH Encyclopedia** by Derick & Baker $25
- NEW ☐ **The Complete FORTH** by Winfield $16
- ☐ **Understanding FORTH** by Reymann $3
- ☐ **FORTH Fundamentals,** Vol. I by McCabe $16
- ☐ **FORTH Fundamentals,** Vol. II by McCabe $13
- NEW ☐ **FORTH Tools, Vol.1** by Anderson & Tracy $20
- ☐ **Beginning FORTH** by Chirlian $17
- ☐ **FORTH Encyclopedia Pocket Guide** $7
- ☐ **And So FORTH** by Huang. A college level text. $25
- ☐ **FORTH Programming** by Scanlon $17
- ☐ **FORTH on the ATARI** by E. Floegel $8
- ☐ **Starting FORTH** by Brodie. Best instructional manual available. (soft cover) $18
- ☐ **Starting FORTH** (hard cover) $23
- ☐ **68000** fig-Forth with assembler $25
- ☐ **Jupiter** ACE Manual by Vickers $15
- ☐ **1980 FORML Proc.** $25
- ☐ **1981 FORML Proc** 2 Vol $40
- ☐ **1982 FORML Proc.** $25
- ☐ **1981 Rochester FORTH Proc.** $25
- ☐ **1982 Rochester FORTH Proc.** $25
- ☐ **1983 Rochester FORTH Proc.** $25
- ☐ **A Bibliography of FORTH References, 1st Ed.** $15
- **The Journal of FORTH Application & Research**
  - ☐ **Vol. 1, No. 1** $15
  - ☐ **Vol. 1, No. 2** $15
- ☐ **METAFORTH** by Cassady $30
- ☐ **Threaded Interpretive Languages** $23
- ☐ **Systems Guide to fig-FORTH** $25
- ☐ **Invitation to FORTH** $20
- ☐ **PDP-11 User Man.** $20
- ☐ **FORTH-83 Standard** $15
- ☐ **FORTH-79 Standard** $15
- ☐ **FORTH-79 Standard Conversion** $10
- ☐ **Tiny Pascal fig-FORTH** $10
- ☐ **NOVA fig-FORTH** by CCI Source Listing $25
- ☐ **NOVA** by CCI User's Manual $25
- ☐ **Installation Manual for fig-FORTH,** $15

Source Listings of fig-FORTH, for specific CPU's and computers. The Installation Manual is required for implementation. Each $15

☐ 1802	☐ 6502	☐ 6800	☐ AlphaMicro ☐ IBM NEW
☐ 8080	☐ 8086/88	☐ 9900	☐ APPLE II
☐ PACE	☐ 6809	☐ NOVA	☐ PDP-11/LSI-11
☐ 68000	☐ Eclipse	☐ VAX	☐ Z80

8/84

MOUNTAIN VIEW PRESS, INC.

PO BOX 4656 MOUNTAIN VIEW, CA 94040 (415) 961-4103

Astronomical Problems

William F. Ragsdale
Hayward, California

"Ask the Doctor" is Forth Dimensions' *health maintenance organization devoted to helping you use and understand Forth. Questions about problems you have, references you need or contemporary techniques are most appropriate. When needed, our columnist will call in specialists. Published letters will receive a pre-print of the column as a direct reply.*

Walter Milton of Camden, New Jersey has just begun to use the popular Forth from Texas Instruments on his TI 99/4A. He offers a request that the good doctor hopes will stir thoughts for new application programs in the mind of at least one programmer looking for a challenge.

"I am an amateur astronomer and would appreciate any help or direction you can supply. **I would like to write programs to do the following: 1) Code the celestial mechanics equations for planetary motion within the solar system. 2) Generate plots specifically for Jupiter and the four bright (Galilean) satellites. 3) I would also like to input my observations of Halley's comet and compare with others' reports.**"

Rx: Forth had early use in the astronomical community due to Charles Moore's association with the National Radio Astronomical Observatory and the Kitt Peak National Observatory. His work aided research there, at Owens Valley Radio Observatory and at other observatories world wide. One can surmise that astronomers were limited by funds and hardware, and depended on Forth to extend their resources.

Two proponents of Forth are Roger Stapleton at St. Andrews and Hans Nieuwenhuijzen at the State University at Utrecht. At this moment, I am being tantalized by a listing from St. Andrews for the measurement titled "Lunar Occultation of Aldebaran." This is exchange material from 1979, sent by Roger. Since the application is quite specific to his hardware and support software, it serves to inspire rather than to provide specific guidance. I trust that both of these educators could provide a short list of references on the appropriate calculation methods and practical approaches.

While not programmed in Forth, the program TellStar (from Information Unlimited Software of Sausalito, California) will produce customized viewing charts for your site and time from an internal data base and planetary calculations. The product also presents the constellations of the Southern Hemisphere, so you may explore, in simulation, portions of the heavens only available otherwise by extended travel.

Who would like to combine the elements of Julian date, star coordinates and planetary motion, yielding a Forth simulation system?

The gentlemen mentioned above may be contacted at:

Dr. J. Roger Stapleton
University Observatory
Buchanan Gardens
St. Andrews, Fife
Scotland

Dr. Hans Nieuwenhuijzen
Sterrewacht Sonnenborgh
Zonnenburg 2
3512 NL Utrecht
The Netherlands

Frans Van Duinen of Toronto, Canada, graciously writes: "Firstly, let me express my appreciation for all the good work you and the many other Figgers have done and are still doing. Thanks! **In the original fig-FORTH Model (v1.1), as in the Forth-83 Standard, the colon changes the first vocabulary in the search order to the one receiving the new definitions (CURRENT).** Why is that? There are good reasons, no doubt, but for the life of me I see none, only a disadvantage. When I set the search order to **CP/M FORTH DBASE ONLY** I don't appreciate the system changing it to **DBASE FORTH DBASE ONLY** just because I happen to be adding definitions to **DBASE**. Your comments, please."

Rx: A small number of reasons come to mind for this characteristic. I'm unsure if they are compelling. A broader discussion could bring to light additional aspects, favorable or not. The short answer is that it's always been that way. The contemporary practice is reflected in Forth, Inc. products, fig-FORTH, and the Forth-78, Forth-79 and Forth-83 Standards.

A more satisfying reason is that during testing you may switch to another vocabulary; when resuming compilation, you might now compile identically named words from that other vocabulary. If colon (:) did not move back to your application vocabulary, you would compile the editor **I**, which would be quite inappropriate inside a **DO...LOOP**.

Forth program development is highly interactive. You move from testing to editing to compiling in a matter of seconds. Many would find it irritating to have to type in the application vocabulary name at the end of editing. You presently just have to load the next application block and the editor is replaced by the application vocabulary.

The restoration of the **CURRENT** vocabulary was much more important back in the days when the search order was specified in the order in which vocabularies were defined. (This is compile-time chaining as contrasted to the contemporary method of run-time chaining with **ONLY** and **ALSO**.) Since the editor chained only to **FORTH** in those systems, without the automatic switch you would search **EDITOR**, then **FORTH** and then the **CURRENT** vocabulary. Since your application would be searched last, any synonyms in **EDITOR** or in **FORTH** would be found first.

We see in compile-time-chained systems that the context shift at the start of compiling is needed for uniform behavior. The importance is reduced, but not eliminated, in run-time-chained systems.

Forth is a very fluid environment. Vended systems can be extended or modified. Ross Grable is concerned about validation. He writes, "Dear Colleague: **Is there any software that can verify the operation of a version of Forth? I am looking for a program that will verify a system against a glossary or a standard.** It seems that this would be an extrememly useful tool, particularly since cross-computer portability is a strong feature of the language. Testing Forth automatically would certainly raise some interesting questions, such as how to test the limitations of compilation and defining words."

Rx: In early 1982 a team of four system implementors formed in Northern California to develop a validation package for Forth-79. By the time they were under way, the prospect of Forth-83 suggested that their work was premature. However, they didn't resume their effort upon the release of Forth-83.

The field appears to be wide open, even invited by the standard itself. Section 7.2 of Forth-83 gives the required hardware configuration for testing (2000 bytes of memory, thirty-two disk blocks, stacks of sixty-four and forty-eight bytes, and an ASCII terminal). Some of the steps likely to be in a validation suit would be to inventory all words, classify if immediate or non-immediate, verify that the validation source code has not been altered and check characteristics of the words. Typical subtleties would include tests that **IF...THEN** branches can branch further than 127 bytes, and that **CREATE DOES>** words are of the specified construction. **FIND** classifies words as immediate, non-immediate or missing. Prospects of testing prompted this Standards Team choice. Such a validation program could be the basis of a product, or at least a very lively conference paper.

David Fu of San Diego, California, has set his sights at the high end of the hardware spectrum. He queries, **"I want to know whether anybody put Forth on the CDC Cyber, using the Compass language. I'd like to try it."**

In August of 1978, your faithful practitioner received a listing of Forth in Compass Assembler for the CDC 6600. This was submitted to the Forth Implementation Workshop by Gregory Walker of the Population Research Center of the University of Texas at Austin. About thirty-five words were written in code. Only ten were complex. Mr. Walker's method was to write the text interpretation words in machine code and to compile most of what is usually code nucleus words in high-level form. This inverts the usual form but is ideal from a portability standpoint. The initial vocabulary is in the general form

```
@ ! : ; INTEGER INTERPRET EXECUTE STATE
NUMBER FIND NAME LOAD + - * / OR AND
XOR  EQZ LTZ CURRENT PUTDICT
```

A beautiful touch was his method for the common stack operations. Gregg discards into a variable called **TRASH** in this fashion:

```
0 INTEGER TRASH
0 INTEGER TOP
: DROP TRASH ! ;
: DUP TRASH ! TRASH @ TRASH @ ;
: SWAP TOP ! TRASH ! TOP @ TRASH @ ;
```

We were not able to include Gregg's work in the FIG implementations since the market acceptance was uncertain and our CDC 6600 was unavailable for testing.

MicroMotion

MasterFORTH

It's here – the next generation of MicroMotion Forth.

- Meets all provisions, extensions and experimental proposals of the FORTH-83 International Standard.
- Uses the host operating system file structure (APPLE DOS 3.3 & CP/M 2.x).
- Built-in micro-assembler with numeric local labels.
- A full screen editor is provided which includes 16 x 64 format, can push & pop more than one line, user definable controls, upper/lower case keyboard entry, A COPY utility moves screens within & between lines, line stack, redefinable control keys, and search & replace commands.
- Includes all file primitives described in Kernigan and Plauger's Software Tools.
- The editor, assembler and screen copy utilities are provided as relocatable object modules. They are brought into the dictionary on demand and may be released with a single command.
- Many key nucleus commands are vectored. Error handling, number parsing, keyboard translation and so on can be redefined as needed by user programs. They are automatically returned to their previous definitions when the program is forgotten.
- The string-handling package is the finest and most complete available.
- A listing of the nucleus is provided as part of the documentation.
- The language implementation exactly matches the one described in FORTH TOOLS, by Anderson & Tracy. This 200 page tutorial and reference is included with MasterFORTH.
- The input and output streams are fully redirectable.
- Floating Point & HIRES options available.
- Available for APPLE II/II+/IIe & CP/M 2.x users.
- MasterFORTH – $100.00. FP & HIRES – $40.00 each
- Publications
  - FORTH TOOLS – $20.00
  - 83 International Standard – $15.00
  - FORTH-83 Source Listing 6502, 8080, 8086 – $20.00 each.

Contact:

MicroMotion
12077 Wilshire Blvd., Ste. 506
Los Angeles, CA 90025
(213) 821-4340

Techniques Tutorial

Part Two

Debugging Techniques

Henry Laxen
Berkeley, California

Last time, we looked at machine-independent ways of debugging in Forth. We implemented the word **UNRAVEL** which, when executed, prints on the terminal the current nesting structure of Forth. This is very useful for run-time error checking. Now we are going to look at what kind of debugging tools we can implement if we are willing to descend to the depths of assembly language, and if we have a working knowledge of the internal structure of Forth. I will assume that you know how Forth works and will not endeavor to explain the workings of **NEXT** or **NEST**ing or **UNNEST**ing. If these things are unfamiliar to you, I suggest you refer to Tracy's *Forth Tools* or to Brodie's *Starting Forth* for further information. If you are unfortunate enough to own an IBM PC or compatible, you may be able to mindlessly use the code I am presenting. These same tools can be implemented on other CPUs but the code words must be modified, of course. Mike Perry and I have implemented these debugging words in our Forth-83 system on the 8080, 8086 and 68000 CPUs with little or no difficulty.

All of the following techniques will only work if **NEXT** is not executed in-line, but is jumped to. The techniques rely on patching **NEXT** to perform additional functions besides incrementing the IP and jumping to the next word to execute. Because of that, the run time of the system will be affected. By judicious use of code, this overhead can be kept to well under fifty percent, which is not bad for debugging.

The first and most useful debugging tool allows you to selectively single-step trace through the Forth word of your choice. One of the very nice things about this implementation is that you need not re-compile anything in order

```
\ Debugging - Low Level
HEX
VARIABLE 'DEBUG      ( Holds CFA of TRACE routine )
VARIABLE <IP         ( Lower limit of IP during tracing )
VARIABLE IP>         ( Upper limit of IP during tracing )
CODE GOTO  ( n -- )  IP POP  AX LODS  AX W MOV  0 [W] JMP  C;
CODE UNBUG ( -- )
   BYTE 0AD # >NEXT #) MOV  0D88B # >NEXT 1+ #  MOV NEXT C;
ASSEMBLER LABEL DEBNEXT
   <IP #) IP CMP  U> IF
      IP> #) IP CMP  U<= IF
         'DEBUG #) W MOV  0 [W] JMP
   THEN  THEN  ' GOTO @ 1+ #) JMP ( Back to NEXT )
CODE PNEXT   ( -- )
   BYTE 0E9 # >NEXT #) MOV
   DEBNEXT >NEXT 3 + - # >NEXT 1+ #) MOV   NEXT  C;  DECIMAL
```

Figure One

```
\ Debugging - High Level
: L.ID   ( cfa len -- )
   OVER >NAME DUP .ID  ROT 1- - + SPACES   ;
: 'UNNEST   ( pfa -- pfa' )
   BEGIN   1+ DUP @  ['] UNNEST = UNTIL   ;
VARIABLE RES       ( Used to RESUME tracing )
: TRACE   ( -- )
   UNBUG   CR R> DUP @ 10 L.ID >R   .S SPACE   KEY UPC
   ASCII F OVER = IF
      RES OFF  DROP  BEGIN  QUERY RUN  RES @ UNTIL  THEN
   ASCII Q OVER = IF   DROP QUIT   THEN
   DROP   PNEXT   R> GOTO   ;   ' TRACE 'DEBUG !
: DEBUG   ( -- )
   UNBUG   '   DUP <IP !  'UNNEST IP> !   PNEXT   ;
: RESUME   ( -- n )
   RES ON   0   ;
```

Figure Two

```
\ Watch a Memory Location
VARIABLE WATCH-POINT 0 ,
ASSEMBLER LABEL WNEXT
   WATCH-POINT #) BX MOV  0 [BX] AX MOV
   WATCH-POINT 2+ #) AX CMP  0<> IF
      AX WATCH-POINT 2+ #) MOV  ' UNRAVEL # W MOV  0 [W] JMP
   THEN   ' GOTO @ 1+ #) JMP   ( Regular NEXT )
CODE PNEXT   ( -- )
   HEX BYTE 0E9 # >NEXT #) MOV   DECIMAL
   WNEXT >NEXT 3 + - # >NEXT 1+ #) MOV   NEXT   C;
WATCH   ( addr -- )
   DUP WATCH-POINT !   @ WATCH-POINT 2+ !   PNEXT   ;
```

Figure Three

to trace it. You can retroactively decide what to trace and, in fact, can change your mind in the middle of the trace and begin tracing a different word. This ability to debug previously compiled words is very powerful and usually requires hardware help. It is possible in Forth because the user has power over Forth's virtual machine, namely **NEXT**, and can modify it according to his whim.

The idea, then, is to modify **NEXT** so that in addition to incrementing the IP and jumping to the next word, it checks to see if the IP is within a certain range. If it is, it calls a special word that initiates debugging, instead of the word that it was going to execute. The idea is very simple but the implementation is complicated by the fact that the special word that **NEXT** is calling must itself use **NEXT** in order to implement the tracing function. Furthermore, we want to execute step-by-step the word being traced, so we sometimes need to turn off this special **NEXT**.

Now let's take a look at figures one and two. The variables **<IP** and **IP>** hold the range of IP values we are interested in tracing. Whenever the actual IP is between these two values, we will initiate tracing. Fortunately, the structure of high-level Forth words is such that they are contiguous in memory. Thus, while we are executing a particular high-level Forth word, the IP will always be greater than the code field address of that word, and less than the address of the **UNNEST** code field that was compiled by ;. The label **DEBNEXT** is the new version of **NEXT** that gets jumped to when tracing is enabled. It simply compares the IP value to the contents of the variables **<IP** and **IP>**. If the IP is between these values, the high-level word pointed to by the variable **'DEBUG** is executed. If the IP is not in that range, then a normal **NEXT** is executed and Forth proceeds as though nothing had happened. The words **UNBUG** and **PNEXT** are inverses of each other. **PNEXT** patches **NEXT** to point to **DEBNEXT**, and **UNBUG** restores **NEXT** back to how it used to be. 0E9 is a JMP instruction on the 8080, and AD880D is what **NEXT** normally is on an 8088. The last code word we need to worry about is **GOTO**, which is simply a high-level branch. It has its own in-line **NEXT** to avoid the problem of recursively tracing the same code field forever.

The high-level work is done in figure two. **L.ID** prints the name of a word left-justified in a fixed-length field. **'UNNEST** searches for the code field of **UNNEST** given a starting address, and returns the address found. In F83, the ; word compiles the code field of **UNNEST** while **EXIT** compiles a different code field that does the same thing. This allows F83 to implement this debugging feature and a nifty decompiler very simply, since colon definitions are now terminated by a unique code field.

Now let's take a look at **TRACE**. The first thing it does is turn off debugging. This is just in case you happen to be tracing a word that is being used inside **TRACE**. You would get very confused, I assure you. Next, the name of the word about to be executed is printed, along with the current parameter stack contents. The **R>** on line seven gives us the address inside the word we are tracing of the code field of the word about to be executed. Thus, when we fetch it, we have the actual code field of the next word to be executed. Notice that our **DEBNEXT** version of **NEXT** has not yet incremented the IP; thus, the current address on the return stack is really the address of the current word that was about to be executed. We must replace this address on the return stack with the **>R** before printing out the parameter stack, or else confusion will again result. Next we wait for a **KEY**. At this point the user has three choices. If you want to continue single-stepping through the word, press any key other than Q or F. If you press Q for Quit, tracing will cease and you will be returned to the Forth interpreter. If you press F, you will re-enter the Forth interpreter, but will be able to continue tracing right where you left off if you type the word **RESUME**.

Let's take a look at how this works and what it really means. First, the variable **RES** is turned **OFF**, meaning it gets set to zero. The following **DROP** throws away the character you typed. Thus, what is on the stack at this point is exactly what was there when tracing began. The **BEGIN...UNTIL** loop repeatedly gets a line of input from the terminal with **QUERY** and then executes it with **RUN**. (On your system, **INTERPRET** may have to be substituted for **RUN**.) This continues until the variable **RES** is set to a non-zero value. Looking at **RESUME**, that is precisely what it does, as well as leaving a value on the stack to make up for the **DROP** that threw away the keystroke. This ability to re-enter Forth and then **RESUME** debugging is extremely powerful. At any point while you are tracing, you have the full power of Forth at your disposal. For example, you could examine variables, dump memory, list screens, or anything you want, right in the middle of executing a word. And then, by typing **RESUME** you can pick up right where you left off before re-entering Forth. One note of caution! If you make a mistake while you have re-entered Forth, the error handling of Forth will take over and you will wind up back in the Forth interpreter and unable to **RESUME** debugging of the word you were in. Such is life.

The user interface to all of this is the word **DEBUG**. It should be followed by the name of the word you want to debug; for example, **DEBUG WORDS**. Then the next time **WORDS** is executed, it will be single-step traced. All **DEBUG** does is set up the variables **<IP** and **IP>**, and patch **NEXT** to point to **DEBNEXT**. Needless to say, you can make your own custom version of **TRACE** and simply place its code field address in the variable **'DEBUG**. Then it will receive control whenever the IP is within the range specified. The possibilities are only limited by your imagination.

Finally, a totally different but sometimes invaluable tool is presented in figure three. Again **NEXT** is patched to point to **WNEXT**. **WNEXT** monitors a particular memory word whose address is at **WATCH-POINT** and whose initial contents is at **WATCH-POINT 2+**. If the value at that memory location differs from **WATCH-POINT 2+** then we call **UNRAVEL**, which will give us a trace of our return stack. Thus, we will be able to see exactly where we were at the time the location changed. This is an incredible tool usually found only in hardware emulators. It allows us to catch a "random store" bug almost immediately, in fact before the execution of the next high-level word: and the cost is one screen of code.

I hope these debugging ideas will come in handy for you. They have saved me countless hours. Good luck, and may the Forth be with you.

33 KFLOPS

Use your IBM PC (or compatible) to multiply two 128 by 128 matrices at the rate of 33 thousand floating-point operations per second (kflops)! Calculate the mean and standard deviation of 16,384 points of single precision (4 byte) floating-point data in 1.4 seconds (35 kflops). Perform the fast Fourier transform on 1024 points of real data in 6.5 seconds. Near PDP-11/70 performance when running the compute intensive Owen benchmark.

WL FORTH-79

FORTH-79 by WL Computer Systems is a powerful and comprehensive programming system which runs on the IBM PC (and some compatibles). If your computer has the 8087 numeric data processing chip (NDP) installed, then this version of FORTH-79 will unleash the awesome floating-point processing power which is present in your system. If you haven't gotten around to installing the 8087 NDP coprocessor in your computer, you can still use WL FORTH to write applications using standard FORTH-79.

8087 support and other features

WL FORTH features extremely fast floating point calculations because it uses the 8087 hardware stack to store intermediate results and achieve 16 to 18 digits precision. The system includes a large set of transcendental functions, such as SIN, COS, TAN, ASIN, ACOS, ATAN, Y^Z, LN, LOG, SQRT. FORTRAN like conversion specification words allow the user to specify output field width, places beyond the decimal point and fixed or scientific notation.

The FORTH assembler allows the user to code time critical words in 8087/8088 assembly language and includes structured branch and looping constructs. The entire 1 Mb address space is available for array storage. Definitions can include SWITCH to different screen files, thereby allowing dynamic switching of screen files. SAVE allows current system to be saved as a .COM file and ZAP prevents your applications from being decompiled. The system includes interrupt driven exception handlers for both the 8087 and 8088, and the programmer can select the desired number of screen buffers.

But can I get the source?

Unlike most other products, the **complete** source is available at a very affordable price.

Package 1 includes FORTH-79 versions with and without 8087 support. Included are screen utilities, 8087 and 8088 FORTH assemblers. $100

Package 2 includes package 1 plus the assembly language source for the WL FORTH-79 nucleus. $150

Package 3 includes package 2 plus the WL FORTH-79 source screens used to add the 8087 features to the vocabulary. $200

Starting FORTH book. $22

WL Computer Systems
1910 Newman Road
W. Lafayette, IN 47906
(317) 743-8484

Visa and Master Card accepted.

IBM is a trademark of International Business Machines

Chapter News

John D. Hall
Oakland, California

We have five new chapters. That makes sixty-two!

Kodiak Area FIG Chapter, Kodiak, Alaska

Monterey/Salinas FIG Chapter, Salinas, California

Fort Wayne FIG Chapter, Fort Wayne, Indiana

Albuquerque FIG Chapter, Albuquerque, New Mexico

Cincinnati FIG Chapter, Cincinnati, Ohio

Arizona FIG Chapter

May 24: Charles Moore attended! He discussed his Forth computer chip's architecture in detail. At the time, he was in Arizona testing prototypes of the chip prior to production runs. An overflow crowd attended and the meeting was video-taped. The tapes may be available for distribution after approvals are obtained.

Orange County FIG Chapter

February 22: Wil Baden presented his FIX-FORTH which fixes the F83 Model so that it will meet the 83 Standard. John Broderick presented a program for cross-referencing words and where they are used.

March 28: Dan Slater spoke to the group and presented a slide show of the work he did at TRW, and a robot camera he worked on for Hollywood space movies. The group was very impressed with the work he was able to turn out. Wil Baden reported that he spoke on the virtues of Forth to the Rockwell Space Systems Group at the Lunch Hour Computer Club.

April 4: Dave Harralson presented a paper on the generalization of Forth control structures. Wil Baden also spoke about control logic.

Sacramento FIG Chapter

May 22: Fifteen people attended, including Jack Park, author of MVP's Expert System. We discussed our reasons for attending a FIG meeting and tried to identify areas of mutual interest. Mr. Park identified an area of concern for the chapters: that of reconciling our levels of experience in Forth. If a topic or speaker addresses a subject at an expert level, the less experienced won't be able to follow and will become bored. If it is addressed at a novice level, the more experienced won't need to follow and will become bored. Tom Ghormley is sure that their chapter is not the first to address this. Any suggestions? *(J.H.: This sounds like a discussion of the blind leading the blind and the faithful only talking to the faithful. In reality, it shouldn't be this polarized. A chapter meeting is a place to discuss, criticize and learn by all. It's not a place to accept someone's ideas as the "truth" any more than it is a place to present the "true" solution. In all cases, papers and presentations are tutorials for the author as well as for the audience, and the process to obtain the solution should be the main message, not the product. With this attitude, both "novices" and "experts" are learning, and the only distinction is how far along each is in the learning process. As soon as the topic changes slightly, the role of "novice" and "expert" will possibly change. The test of the strength of the Forth community and chapter then becomes how well those further along can assist those earlier in the process.)*

June 12: Walt Winter of Engineering Logic gave a talk on his Forth-operated TM990 control system. Wes Lane gave a talk on the 83 Standard experimental proposed words **ONLY** and **ALSO** as well as a short exposition of direct, indirect and tokenized threaded code. Tom Ghormley passed around his article on Forth for comments and corrections. It appeared in the July issue of the *Sacramento Valley Computer News.*

July 10: Twenty-two people attended the meeting, several because of Tom Ghormley's article. A demonstration of Forth application using a Commodore 64 and several J&J instruments had to be postponed because of a bad disk. Bob Nash was rushed into service and gave a short comparison of three versions of Forth available for the Commodore 64 as well as a more in-depth look at SuperForth 64, complete with demo.

Connecticut FIG Chapter

May 4: The first FIG meeting of the Connecticut chapter was held at the Meridan Public Library. The group had an excellent assortment of personalities and programming experience, and was especially fortunate to have in attendance two very knowledgeable and experienced Forthers (Forthites, Forthies...?): John Moran, a software group leader; and Bryan Lockwood, an independent consultant. John told the group about the trials and tribulations of getting Forth in ROM and his use of a metacompiler to do so. He also discussed his experience getting Forth up on the VAX. Bryan explained some of the problems and limitations of trying to access more than 64K of address space from a sixteen-bit bus. He also gave a short talk on headerless code as a means of saving memory. Dan Kern-Ekins, an independent consultant and president of a local Commodore 64 club, told us about his version of Forth for that machine, and described some of the peculiarities of his disk operation. Tom Evans, a software engineer, spoke of his use of Forth at work for data acquisition and analysis of optical surfaces. Mike Davis talked about the new IBM-PC he got at work and his plans to get Forth for the NEC-8201A. Jean-Pierre Jaborska surprised many of the group when he commented that the Commodore Pet, his Forth machine, is still being manufactured. Jean-Pierre, who is from France, was disappointed that so many Forth articles are at the system level, and so few talk about real-world applications — a good point. Charlie Krajewski outlined a Forth project in which he hooked his bicycle to an 8088-based computer for indoor winter training. Unfortunately, the project was finally completed in early May.

Forth Gesellschaft

April 28: Sixteen people got together to get the ball rolling for a German FIG chapter. Three of us are "old FIG hands" who, once upon a time, keyed the FIG listings into their machines. We are going to work as a chapter under the name of Forth Gesellschaft and we will publish a newsletter under the name of *Vierte Dimension*. The first issue will be mailed by the end of May and will feature a translation of Bill Ragsdale's interview from *Forth Dimensions* (V/6). Six working groups constituted themselves:

1) Forth-83: we will be distributing Laxen and Perry's F83 and work is underway to translate the documentation into German.

2) Leibniz: one of the major difficulties in gaining wide acceptance for Forth is constituted by the language barrier. Hence, we are going to create a "new" programming language, under the name of Leibniz, which will have German names and will follow the Forth-83 Standard semantically. Leibniz will be derived from the version of Forth which was developed by Klaus Scheisiek and it will be put into the public domain.

3) One working group will keep tabs on Forth publications and products, and will compile an index.

4) Newsletter: *Vierte Dimension* will be published approximately four times a year as a forum and communications vehicle for Forth users in Germany. Initial subscription is 23,- DM for individuals and 55,- DM for corporations.

5) Fifth Dimension: this is a "brainstorming" group which is going to envision the future development of Forth toward friendly and not necessarily Forth-like user interfaces and higher-level constructs. We will see what they are going to come up with. They want to use Forth as a base on which to build more sophisticated structures.

6) Operations and management: Horst-Günter Lynsche was elected secretary of Forth Gesellschaft and he will be responsible for the management aspects — keeping tabs on the subscribers, publishing *Vierte Dimension*, etc.

TOTAL CONTROL:

***FORTH:* FOR Z-80®, 8086, 68000, and IBM® PC**

Complies with the New 83-Standard

GRAPHICS • GAMES • COMMUNICATIONS • ROBOTICS
DATA ACQUISITION • PROCESS CONTROL

- ***FORTH*** programs are instantly portable across the four most popular microprocessors.
- ***FORTH*** is interactive and conversational, but 20 times faster than BASIC.
- ***FORTH*** programs are highly structured, modular, easy to maintain.
- ***FORTH*** affords direct control over all interrupts, memory locations, and i/o ports.
- ***FORTH*** allows full access to DOS files and functions.
- ***FORTH*** application programs can be compiled into turnkey COM files and distributed with no license fee.
- ***FORTH*** Cross Compilers are available for ROM'ed or disk based applications on most microprocessors.

Trademarks: IBM, International Business Machines Corp.; CP/M, Digital Research Inc.; PC/Forth+ and PC/GEN, Laboratory Microsystems, Inc.

FORTH Application Development Systems include interpreter/compiler with virtual memory management and multi-tasking, assembler, full screen editor, decompiler, utilities, and 130 + page manual. Standard random access files used for screen storage, extensions provided for access to all operating system functions.

Z-80 FORTH for CP/M® 2.2 or MP/M II, $50.00; **8080 FORTH** for CP/M 2.2 or MP/M II, $50.00; **8086 FORTH** for CP/M-86 or MS-DOS, $100.00; **PC/FORTH** for PC-DOS, CP/M-86, or CCPM, $100.00; **68000 FORTH** for CP/M-68K, $250.00.

FORTH + Systems are 32 bit implementations that allow creation of programs as large as 1 megabyte. The entire memory address space of the 68000 or 8086/88 is supported directly.

PC FORTH +	$250.00
8086 FORTH + for CP/M-86 or MS-DOS	$250.00
68000 FORTH + for CP/M-68K	$400.00

Extension Packages available include: software floating point, cross compilers, INTEL 8087 support, AMD 9511 support, advanced color graphics, custom character sets, symbolic debugger, telecommunications, cross reference utility, B-tree file manager. *Write for brochure.*

VISA

LMI

Laboratory Microsystems Incorporated
4147 Beethoven Street, Los Angeles, CA 90066
*Phone credit card orders to **(213) 306-7412***

MasterCard

New Chapters in Formation

Here are more of the new chapters that are forming. If you live in any of these areas, contact these people and offer your support and help in forming a FIG chapter. You are not expected to be one of the "experts." The job of organizing a chapter can be done as well by people who are better at organizing than at programming, or by people who are in need of the help and support that a chapter can return. Lend a hand!

Bruce N. Collins
c/o Malemute Software
P.O. Box 81746
College, AK 99708

Gary Smith
Hawg Wild Software
P.O. Box 7668
Little Rock, AR 72217

Herman B. Gibson
8014 Gondola Dr.
Orlando, FL 32809

Alexander Luoma
P.O. Box 10432
Talahassee, FL 32302

Richard Wagner
728 E. Colfax Ave.
South Bend, IN 46617

Michael G. Waldon
Inst. of Environmental Studies
Rm. 42, Atkinson Hall
Louisiana State University
Baton Rouge, LA 70803-5705
504/388-8521

Claude W. Hesselman
2545 Bainbridge Blvd.
Chesapeake, VA 23324
804/545-1240

Jean-Marc Bertinchamps
Rue N. Monnom, 2
B-6290 Nalinnes
Belgium

Darryl C. Oliver
Pacer Electronics
2302 Marengo
New Orleans, LA 70115
504/899-8922

Tom Chrapkiewicz
P.O. Box 1056
Dearborn, MI 48121
313/524-2100

Gene Embry
Route 1, Box 151-H
Morrisville, NC 27560

Bill Morrissey
Rio Grande Electronics
1595 W. Picacho, Ste. 28
Las Cruces, NM 88005

J. Rennie
1809 N.W. 34th
Oklahoma City, OK 73118

Phillip A. Marciner
243 Judith Dr.
Johnstown, PA 15905

Bard Ermentrout
5464 Upsal Place
Pittsburgh, PA 15206

Terry L. Wallis
322 Haverford
San Antonio, TX 78217

Al Amway
Rare Earth Services, Inc.
3115 Willow Rd. N.W.
Roanoke, VA 24017

David Caulkins
Mad Apple FIG
P.O. Box 5103
Madison, WI 53705

Ray St. Laurent
P.O. Box 95
Vars, ON K0A 3H0
Canada

Jan Langerad
Loebpakken 1
3520 Sarum
Denmark

Mrs. J.J. van der Hoek
Forth Interesse Groep
Medemblikpad 70
8304 CZ Emmeloord
The Netherlands

Multiuser/Multitasking
for 8080, Z80, 8086

Industrial Strength FORTH

TaskFORTH™
The First
Professional Quality
Full Feature FORTH
System at a micro price*

LOADS OF TIME SAVING PROFESSIONAL FEATURES:

☆ Unlimited number of tasks
☆ Multiple thread dictionary, superfast compilation
☆ Novice Programmer Protection Package™
☆ Diagnostic tools, quick and simple debugging
☆ Starting FORTH, FORTH-79, FORTH-83 compatible
☆ Screen and serial editor, easy program generation
☆ Hierarchical file system with data base management

* Starter package $250. Full package $395. Single user and commercial licenses available.

If you are an experienced FORTH programmer, this is the one you have been waiting for! If you are a beginning FORTH programmer, this will get you started right, and quickly too!

Available on 8 inch disk under CP/M 2.2 or greater also various 5¼" formats and other operating systems

FULLY WARRANTIED, DOCUMENTED AND SUPPORTED

master charge DEALER INQUIRES INVITED VISA

Shaw Laboratories, Ltd.
24301 Southland Drive, #216
Hayward, California 94545
(415) 276-5953

FIG Chapters

U.S.

• ALASKA

Kodiak Area Chapter
Call Norman C. McIntosh
907/486-4843

• ARIZONA

Phoenix Chapter
Call Dennis L. Wilson
602/956-7678

Tucson Chapter
Twice Monthly, 2nd & 4th Sun., 2 p.m.
Flexible Hybrid Systems
2030 E. Broadway #206
Call John C. Mead
602/323-9763

• CALIFORNIA

Berkeley Chapter
Monthly, 2nd Sat., 1 p.m.
10 Evans Hall
University of California
Berkeley
Call Mike Perry
415/644-3421

Los Angeles Chapter
Monthly, 4th Sat., 11 a.m.
Allstate Savings
8800 So. Sepulveda Boulevard
½ mile North of LAX
Los Angeles
Call Phillip Wasson
213/649-1428

Monterey/Salinas Chapter
Call Bud Devins
408/633-3253

Orange County Chapter
Monthly, 4th Wed., 7 p.m.
Fullerton Savings
Talbert & Brookhurst
Fountain Valley
Monthly, 1st Wed., 7 p.m.
Mercury Savings
Beach Blvd., & Eddington
Huntington Beach
Call Noshir Jesung
714/842-3032

San Diego Chapter
Weekly, Thurs., 12 noon.
Call Guy Kelly
619/268-3100 ext 4784

Sacramento Chapter
Monthly, 2nd Tues., 7 p.m.
170B 59th St., Room C
Call Tom Ghormley
916/444-7775

Silicon Valley Chapter
Monthly, 4th Sat., 1 p.m.
Dysan Auditorium
5201 Patrick Henry Dr.
Santa Clara
Call Glenn Tenney
415/574-3420

Stockton Chapter
Call Doug Dillon
209/931-2448

• COLORADO

Denver Chapter
Monthly, 1st Mon., 7 p.m.
Call Steven Sarns
303/477-5955

• CONNECTICUT

Central Connecticut Chapter
Monthly, 1st Thurs., 7 p.m.
Meriden Public Library
Call Charles Krajewski
203/344-9996

• FLORIDA

Southeast Florida Chapter
Miami
Call John Forsberg
305/252-0108

• ILLINOIS

Central Illinois Chapter
Urbana
Call Sidney Bowhill
217/333-4150

Fox Valley Chapter
Call Samuel J. Cook
312/879-3242

Rockwell Chicago Chapter
Call Gerard Kusiolek
312/885-8092

• INDIANA

Central Indiana Chapter
Monthly, 3rd Sat., 10 a.m.
Call Richard Turpin
317/923-1321

Fort Wayne Chapter
Call Blair MacDermid
219/749-2042

• IOWA

Iowa City Chapter
Monthly, 4th Tues.
Engineering Bldg., Rm. 2128
University of Iowa
Call Robert Benedict
319/337-7853

• KANSAS

Wichita Chapter (FIGPAC)
Monthly, 3rd Wed., 7 p.m.
Wilbur E. Walker Co.
532 S. Market
Wichita, KS
Call Arne Flones
316/267-8852

• MASSACHUSETTS

Boston Chapter
Monthly, 1st Wed.
Mitre Corp. Cafeteria
Bedford, MA
Call Bob Demrow
617/688-5661 after 7 p.m.

• MINNESOTA

MNFIG Chapter
Even month, 1st Mon., 7:30 p.m.
Odd Month, 1st Sat., 9:30 a.m.
Vincent Hall Univ. of MN
Minneapolis, MN
Call Fred Olson
612/588-9532

• MISSOURI

Kansas City Chapter
Monthly, 4th Tues., 7 p.m.
Midwest Research Inst.
Mag Conference Center
Call Linus Orth
816/444-6655

St. Louis Chapter
Monthly, 3rd Tues., 7 p.m.
Thornhill Branch of
St. Louis County Library
Call David Doudna
314/867-4482

• NEVADA

Southern Nevada Chapter
Suite 900
101 Convention Center Drive
Las Vegas, NV
Call Gerald Hasty
702/452-3368

• NEW MEXICO

Albuquerque Chapter
Call Rick Granfield
505/296-8651
William Edmonds
804/898-4099

• NEW YORK

FIG, New York
Monthly, 2nd Wed., 8 p.m.
Queens College
Call Tom Jung
212/432-1414 ext. 157 days
212/261-3213 eves.

Rochester Chapter
Bi-monthly, 4th Sat., 2 p.m.
Hutchison Hall
Univ. of Rochester
Call Thea Martin
716/235-0168

Syracuse Chapter
Monthly, 1st Tues., 7:30 p.m.
Call C. Richard Corner
315/456-7436

• OHIO

Athens Chapter
Call Isreal Urieli
614/594-3731

Cleveland Chapter
Call Gary Bergstrom
216/247-2492

Cincinatti Chapter
Call Douglas Bennett
513/831-0142

Dayton Chapter
Twice monthly, 2nd Tues &
4th Wed., 6:30 p.m.
CFC 11 W. Monument Ave.
Suite 612
Dayton, OH
Call Gary M. Granger
513/849-1483

• OKLAHOMA

Tulsa Chapter
Monthly, 3rd Tues., 7:30 p.m.
The Computer Store
4343 South Peoria
Tulsa, OK
Call Art Gorski
918/743-0113

• OREGON

Greater Oregon Chapter
Monthly, 2nd Sat., 1 p.m.
Computer & Things
3460 SW 185th, Aloha
Call Timothy Huang
503/289-9135

• PENNSYLVANIA

Philadelphia Chapter
Monthly, 3rd Sat.
LaSalle College, Science Bldg.
Call Lee Hustead
215/539-7989

• TEXAS

Dallas/Ft. Worth
Metroplex Chapter
Monthly, 4th Thurs., 7 p.m.
Software Automation, Inc.
14333 Porton, Dallas
Bill Drissel
214/264-9680

Houston Chapter
Call Dr. Joseph Baldwin
713/749-2120

• VERMONT

Vermont FIG Chapter
Monthly, 3rd Mon., 7:30 p.m.
Vergennes Union High School
Rm. 210, Monkton Rd.
Vergennes, VT
Call Hal Clark
802/877-2911 days
802/452-4442 eves

• VIRGINIA

Norfolk FIG Chapter
Call William Edmonds
804/898-4099

Potomac Chapter
Monthly, 1st Tues., 7 p.m.
Lee Center
Lee Highway at Lexington St.
Arlington, VA
Call Joel Shprentz
703/437-9218 eves.

Richmond Forth Group
Monthly, 2nd Wed., 7 p.m.
Basement, Puryear Hall
Univ. of Richmond
Call Donald A. Full
804/739-3623

FOREIGN

• AUSTRALIA

Melbourne Chapter
Monthly, 1st Fri., 8 p.m.
Contact: Lance Collins
65 Martin Road
Glen Iris, Victoria 3146
03/29-2600

Sydney Chapter
Monthly, 2nd Fri., 7 p.m.
John Goodsell Bldg.,
Rm. LG19
Univ. of New South Wales
Sydney
Contact: Peter Tregeagle
10 Binda Rd., Yowie Bay
02/524-7490

• BELGIUM

Belgium Chapter
Monthly, 4th Wed., 20:00h
Contact: Luk Van Loock
Lariksdreff 20
2120 Schoten
03/658-6343

Southern Belgium FIG Chapter
Contact: Jean-Marc Bertinchamps
Rue N. Monnom, 2
B-6290 Nalinnes
Belgium
071/213858

• CANADA

Nova Scotia Chapter
Contact: Howard Harawitz
227 Ridge Valley Rd.
Halifax, Nova Scotia B3P 2E5
902/477-3665

Southern Ontario Chapter
Monthly, 1st Sat., 2 p.m.
General Sciences Bldg.
Rm. 312
McMaster University
Contact: Dr. N. Solntseff
Unit for Computer Science
McMaster University
Hamilton, Ontario L8S 4K1
416/525-9140 ext. 2065

Toronto FIG Chapter
Contact: John Clark Smith
P.O. Box 230, Station H
Toronto, ON M4C 5J2

• COLOMBIA

Colombia Chapter
Contact: Luis Javier Parra B.
Aptdo. Aereo 100394
Bogota
214-0345

• ENGLAND

Forth Interest Group — U.K.
Monthly, 1st Thurs., 7 p.m., Rm. 408
Polytechnic of South Bank
Borough Rd., London
Contact: Keith Goldie-Morrison
Bradden Old Rectory
Towchester, Northamptonshire
NN12 8ED

• FRANCE

French Language Chapter
Contact: Jean-Daniel Dodin
77 rue du Cagire
31100 Toulouse
(16-61) 44.03

• GERMANY

Hamburg FIG Chapter
Monthly, 4th Sat., 1500 hrs.
Contact: Horst-Gunter Lynsche
Holstenstr. 191
D-2000 Hamburg 50

• IRELAND

Irish Chapter
Contact: Hugh Dobbs
Newton School
Waterford
051/75757
051/74124

• ITALY

FIG Italia
Contact: Marco Tausel
Via Gerolamo Forni 48
20161 Milano
02/645-8688

• REPUBLIC OF CHINA

R.O.C.
Contact: Ching-Tang Tzeng
P.O. Box 28
Lung-Tan, Taiwan 325

• SWITZERLAND

Swiss Chapter
Contact: Max Hugelshofer
ERNI & Co. Elektro-Industrie
Stationsstrasse
8306 Bruttisellen
01/833-3333

SPECIAL GROUPS

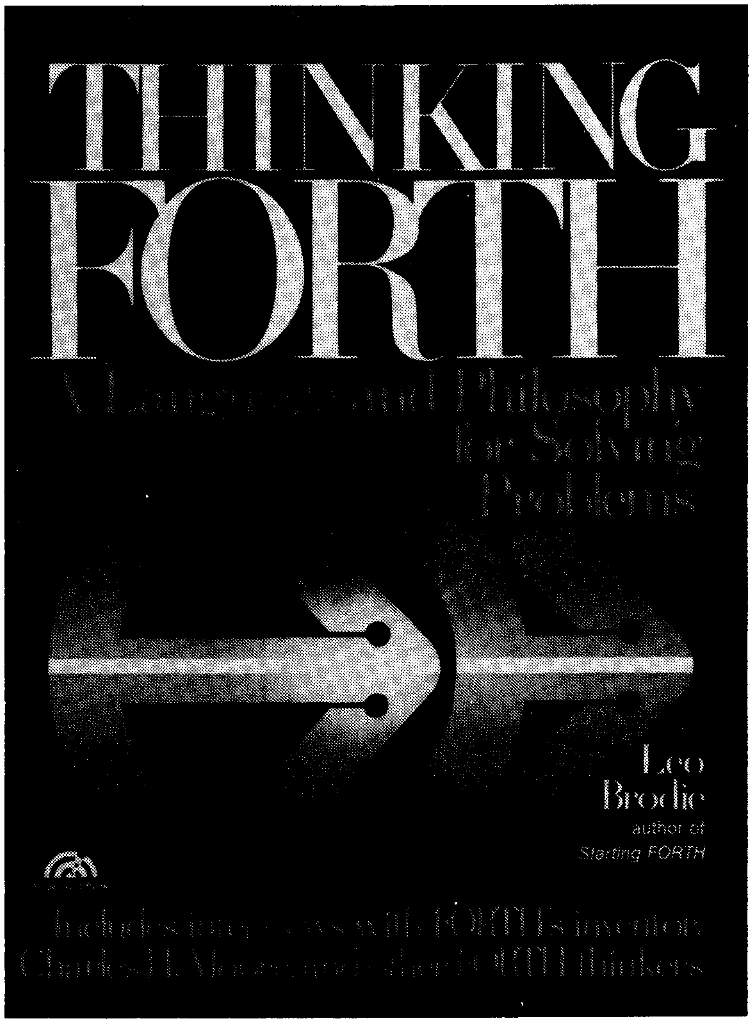
Apple Corps Forth Users Chapter
Twice Monthly, 1st &
3rd Tues., 7:30 p.m.
1515 Sloat Boulevard, #2
San Francisco, CA
Call Robert Dudley Ackerman
415/626-6295

Baton Rouge Atari Chapter
Call Chris Zielewski
504/292-1910

Detroit Atari Chapter
Monthly, 4th Wed.
Call Tom Chrapkiewicz
313/524-2100

FIGGRAPH
Call Howard Pearlmutter
408/425-8700

ANNOUNCING


THINKING FORTH
A Language and Philosophy for Solving Problems
Leo Brodie
author of Starting FORTH
Includes interviews with FORTH's inventor Charles H. Moore and other FORTH thinkers

ORDER FROM THE FORTH INTEREST GROUP

COMPLETE ORDER FORM ON PAGE 22

FORTH INTEREST GROUP
P.O. Box 1105
San Carlos, CA 94070

BULK RATE
U.S. POSTAGE
PAID
Permit No. 3107
San Jose, CA

ROBERT SMITH
2300 ST. FRANCIS DR.
PALO ALTO, CA 94303

Address Correction Requested