FORTH IN OPTIMAL CONTROL

ANS FORTH: REQUIRED WORDS

SEARCH FOR BETTER NUMERIC INPUT

FIBONACCI RANDOM NUMBER GENERATOR
SC/FOX PCS32 Parallel Coprocessor System

Uses the 32-bit SC32\textsuperscript{TM} Forth CPU.
System speed options: 8 or 10 MHz.
Full-length 8- or 16-bit PC/XT/AT plug-in board.
64K to 1M byte, 0-wait-state static RAM.
Hardware expansion, two 50-pin strip headers.
Includes SC/Forth32, based on the Forth-83 Standard.

SC/FOX PCS Parallel Coprocessor System

Uses Harris RTX 2000\textsuperscript{TM} real-time Forth CPU.
System speed options: 8 or 10 MHz.
Full-length 8- or 16-bit PC/XT/AT plug-in board.
32K to 1M bytes, 0-wait-state static RAM.
Hardware expansion, two 50-pin strip headers.
Includes FCompiler; SC/Forth optional.

SC/FOX SBC Single Board Computer

Uses RTX 2000 real-time Forth CPU.
System speed options: 8, 10, or 12 MHz.
32K to 512K bytes 0-wait-state static RAM.
RS232 56K-baud serial and printer ports.
Hardware expansion, two 50-pin strip headers.
64K bytes of shadow-EPROM space.
Eurocard size: 100\,mm by 160\,mm.
Includes FCompiler; optional SC/Forth EPROM.

SC/FOX SCSI I/O Daughter Board

Plug-on daughter board for SC/FOX PCS and SBC.
Source s/w drivers for FCompiler and SC/Forth.
SCSI adaptor with 5 Mbytes/sec synchronous or
3 Mbytes/sec asynchronous transfer rates.
Floppy disk adaptor, up to 4 drives, any type.
Full RS-232C Serial Port, 50 to 56K Baud.
16-bit bidirectional, latching parallel port.

SC/Forth\textsuperscript{TM} Language

Based on the Forth-83 Standard.
15-priority time-sliced multitasking.
Supports user-defined PAUSE.
Automatic optimization and \texttt{\#}code support.
Turnkey application support.
Extended structures and case statement.
Double number extensions.
Infix equation notation option.
Block or text file interpretation.
Optional source-code developer system.

Ideal for embedded real-time control, high-speed data acquisition and reduction, image or signal processing, or computation-intense applications. For additional information, please contact us at:

SILICON COMPOSERS INC 208 California Avenue, Palo Alto, CA 94306 (415) 322-8763
What, exactly, is ANS Forth and how will the changes it introduces affect your programming practices and existing code? A member of the standardization committee reports that there are hard decisions to make and provides plenty of details to mull over until his next report reaches us. This installment describes some differences between Forth-83 and ANS Forth.

FIBONACCI RANDOM NUMBER GENERATOR
NATHANIEL GROSSMAN

No method is known for producing truly random numbers on a digital computer, but 'pseudo-random' number generators are acceptable if they pass enough randomness tests. This paper presents a particularly simple random number generator, and describes how a suitably extended standard Forth package and a text formatter can be used to write readable, well-commented code.

FORTH IN OPTIMAL CONTROL
J.B. HO, P.Y. KOKATE, M. HUDA, R. HASKELL, N.K. LOH

"Optimal control" has been used in the process industry, the space program, and the defense industry. The linear quadratic regulator is the most commonly used form of optimal controller where the control law is obtained by minimizing a quadratic cost functional. An LQR is implemented here on a fourth-order ball-balancing system in the laboratory.

INCREASE MEMORY FOR THE TI 99/4A - HOWARD H. ROGERS

The amount of available RAM in the TI99/4A after loading TI-Forth can be as little as 6K. This paper discusses a method of increasing that memory by over 8K, primarily for arrays. 16K of RAM is associated with the video display processor; 8K of this is unused in most modes and can be used by Forth with no conflicts.

VOLUME X INDEX - MIKE ELOLA

The comprehensive index to all material that appeared in our most recent volume. Use it to find an elusive article or to get references and inspiration for your work.

IN SEARCH OF A BETTER NUMBER INPUT ROUTINE
MIKE ELOLA

The author shares his process of developing a number input routine that is simple yet flexible, a search that began in response to a troublesome problem: Forth simply terminates execution if a number-conversion error is detected. The most dramatic end-user benefit imparted by his solution is its improved error handling; and it is easily modified via "picture strings" to display formatted numbers like dates and currency amounts even while they are being entered.
Software Submissions, Hardware Issue Update

Nathaniel Grossman’s original manuscript, “Fibonacci Random Number Generator” came better presented than many published pages and it presented a production challenge. Mathematical typesetting has always been a demanding field, certainly no less in these days of desktop publishing. There are standalone programs for typesetting formulas, but in the interests of publishing. There are standalone programs for typesetting formulas, but in the interests of

	
time we ended up using some of his original equations for photostatic reproduction here. It reminded us how far some typesetting programs have come and, at the same time, how far most of them have to go in terms of ease of use. In addition to his random number generator, Grossman’s ideas about commenting Forth source code are timely (and relevant to Glen Haydon’s proposal for commenting source code, published in FD X/6). When preparing his article, we respected the author’s style conventions—which exemplify his points—rather than following our usual style sheet. Let us know what you think.

There have been some questions about the upcoming issue on Forth hardware, which led me to reply with the following material. The call for articles about this topic (see the editorial in FD XI/2 and the advertisement elsewhere in this issue) was stated in the most general possible terms; the intention is not to get you to write to our specifications, but to convince you to write about what you consider important, interesting, challenging, and useful. I encourage a broad spectrum of hardware-related submissions, a spectrum that might range from the design of general-purpose microprocessors whose native language is Forth, to Forth-controlled embedded systems and custom hardware implementations, to objective details of your experience with any of the commercially available Forth hardware systems.

However, don’t let the above examples restrict you—choose a topic according to your interests and expertise. Then just consider what you would want in a thorough, well-written article on that topic. Would illustrations or source code help to express your ideas? Are there objective standards or benchmarks which could be applied? What bearing, if any, does the subject of your article have on the Forth community and on the business of micro-computers?

I hope this helps a little to stimulate your thinking. Rather than make it too complex and constrictive, I suggest just opening the floodgates.

We have extended the deadline for theme-related submissions to December 1, 1989. Articles received after that date will still be considered for publication but will not be among the three selected for payment.

Recently, I was looking at R.D. Lurie’s tutorial series about Forth in 68 Micro Journal. In the April 1989 installment, he cites Dan Johnson’s conditional construct maybe... then again... maybe not and says he loves those names. Maybe he does, but then again...

—Marlin Ouverson
Editor

Forth Dimensions
Published by the
Forth Interest Group
Volume XI, Number 4
November/December 1989
Editor
Marlin Ouverson
Advertising Manager
Kent Safford
Design and Production
Berglund Graphics

Forth Dimensions welcomes editorial material, letters to the editor, and comments from its readers. No responsibility is assumed for accuracy of submissions.

Subscription to Forth Dimensions is included with membership in the Forth Interest Group at $30 per year ($42 overseas air). For membership, change of address, and to submit items for publication, the address is: Forth Interest Group, P.O. Box 8231, San Jose, California 95155. Administrative offices and advertising sales: 408-277-0668.

Copyright © 1989 by Forth Interest Group, Inc. The material contained in this periodical (but not the code) is copyrighted by the individual authors of the articles and by Forth Interest Group, Inc., respectively. Any reproduction or use of this periodical as it is compiled or the articles, except reproductions for non-commercial purposes, without the written permission of Forth Interest Group, Inc. is a violation of the Copyright Laws. Any code bearing a copyright notice, however, can be used only with permission of the copyright holder.

About the Forth Interest Group

The Forth Interest Group is the association of programmers, managers, and engineers who create practical, Forth-based solutions to real-world needs. Many research hardware and software designs that will advance the general state of the art. FIG provides a climate of intellectual exchange and benefits intended to assist each of its members. Publications, conferences, seminars, telecommunications, and area chapter meetings are among its activities.

“Forth Dimensions (ISSN 0884-0822) is published bimonthly for $24.36 per year by the Forth Interest Group, 1330 S. Bascom Ave., Suite D, San Jose, CA 95128. Second-class postage paid at San Jose, CA. POSTMASTER: Send address changes to Forth Dimensions, P.O. Box 8231, San Jose, CA 95155.”
Keep Art Alive!

Dear Mr. Ouverson,

I would like to take this time to thank you for keeping Forth alive through the publication of Forth Dimensions. I have enjoyed it for the last two years and look forward to enjoying more of it. I think that publications like this one help to keep the "art" of programming open and alive. Keep up the good work.

Although I have never participated in the "Dimensions" in the past, I may find the time to do so in the future. Please send me a copy of the Writer's Guide. Also, I would be interested in any information you may have concerning Forth Interest Group chapters in my area.

Thank you for your time,
Jay E. Topping

Stack Caveat Cured

Dear Mr. Ouverson:

Mr. Paul Condon in FD XI/3 made a good observation in his comment on my article, "Forth Needs Three More Stacks." Indeed, IF is not supposed to alter the stack if it is being skipped. I checked my CSU Forth source code and found that IF only removes the flag if it is executed, and merely pushes a don't care if it is skipped. Therefore, item four of my article (FD XI/1, page 27) should read,

"4. The word IF will move the flag from the parameter stack to the condition stack if the flag on top of the condition stack is true; otherwise, it will push don't care onto the condition stack."

This only makes sense, because if words are being skipped, no flags will be pushed, therefore no flags should be popped! The high-level definition of IF should also be corrected as follows:

: IF
  3 S@
  IF 3 >S
  ELSE 0 3 >S
  THEN ; MUST-EXEC

Mr. Condon objected to the need to mark special words so that the interpreter will honor them regardless of the status of the condition stack. What I call Must-Exec words. This marking is necessary if we implement his alternative algorithm for a branchless IF; otherwise the "special actions" that should be taken by the interpreter when skipping over IF, ELSE, and THEN will be impossible because the interpreter won't see these words. I'd be interested to know if Mr. Condon has applied a similar algorithm to implement the CASE statement.

He also thought that, for this marking to work with compiled words, the interpreter will have to execute >LINK for every word executed. Perhaps he meant >NFA. CSU Forth compiles the NFA of words, therefore the overhead of checking the Must-Exec mark is minimal. I have not noticed any degradation of performance after I implemented the additional stacks as opposed to the standard way.

I'd like to assure Mr. Condon that all branch words are absent from CSU Forth. The LOOP and BEGIN constructs do not use the branch words. I have not talked about that in the article. I agree with him that some degree of branching will always be there and cannot be totally eliminated.

Last, about the definition of CASE containing IF and which IF the interpreter should execute. There is only one IF in CSU Forth and it's defined in assembly. The high-level definition I have in the article was for illustration purposes.

Sincerely,
Ayman Abu-Mostafa
7932 Lampson Ave. #25
Garden Grove, California 92641-4147

Running from Repetition

Dear Mr. Ouverson:

Your readers might be interested in a word that I find quite useful for avoiding repetitive typing at the keyboard. I call this word RUN" (see Figure One), and use it to compile a sequence of characters that acts as though they are input from the keyboard at run time. For example, suppose you frequently type FORGET TASK followed immediately by

: TASK ;

You can use RUN" to capture this typing sequence in a definition:

: RENEW
  RUN" FORGET TASK
  TASK ;

Then you can just type RENEW whenever you want to execute the whole sequence.

I mostly find RUN" useful while editing. First a word FI ("Forth's I") is defined to avoid conflict with the editor's word I (see Figure One). Then you can easily, for example, comment out lines five through ten of the screen being edited by typing:

:: 11 5 DO
RUN" is defined in a simple way that doesn't allow words using RUN" to be nested, but this definition is nevertheless adequate for most uses.

Sincerely yours,
Adin Tevet
P.O. Box 217
44-101 Kfar Sava
Israel

Correction
The preceding issue of Forth Dimensions contained the article "Multiprocessor Forth Kernel" by Bradford J. Rodriguez. Our reproduction of his Figure Three is somewhat unclear—the zeroes represent null pointers and should have been positioned to indicate the inactive queues (i.e., those without any pointer to a task). Our apologies to the author and to any who were perplexed because of this error.

Figure One. Words to compile oft-repeated keyboard sequences.

```
:: (RUN") ( adr cnt )
DUP #TIB !
TIB SWAP ( adr tib cnt ) CMOVE
0 BLK ! 0 >IN ! INTERPRET ;
:: (RUN") [COMPILE] "
COMPILE (RUN") ; IMMEDIATE
:: FI[
ALSO FORTH ]
COMPILE I [ PREVIOUS ] ; IMMEDIATE
:: JUNK-WORD ; \ So FI isn't the last word before using ::
```

```
: (RUN") ( adv cnt )
DUP #TIB !
TIB SWAP ( adv tib cnt ) CMOVE
0 BLK ! 0 >IN ! INTERPRET ;
```
Creating a new Forth standard is a juggling act. The standards committee must balance the desire to bring Forth up to date with current computer technology and the need to protect the investment made in Forth-83. This investment includes the time spent learning Forth-83 and applications developed in Forth-83. There are hard decisions to make and you can’t please everyone. This article describes some of the differences between Forth-83 and ANS Forth (as of July 1989).

The Forth-83 virtual machine was precisely specified as operating on 16-bit data using two’s complement arithmetic and addressing memory as successive eight-bit bytes. This model closely matched the most common computers available in 1983. However, computer technology has advanced since then. 32-bit microprocessors are common, and several Forth-in-hardware systems are available. Unfortunately, these types of machines have difficulty living within the Forth-83 constraints. A major goal of the standard is to allow efficient implementations of ANS Forth on a wide variety of processor architectures.

In parallel with the evolution of computers, Forth has evolved, too. New implementation techniques are constantly emerging. For example, subroutine threaded/native code implementations are now common. ANS Forth will encourage a wide range of implementation options. New Forth language constructs and programming techniques have also been developed. Those that are mature and have become indispensable will be standardized.

Table One [page 35] summarizes additions to the Required Word Set and Table Two shows deletions from the Required Word Set. The remainder of this article discusses some of these changes in detail. Other changes from Forth-83, such as the addition of floating-point and file-extension word sets, are subjects for future articles.

Additions
Many additions to the Required Word Set are minor. For example, the set of two-cell operators has been rounded out by the addition of 2>, 2DROP, 2DUP, 2OVER, 2R>, and 2SWAP. 2!, 28, and 2* are already in most Forth systems and are now required. C:, completes the set of character operators (C@, C!, and C,).

Some of the additions are new capabilities over Forth-83. ANS Forth will allow the construction of string literals and character literals. " (quote) constructs a string literal within a colon definition:

```
: HELLO " hello world" TYPE ;
```

CHAR pushes the first character of the next word in the input stream onto the stack: CHAR A CONSTANT 'A'

```
[CHAR] is like CHAR but it compiles the character as literal:

: Foo ...
     [CHAR] A EMIT ...
```

ANS Forth will have more control flow functionality. RECURSE recursively calls the word that contains the RECURSE (this is called MYSELF on some systems). Forth-83 forbade the use of EXIT within a DO ... LOOP. This was done because there was no portable way to clean the loop control parameters off of the return stack before doing the EXIT. This has been remedied in ANS Forth by the addition of UNLOOP. UNLOOP allows a word to be EXITed from within a DO ... LOOP:

```
DO ...
  IF ...
  UNLOOP
  EXIT THEN
LOOP
```

This solves many sticky control flow problems.

ANS Forth allows programs to explicitly access the Forth interpreter. Many Forth systems have a word called INTERPRET. ANS Forth includes a similar word called EVALUATE that, when passed a string, interprets the string as Forth text. For example,

```
: 2+ " 2 +" EVALUATE ; IMMEDIATE
```

2+, which has been deleted from ANS Forth, could be defined as shown for backwards compatibility. Everywhere a 2+ occurs in subsequently loaded code, the phrase 2 + would be evaluated. This is equivalent to using a text editor to search for all occurrences of 2+ in the source code and replace them with 2 +. EVALUATE is a powerful feature.

A major goal of the ANS Forth effort is to permit both 16-bit and 32-bit Fortths to be standard. This has been accomplished by allowing the size of a data “cell” to be implementation-defined (e.g., 16 bits in Forth-83). However, once this generalization is made, something remarkable happens. In addition to 16-bit and 32-bit processors, a host of other machine architectures, such as 18-, 20-, 24-, or 36-bit processors, are also able to support this more general concept of a cell. Thus, the range of
WE'RE BOOTING UP!

Proceedings of the 1989 Rochester Forth Conference
6 invited papers and 54 presented papers on all aspects of Forth processors, applications and object oriented technology, including:

SwissForth. A Development and Simulation Environment for Industrial and Embedded Controllers
Forth-based Control of an Ion Implanter

: Cellmate/TOOLBOX
Hardware/Software
Workstation/Language DOES>
Automotive/Aerospace
Powertrain/Vehicle
Development/Testing ;

Events and Objects: Industrial Control by Hierarchical Decomposition
Breakthrough in Knowledge Management
An Application Specific Machine Vision System

JFAR Volume 6
Publisher:
Lawrence P. Forsley
Editors:
Dr. S.N. Baranoff, U.S.S.R Editor
Leningrad Institute for Informatika
Dr. J. Basile, Editor-in-Chief
Long Island University
Dr. R. Crawford, U.K. Editor
Microprocessor Engineering Ltd.
Dr. M. Kelly, U.S. Editor
University of Virginia
Dr. H. Nieuwenhuyzen, European Editor
University of Utrecht, The Netherlands

Upcoming Papers:
- 32 Bit Forth Processors
- Forth in the U.S.S.R.
- Object Oriented Extensions

NOW'S THE TIME!

Volume VI Subscriptions

<table>
<thead>
<tr>
<th></th>
<th>Individual</th>
<th>Corporate</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>$50.00</td>
<td>$125.00</td>
</tr>
<tr>
<td>Canada/Mexico</td>
<td>$60.00</td>
<td>$125.00</td>
</tr>
<tr>
<td>Europe/Asia</td>
<td>$70.00</td>
<td>$140.00</td>
</tr>
<tr>
<td>1989 Rochester Conference Proceedings</td>
<td>$25, plus $5 S/H</td>
<td></td>
</tr>
</tbody>
</table>

Send name, full address and phone number. Check or money order in US funds, or, VISA/MC # and exp. date.

To:
Journal of Forth Application and Research
70 Elmwood Avenue
Rochester, NY 14611 USA
(716) 235-0168 • (716) 328-6426 fax

Special Offer! 1989 Proceedings available to Vol. VI subscribers for $20 ($10 savings, for overseas orders add $5 S/H.)

1. A future article will describe the portability features of ANS Forth in more detail.
2. There is not adequate space to list all the merits of POSTPONE here. I presented a paper at the 1989 Rochester Forth Conference that discusses POSTPONE in detail. A copy of that paper may be obtained by sending a self-addressed, stamped envelope to me at Mail Stop 13-S576, Johns Hopkins University, Applied Physics Laboratory, Johns Hopkins Road, Laurel, Maryland 20707.

The Journal of Forth Application and Research

Volume 6

1990

computers that can support standard Forth has been vastly extended at practically no cost. ANS Forth provides operators for portably addressing cells in memory. CELL+ is used to step through arrays of cells in memory and CELLS is used to compute the amount of memory occupied by a given number of cells.

S>D and S>S are used to convert between single- and double-precision numbers. On systems that use two’s complement arithmetic, these words are trivially defined as DUP 0< and DROP respectively. However, use of S>D and S>S clarifies the intent of the code and permits the code to run on systems where the above two’s complement tricks don’t work.

Deletions

Deletions from the Required Word Set must be made cautiously. Adding new features to the language is fine, but deleting features can prevent working Fort83 programs from running on ANS Forth systems. Many apparent deletions in Table Two are merely reorganizations of the word sets. For example, the block words have been moved into a Block Extension Word Set and VOCABULARY has been moved into a Vocabulary Extension Word Set.

Some obsolete Fort83 words have been deleted because they were inefficient or difficult to implement. In most cases, the deleted word has been replaced by one of equal or greater capability. For example, COMP I LE and [COMP I LE] have been replaced by the single word POSTPONE. COMP I LE was the biggest barrier to implementing ANS Forth using subroutine-threaded code, the preferred implementation technique for Forth on Forth chips. Since a standard Forth that wouldn’t run on Forth chips would have been disappointing, POSTPONE was introduced. In all but a few rare cases, POSTPONE may be used instead of COMPILE or [COMP I LE]. For backward compatibility, COMP I LE may be defined as:

Dr. R. Crawford, Editor-in-Chief
Publisher:
Lawrence P. Forsley
Editors:
Dr. S.N. Baranoff, U.S.S.R Editor
Leningrad Institute for Informatika
Dr. J. Basile, Editor-in-Chief
Long Island University
Dr. R. Crawford, U.K. Editor
Microprocessor Engineering Ltd.
Dr. M. Kelly, U.S. Editor
University of Virginia
Dr. H. Nieuwenhuyzen, European Editor
University of Utrecht, The Netherlands

Upcoming Papers:
- 32 Bit Forth Processors
- Forth in the U.S.S.R.
- Object Oriented Extensions

Volume VI Subscriptions

<table>
<thead>
<tr>
<th></th>
<th>Individual</th>
<th>Corporate</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>$50.00</td>
<td>$125.00</td>
</tr>
<tr>
<td>Canada/Mexico</td>
<td>$60.00</td>
<td>$125.00</td>
</tr>
<tr>
<td>Europe/Asia</td>
<td>$70.00</td>
<td>$140.00</td>
</tr>
<tr>
<td>1989 Rochester Conference Proceedings</td>
<td>$25, plus $5 S/H</td>
<td></td>
</tr>
</tbody>
</table>

Send name, full address and phone number. Check or money order in US funds, or, VISA/MC # and exp. date.

To:
Journal of Forth Application and Research
70 Elmwood Avenue
Rochester, NY 14611 USA
(716) 235-0168 • (716) 328-6426 fax

Special Offer! 1989 Proceedings available to Vol. VI subscribers for $20 ($10 savings, for overseas orders add $5 S/H.)

1. A future article will describe the portability features of ANS Forth in more detail.
2. There is not adequate space to list all the merits of POSTPONE here. I presented a paper at the 1989 Rochester Forth Conference that discusses POSTPONE in detail. A copy of that paper may be obtained by sending a self-addressed, stamped envelope to me at Mail Stop 13-S576, Johns Hopkins University, Applied Physics Laboratory, Johns Hopkins Road, Laurel, Maryland 20707.
[COMPILE] may be defined (identically) as:

: [COMPILE]
  POSTPONE POSTPONE
  IMMEDIATE

COMPILE and MOVE have been replaced by a single operator, MOVE. Forth-83 specifies that CMOVE causes patterns to propagate through memory when the source and destination blocks overlap. For example:

CREATE X 10 ALLOT
0 X ! X X 1+
9 CMOVE

fills the array X with zeroes. This means that CMOVE must move one byte at a time. This is inefficient on many machines where multiple bytes can be transferred simultaneously. This diminishes the utility of CMOVE as a block move operator, its primary function. Consequently, MOVE is permitted to move a block of memory as expeditiously as possible. Pattern propagation is easily achieved with C@, C!, and DO ... LOOP.

PICK and ROLL are problematic. They are very inefficient on some architectures and are generally regarded as ugly programming constructs. Therefore, they have been moved from the Required Word Set to the Extension Word Set. Unfortunately, no equivalent functionality, such as local variables, has been added. It is recommended that implementations of ANS Forth provide PICK and ROLL to support old programs and that new programs be written so that they don't rely on PICK or ROLL. At worst, PICK and ROLL could be defined:

: PICK
  ?DUP IF SWAP >R
  1- RECURSE R>
  SWAP ELSE DUP THEN ;

: ROLL
  ?DUP IF SWAP >R
  1- RECURSE R>
  SWAP THEN ;

Summary

ANS Forth is a descendant of Forth-83. Consequently, knowledge gained about Forth-83 may be applied identically to ANS Forth.

(Continued on page 35.)
A random number generator spits out a sequence of numbers, integers or reals, that are randomly distributed according to sundry criteria. No method is known for producing truly random numbers on a digital computer. Instead, sequences of 'pseudo-random' numbers are produced, and these generators are acceptable if they pass sufficiently many of the tests for randomness that any truly random number generator would pass. No truly random number generator is known, so, for convenience, pseudo-random number generators are called simply random number generators.

This paper has two goals. First, it presents an implementation of a particularly simple and easy-to-implement random number generator in the real numbers modulo 1. Second, it describes how a suitably extended standard Forth package and an agreeable text formatter can be used synergistically to produce readable, well-commented code as a 'real-time' endeavor. We have used Knuth's \TeX formatter, a huge program that has a long, steep learning curve. There probably are no \TeX beginners, but \TeX intermediates like this writer can have the pleasure of producing good-looking text with mathematical formulas, program code, tables, and arrays composed and formatted almost automatically. True \TeX experts can work miracles. The actual formatting used to produce the manuscript came from the I\TeX macro package.

The Forth code is no longer organized in screens. Typefaces distinguish the state of the characters. Code to be typed in and compiled is presented in typewriter font. Forth words, lines, and examples entered interactively from the keyboard are shown bold.

Kinds of Generators

Congruential generators

The linear congruential generator is the one likely to be found in most Forth packages. Such a generator produces a sequence of integers \( x_n \) from a 'seed' \( x_0 \), developing the sequence according to the formula:

\[
x_{n+1} = a x_n + c \pmod{M}.
\]

The seeds are supposed to be 'random'

The integers \( a \) and \( c \) are positive and given, and reduction modulo \( M \) returns the remainder between 0 and \( M-1 \) inclusive when \( ax_n + c \) is divided by \( M \). No more than \( M \) different integers can be generated before the sequence begins to repeat. One goal in the design of random number generators is to obtain a long run before repetition begins. The mutual relation of \( a, c, \) and \( M \) is crucial: a few choices produce long sequences of well-distributed integers, while others are miserably bad. Some good choices are as follows:

\[
\begin{array}{cccc}
x_0 & a & c & M \\
256 & 75 & 0 & M_{31} \\
? & 31421 & 6927 & 2^{26} \\
0 & 3141592653 & 2718281829 & 2^{35} \\
\end{array}
\]

(The number \( M_{31} = 2^{31} - 1 \).) The sources for these are respectively [Che85], [Bro87], and [Knu69].

Linear congruential generators in which the modulus \( M \) is a power of two are especially suitable for binary computers, particularly if the exponent can be chosen so that the mod-operation is just the lopping off of a few bytes at the head of the current number. However, these generators are not so suitable for computing real (floating point) random numbers directly. While the integers could be floated, divided by a modulus, and then truncated from the head, there is still the matter of the overhead of the multiplications and divisions, whether the goal is integers or reals.

Fibonacci generators

The overhead problem of the operations can be overcome by using a Fibonacci random number generator, which belongs to the class that obtains the current number as the sum, difference, or product of previously computed members of the sequence. The Fibonacci sequence (published in 1202 by Leonardo of Pisa, called Fibonacci) starts from \( x_0 = 1 \) and \( x_1 = 1 \) and unrolls according to the recurrence \( x_{n+1} = x_n + x_{n-1} \), giving 1, 1, 2, 3, 5, 8, 13, 21, ... . This sequence fails simple tests for randomness. A simple floating-point Fibonacci generator that passes almost all the known tests for randomness is available, namely

\[
x_{k+1} = x_{k+7} - x_{k+5}
\]

(1)

If this generator is implemented in floating point and its output is combined with the output of a suitably designed floating-point congruential generator, the re-
sulting generator passes all known tests for randomness [Kah89]. (We are accepting this assertion by Kahaner, et al. on the strength of their reputations. References to the literature are mostly to inaccessible technical reports. Jansson's book [Jans66] contains only the barest mention of Fibonacci generators, and nothing of the present one. A recent, hard-to-secure Berkeley master's thesis by Akers also treats Fibonacci sequences briefly. Both of these sources contain useful general information on the history and properties of a wide selection of random number generators.)

We will implement the generator (1) in its simplest form, not amalgamating it with a congruential generator. For this purpose, we will have to keep 18 consecutive elements of the sequence on tap. The easiest way to do this is to store them in an 18-element circular array with pointers to the currently called elements. Elements no longer needed will be overwritten. The currently generated number will be written one forward—equivalently, 17 back.

The circular array will have to be seeded with the first 18 elements \( x_0, \ldots, x_p \). These might in fact be poorly chosen. Those readers who have studied the solution of linear differential equations with constant coefficients should be able to see why the general solution of (1) has the form

\[
x_k = \sum_{i=0}^{17} c_i r_i^k,
\]

where the numbers \( r_0, \ldots, r_p \) are the roots, real and complex, of the secular equation

\[
r^{18} + r^{12} - 1 = 0
\]

and the coefficients \( c_0, \ldots, c_{17} \) may be chosen arbitrarily provided that the sum (2) is real-valued.\(^3\) With optimal choices for the initial values \( x_0, \ldots, x_{17} \), the theoretical number of elements generated by (2) before the sequence recycles can be very large: if \( \mu \) is the number of bits (not counting the sign bit) in the mantissa of the floating-point representation, the period can be as large as \( p = (2^{18} - 1)2^{12} \). If \( \mu = 55 \), say, corresponding to a seven-byte mantissa with one sign bit, then \( p = 2^{72} \), almost \( 5 \times 10^{21} \).

**Generator Startup**

In order to implement the generator (1), we will have to determine the mantissa width \( \mu \) and then seed the circular array that holds previously generated \( x \).

**Mantissa width**

We have defined \( \mu \), the mantissa width, to be the number of bits—not counting the sign bit—in the mantissa of the floating-point representation. A user may have foreknowledge of \( \mu \) for the particular floating-point enhancement that she is using. We will present a Forth word \( \text{f.mu} \) that automatically determines \( \mu \) when the random number generator is loaded. For maximum portability, \( \text{f.mu} \) is written in only highest-level floating-point words that assume absolutely no special knowledge of the structure of the floating-point number. Of course, we assume that the mantissa is stored somewhere as a string of bits, although that string need not even be connected in the memory. The sign bit will not be counted. Certain floating-point systems use a special normalization that counts on the mantissa always starting with the (binary) digit 1, so that the left-most digit is carried as a virtual digit in order to gain an extra binary digit's worth of precision. We will not worry about such special systems or, what is operationally equivalent, we will treat such systems as if they were non-virtual. If the width \( \mu \) were counted one too large, the only loss would be a tiny waste of processing time in finding the circular array. If the count were one too small, the maximum cycle length would be cut to half the theoretical maximum, not likely to be a great tragedy.

The technique is simple. We imagine the mantissa as carried in a linear register of \( \mu \) bits initialized to the string 1000.... Then we insert digits one from the left end, pushing the previous contents to the right. Each time we push in a one, we compare the previous contents with the new contents. When there is agreement, the register has been filled with a string of ones whose number is the length of the register. First we will need a bin for counters:

```forth
variable F.MU
```

Now the counting word itself is straightforward. The comparisons are made while keeping both the old and new contents on the (floating-point) stack.

```forth
: F.MU=? ( -- )
  Oe \ empty register
  1e \ first push
  1 f.mu ! \ count that push
  begin
    fswap fover ( old new )
    f= not \ different?
  while
    fdup
    2e f/ 1e f+(newold newnew)
    1 f.mu +! \ 1 wider
  repeat \ 'til the same
  fdrop \ clean stack
```

The word \( \text{f.mu}=? \) should run at loading for initializing.

I tested this word on F-PC, the super-enhanced child of F83 developed by Tom Zimmer and his co-conspirators. Using the hardware floating-point enhancement contained in the file HFLOAT.SEQ, I entered

```forth
f.mu=? f.mu
```

and found the mantissa width to be 55. From this I conclude that the mantissa occupies seven bytes, of which one bit is the sign.

**The circular array**

We want to create a ‘circular’ array of 18 floating-point numbers. Of course, we really create a linear array, but imagine that fetches are to wrap around: positive offsets that go past the last element continue onward from the initial entry, with a similar treatment of negative offsets. The actual

7The equation \( r^{18} + r^{12} - 1 = 0 \) has eighteen roots, just two of which are real, namely

\[
\rho = \left\{ \frac{2}{3} \cosh \frac{1}{3} \cosh^{-1} \left( \frac{25}{2} \right) - 1 \right\}^{1/6} \approx 0.954214685
\]

and -\( \rho \) together with 14 non-real roots. Of these last, four are arranged with \( \rho \) and -\( \rho \) as vertices of a regular hexagon in the complex plane, while the remaining 12 lie six and six on two complex-conjugate regular hexagons with all 12 vertices on the circle of radius \( \rho^{1/6} \) centered at the origin.

8Why the exponent \( \mu - 1 \)? If the last bit inserted is a one, then the corresponding real number will lie between one and two, so it will be reduced modulo 1 by subtracting one from it.
of 18 elements and only three out of 17 because we will work solely with an array here for a generic array with generic words, setting words themselves. There is no call wrapping around is carried out by the off-

First we introduce some convenient arithmetic words.

: OFFSET.BY.5
  ( n -- n-5 mod 18 )
  5-
  dup 0<
  if ( n-5 < 0 )
  18+ \ slide to pos've
  then ;

To offset back by 17, we can as well offset forward by one. For 17 of 18 cases, k + 1 < 18 if k < 18.

: OFFSET.BY.17
  ( n -- n-17 mod 18 )
  1+
  dup 17>

Now come the actual offsetting words. The first one offsets the index by -5. If k is the current index, then the offset index is, of course, just k - 5. The circular array will be indexed from zero to 17. Therefore, k - 5 is just k - 5, provided that k - 5 ≥ 0. If k - 5 < 0, then it is to be replaced by its (floored) residue modulo 18. Ordinarily we would do this directly by the mod operation, the job for which it was designed. But now we have special information: if k - 5 < 0, then k - 5 ≥ -17. Thus, we will obtain the correct (floored) residue simply by adding 18, an operation much faster than a full-blown division. The first step in the offsetting will be to check the nominal offset index for its location. We have the option of adding 13: 13(-5) = 18. But n ≥ 5 modulo 18 about 75 percent of the time. We observe that 0< is an intrinsically fast operation in most Forth systems, while 17 > is not.

if ( n = 17 )
  drop 0 \ n-17 = 0 mod 18
then ;

Now for the array. Is it a circular array, a clock, or a pie chart with 18 wedges? In honor of Leonardo, we use the last image.

create PISA.PIE ( 18 slices )
18 f#bytes * allot

Then we can locate the element offset n units into the pie:

: PISA ( n -- address )
f#bytes * pisa.pie + ;

Seeding the Array

Now that we have the array, the pisa.pie, we must initialize it—seed it—so that the random number generator can produce the sequence of random reals modulo 1. The seeds are again supposed to be ‘random’ reals, but here the randomness has a specific interpretation. A choice of the coefficients c_0, ..., c_{17} in the formula (2) will determine the initial values x_0, ..., x_{17}. Conversely, a little algebra (involving Vandermonde’s determinant) shows that the first 18 xs also uniquely determine the 18 cs. The randomness sought here is really genericity, resulting in full dimensionality. The cs should be such that the set of all x modulo 1 generated by (2) is ‘18-dimen-

All integers between -32768 and -1 inclusive and 0 and 32767 inclusive. These sets have 32768 elements—the same number—so that we can generate a two-state random sequence by generating a random sequence of Forth signed integers and assigning each to one or the other state, according to its sign.

The simple generator must be seeded. It is desirable for debugging purposes to be able to rerun the same random sequence. Hence, the reseeding of the generator must be voluntary, and the last seed will be kept in a storage bin.

variable SEED

We adapt the generator presented in Starting Forth [Bro87].

: SF.RAND ( -- n )
  seed @
  31421 * 6927 +
  dup seed ! ;

The word sf.rand leaves a Forth signed integer on the stack, and it updates seed. To get a fresh seed in an unbiased way, we obtain a double integer from the system clock and store its least significant digits into seed.

: NEW.SEED ( -- )
  gettime
  swap drop
  seed ! ;

The seed is planted at loading:

new.seed

Now we encounter real numbers. The floating-point implementations in F-PC use a floating-point stack separate from the parameter stack. We use an ad hoc stack notation, with a vertical bar separating the two stacks: parameter | floating.

The next word pushes a ‘random’ integer onto the parameter stack, classifies it according to sign, then pushes the real representing the proper state onto the floating-point stack.

\footnote{Sometimes this scheme will fail by producing ‘unrandom’ seeds. No seeding scheme will always succeed—if one did, we would use that successful scheme as our random number generator and be done. Every calculation employing random numbers should always be run several times and the results examined for a bias caused by a breakdown of the random number generator. See the critique of generators in commercial offerings by [Mod87].}

\footnote{In the 'ancient' days, calculators by hand would glance up at the second hand on the wall clock or decide in which part of the room a pesky fly was at the moment, using 'fate' as the random number generator.}
To get a real modulo 1, we first push the real zero onto the floating-point stack in order to force the exponent to zero. Then we iteratively push zeros and ones into the mantissa from the left. If the last bit pushed is a one, the resulting number lies in the real interval [1,2), so subtracting one correctly reduces it modulo 1.

The word `one.random.real.mod.1` pushes a random real onto the floating-point stack. The next word generates 18 random reals modulo 1 and stores them into the array `pisa pie` as they are generated.

The Generator

The index of the current (last generated) Fibonacci random real number modulo 1 will be kept in the variable `SUBSCRIPT`.

We will make two initializing words available. There is no harm in imagining the seeding xs to have negative indices: $x_{-1}$, $x_{-2}$, ..., $x_0$. Then the calculation begins with the latest subscript set to zero.

**FORML Conference** that the best time to write documentation is before writing the code. Was he talking with tongue-in-cheek? I do not know, but in fact it has always been my practice to write the documentation/commentary/article before I start in on the FORTH code. Of course, I write it out in my mind rather than on paper, postponing until the last possible moment the inevitable time when I must cope with bugs and crashes.

Neither before nor after is the best time to document. The best time is during:

The two ways available up to a short time ago were less than satisfactory: parentheses and backslashes fitted in awkwardly at best with the block/screen source code structure. Shadow screens only mirrored the blocks, being sometimes too small and often too large for the relevant commentary. In one of his written contributions to the 1988 FORML Conference, Dr. Ting laid out the flaws convincingly, arguing for source code in text-file form. But it is not just documentation that is bettered in text-file arrangement; all forms of output are made easier and better. This is why: formatting the file for the compiler and the printer can be done simultaneously.

Here is how I do this dual job at once, knowing that I want to pass my file through the FORTH compiler and, afterwards, send the same file through the TeX formatter and then to the printer. The first requirement is a FORTH implementation that can compile source code from text files. I am personally acquainted with two such. Laboratory Microsystems, Inc. sells UFORTH, a powerful and fast implementation of FORTH-83 for MS-DOS computers. Tom Zimmer and his collaborators have produced F-PC, a large and richly endowed public-domain implementation of FORTH-83. Both of these implementations would have served my purposes, but I used F-PC for an essentially trivial reason: it came ready-supplied with two marvelous words that are powerful extensions of the commenting words (\...) and \ which are restricted in scope to one line of source code. F-PC contains comment: and comment; to allow comments extending over arbitrarily many lines and CR/LF delimiters. The F-PC compiler treats any and all text between successive occurrences of comment: and comment; as a comment and passes over this text.

---

allows easy production of comments with a word processor, whereas the usual commentary requires hand insertions of at the beginning of every line of commentary. That is especially irritating when the word processor has run a line out to the right margin before wrapping to the next line, and it is tedious to insert or delete partial lines of commentary upon afterthought.

On the other hand, the \TeXX formatter operates with embedded commands, those commands (almost always) consisting of space-free strings of ASCII characters beginning with the backslash \. Therefore, comment: and comment; cannot be interpreted by \TeXX as its own commands. I saw two ways to handle this problem. The first, most attractive possibility was that comment: and comment; could serve dual roles. Of course, they would have to be prefixed by \ in order to work in \TeXX. Forth itself would have a new word added to its dictionary: \comment: could be cribbed from the listing in Ting’s F-PC Technical Reference Manual, with the only change being replacement of the delimiter comment; by \comment:. The backslashesd words would work the expected way with Forth compilation. For \TeXX purposes, they should work in the opposite way, with \comment: turning on the special typeface for showing Forth code and \comment: turning it off. That could be done automatically by defining \comment: and \comment; as \TeXX macro commands in the preamble to the document. I tried this method, but didn’t like it because it did not really fit into the way I usually work at the word processor.

Like Forth writing, \TeXX composition is best done in small morsels, with the latest batch put through the \TeXX compiler to catch the bugs. \TeXX does not crash as Forth does, but even a \TeXXpert will sometimes write formatting that sinks under an overload of error messages. Writing and testing Forth code and preparing a manuscript for \TeXX formatting are not all that different. F-PC comes with Tom Zimmer’s SED editor, written to mimic WordStar’s command structure, but with oodles of added conveniences to ease the work of composing and compiling Forth code. It is powerful, but it is WordStar, which I last used four years and several other editors ago. I have been using BRIEF for \TeXX and now use it for Forth code writing. Any word processor will work to write source code for F-PC provided that it can be configured to represent tabs as spaces rather than Ctrl-I (ASCII 9, which F-PC represents as the little ‘circle’ in the IBM graphics set) and to terminate files without Ctrl-Z. BRIEF allows me to have many files on the ‘desktop’ at once, and I usually have two or three open in windows. I set up one ‘text’ file to contain the debugged code and the inter-woven commentary. A second window contains another ‘text’ file that will hold only Forth code and will be used for debugging. As I verify the code in the second file, I copy it over to the first file by using the cut-and-paste facilities of the word processor, squeezing it between \TeXX font-formating commands.9 The result is a file always ready for \TeXX formatting and subsequent printing. Of course, I am also producing the second file of Forth code only, but I can compile the extended file after first using the global search-and-destroy commands to replace embedded commands of the \TeXX ilk by comment: and comment; and making a small, technically required alteration to the first line of the \TeXX preamble.

I’ve also chosen to increase readability of the text by producing the manuscript in a two-column format, which restricts each line to approximately 40 characters. This would seem at first encounter to place a restriction on the in-code commenting. However, I concur with Dr. Ting’s point of view, which is that the current accepted Forth commenting style is a by-product of the 16-line x 64-column matrix of the Forth screen. Without the threat of line 16, there is no reason to put more than three or four words on a line, so that the rhythm that was before expressed by subtle insertions of spaces is now rendered clear by line breaks.10 Neither is there any longer a reason for long parenthetical comments or backslashed lines. Long comments can be moved out into the surrounding text. Indeed, the code can be interrupted for as many lines as is desired of comments.

Different word processors and text formatters may have individual idiosyncrasies that require a little care. For example, \TeXX uses the underscore (e.g., \_\_\_\_\_\_\_) as a control character, so it is best not to use it in Forth words. I like the underscore for readability, but decided to use the period as a separator. In the same way, \_ has a special meaning to \TeXX, but I found that with care I could avoid using it where \TeXX could see it.

In closing, I must declare that the scheme explained in this section is not claimed to be all-embracing or universally feasible. I would not bother to use it if I were programming for an application that was to be contained in a very small portion of memory—embedded systems, for example. It is of no use in Forth systems that are only block oriented. It produces source files cluttered with formatting commands (no problem to an adept but offputting to a casual browser) that must be printed out or viewed in formatted form for full benefit. But, given a Forth implementation that can compile text files, I think it is a programming synthesis that might help to dispel Forth’s reputation as a ‘write once, read never’ language.

References

Nathaniel Grossman is professor of mathematics at the University of California, Los Angeles.

*And it can restore all of those files to the ‘desktop’ at the start of a new session.
*The cycle of loading and unloading is shortened by running BRIEF as a daughter in the DOS shell that F-PC creates with its sys command.
*Dr. Ting already has exploited this ‘vertical’ format in his F-PC Technical Reference Manual.
MEET THAT DEADLINE !!!

- Use subroutine libraries written for other languages! More efficiently!
- Combine raw power of extensible languages with convenience of carefully implemented functions!
- Yes, it is faster than optimized C!
- Compile 40,000 lines per minute!
- Stay totally interactive, even while compiling!
- Program at any level of abstraction from machine code thru application specific language with equal ease and efficiency!
- Alter routines without recompiling!
- Use source code for 2500 functions!
- Use data structures, control structures, and interface protocols from any other language!
- Implement borrowed feature, often more efficiently than in the source!
- Use an architecture that supports small programs or full megabyte ones with a single version!
- Forget chaotic syntax requirements!
- Outperform good programmers stuck using conventional languages! (But only until they also switch.)

WAKE UP !!!

Forth is no longer a language that tempts programmers with "great expectations", then frustrates them with the need to reinvent simple tools expected in any commercial language.

HS/FORTH Meets Your Needs!

Don't judge Forth by public domain products or ones from vendors primarily interested in consulting - they profit from not providing needed tools! Public domain versions are cheap - if your time is worthless. Useful in learning Forth's basics, they fail to show its true potential. Not to mention being s-l-o-w.

We don't shortchange you with promises. We provide implemented functions to help you complete your application quickly. And we ask you not to shortchange us by trying to save a few bucks using inadequate public domain or pirate versions. We worked hard coming up with the ideas that you now see sprouting up in other Forths. We won't throw in the towel, but the drain on resources delays the introduction of even better tools. Don't kid yourself, you are not just another drop in the bucket, your personal decision really does matter. In return, we'll provide you with the best tools money can buy.

The only limit with Forth is your own imagination!

You can't add extensibility to fossilized compilers. You are at the mercy of that language's vendor. You can easily add features from other languages to HS/FORTH. And using our automatic optimizer or learning a very little bit of assembly language makes your addition zip along as well as in the parent language.

Speaking of assembly language, learning it in a supportive Forth environment turns the learning curve into a light speed escalator. People who failed previous attempts to use assembly language, conquer it in a few hours or days using HS/FORTH.

HS/FORTH runs under MSDOS or PC DOS, or from ROM. Each level includes all features of lower ones. Level upgrades: $25. plus price difference between levels. Sources code is in ordinary ASCII text files.

All HS/FORTH systems support full megabyte or larger programs & data, and run faster than any 64k limited ones even without automatic optimization - which accepts almost anything and accelerates to near assembly language speed. Optimizer, assembler, and tools can load transiently. Resize segments, redefine words, eliminate headers without recompiling. Compile 79 and 83 Standard plus F83 programs.

STUDENT LEVEL $145.
- text & scaled/ clipped graphics in bit blit windows, mono, cga, egavga, fast ellipses, splines, bezier curves, arcs, fills, turtles; powerful parsing, formatting, file and device I/O; shells; interrupt handlers; call high level Forth from interrupts; single step trace, decomposer, music; compile 40,000 lines per minute, stacks; file search paths; formats into strings.

PERSONAL LEVEL $245.
- hardware floating point, trig, transcendental, 18 digit integer & scaled integer math; vars: A B * IS C compiles to 4 words, 1.4 dimension var arrays; automatic optimizer-machine code speed.

PROFESSIONAL LEVEL $395.
- full machine code speed; data structures for all data types from simple thru complex 4D var arrays - operations thru complex hyperbolics; turnkey, seal; interactive dynamic linker for foreign subroutine libraries; roundup & interrupt driven multitaskers; dynamic string manager; file blocks, sector mapped blocks; & 68k assemblers.

PRODUCTION LEVEL $495.
- Metacomplier: DOR/ROM/direct/indirect; threaded systems start at 200 bytes, Forth cores at 2 kbytes; C data structures & struct+ compiler; TurboWindow-C MetaGraphics library; 200 graphic/window functions, PostScript style line attributes & fonts, viewports.

PROFESSIONAL and PRODUCTION LEVEL EXTENSIONS:

- FOOPS with multiple inheritance $75.
- 2B6FORTH or 3B6FORTH $295.
  16 Megabyte physical address space or gigabyte virtual for programs and data; DOS & BIOS fully and freely available; 32 bit address/operand range with 386.
- RETRIEVE for HS/FORTH (Novell) $199.
- ROMULUS HS/FORTH from ROM $95.
- FFORTHTRAN translator/mathpak $75.
  Compile Fortran subroutines! Formulas, logic, do loops, arrays; matrix math, FFT, linear equations, random numbers.

HARVARD SOFTWORKS
NUMBER ONE IN FORTH INNOVATION
(513) 748-0390 P.O. Box 69, Springboro, OH 45066
A linear quadratic regulator (LQR) is implemented using Forth on a fourth-order ball-balancing system in the laboratory. The control law is implemented on an IBM PC, as well as on a Motorola MC68HC11 board, to test the feasibility of having a standalone system.

Introduction
Optimal control is a branch of modern control theory. Since 1960, it has been used extensively by control engineers in various areas such as the process industry, the space program, and the defense industry. The LQR is the most commonly used form of optimal controller where the control law is obtained by minimizing a quadratic cost functional. The resulting control law is of the form $u(t) = -Kx(t)$, where the $K$'s are the gains and the $x_i$'s are the system state variables.

Assembly language is invariably preferred for real-time digital implementation, due to memory and speed constraints. However, with speed close to that of assembly language, Forth—being a higher-level language—is an attractive alternative.

The ball-balancing system shown in Figure One, an inherently unstable fourth-order system, is used to demonstrate the feasibility of using Forth in optimal control. A Forth program (600 bytes of code) was used to implement an LQR for this system with an IBM PC and a 12-bit Tecmar data acquisition board. A sampling time of eight msec. could be achieved. Another version was successfully tested using the Motorola MC68HC11 with MaxForth embedded in the ROM. This experiment was conducted as a part of a class project in the course “Design of Embedded Software Computer Systems,” taught by Professor Richard Haskell.

Optimal Control
Consider a linear system, given in state-space form as:

$$\dot{x}(t) = Ax(t) + Bu(t),$$

$$y(t) = Cx(t),$$

where, $x(t) \in \mathbb{R}^n$ is the state vector, $u(t) \in \mathbb{R}^m$ is the input vector, $y(t) \in \mathbb{R}^p$ is the output vector, and $A, B,$ and $C$ are real matrices of compatible dimensions. This system can be controlled if the pair $[A, B]$ is controllable, i.e.,

$$\text{rank}[B, AB, A^2B, A^3B, \ldots, A^{n-1}B] = n.$$ (2)

To design an LQR a cost function $J$, of the form,

$$J = \int [x(t)^TQx(t) + u(t)^TRu(t)]dt$$

is minimized, where $Q \in \mathbb{R}^{n \times n}$ is a positive semi-definite matrix, $R \in \mathbb{R}^{m \times m}$ is a positive definite matrix. The state feedback law which minimizes $J$ is given by [1],

$$u_{opt}(t) = -R^{-1}B^TKx(t).$$ (4)

where $K$ is the positive definite solution of the algebraic Riccati equation [1],

$$KA + A^T K - KBR^{-1}BK - Q = 0.$$ (5)

It can be seen from (4) that we need all the states for the implementation of this control law. If $C$ is a nonsingular matrix, then $x(t)$ can be obtained directly from (1) as $x(t) = [C^T]^{-1}y(t)$. If all the states cannot be measured directly, then the nonmeasurable states can be estimated using a Luenberger observer [2] or Kalman filter [3], provided the pair $[C, A]$ is observable, i.e.,

$$\text{rank}[C^T A^T C^T (A^T)^2 \ldots (A^T)^{n-1}] = n.$$ (6)

We have implemented an LQR for a ball-balancing system [4] using an IBM PC as well as a Motorola MC68HC11 microprocessor. The system, as shown in Figure One, consists of two parallel tracks 1.1 m long. A carriage having a pair of arcs with an arc radius of 0.25 m and subtending an angle of 0.28 rad at the center, slides on top of the tracks. A metal ball rolls on top of the arc. The system in state-space form is given by the formula below

$$A = Ax(t) + Bu(t),$$ (7)
where $\mathbf{x} = [x_1, x_2, x_3, x_4] = [z, z, \dot{z}, \dot{\Theta}]$,

- $z(t)$ carriage position,
- $\dot{z}(t)$ carriage linear velocity,
- $\Theta(t)$ ball position,
- $\dot{\Theta}(t)$ ball angular velocity.

The ball angular velocity $\dot{\Theta}(t)$ cannot be measured, hence it is estimated using the ball position data. The system is unstable and the eigenvalues of $A$ are given by $0, 4.85, -4.97, -3.76$. Consider the quadratic performance measure

$$J = \int\left[10x_1^2(t) + 0.1x_2^2(t) + u^2(t)\right]dt.$$  \hspace{1cm} (8)

The optimal control which minimizes $J$ is given by

$$u_{opt}(t) = 1.8x_1(t) - 3.56x_2(t) + 12x_3(t) + x_4(t).$$ \hspace{1cm} (9)

With the sensor calibration constants, this equation for the IBM implementation becomes,

$$u_{opt}(t) = 0.2161x_1(t) - 0.4398x_2(t) + 0.59x_3(t) + x_4(t).$$ \hspace{1cm} (10)

where, $x_4(t)$ is numerically calculated as

$$x_4(t) = 0.4311x_{old} + 6.3324(x_1(t) - x_{old}).$$ \hspace{1cm} (11)

where $x_{old} = x_1(t-\Delta)$, $\Delta$ being the sampling interval.

For the MC68HC11 implementation, additional scaling needs to be done to account for the limitations of the analog-to-digital device and equation (9) becomes,

$$u_{opt}(t) = 0.4237x_1(t) - 1.7276x_2(t) + 0.6941x_3(t) + x_4(t).$$ \hspace{1cm} (12)

where $x_4(t)$ is calculated as,

$$x_4(t) = 0.5384x_{old} + 8.008(x_1(t) - x_{old}).$$ \hspace{1cm} (13)

Implementation of this control law is explained in the next section.

---

**Statement of Ownership, Management and Circulation**

1) Title of Publication: *Forth Dimensions*
   Publication Number: U.S.P.S. 002-191
2) Date of Filing: 9/19/89
3) Frequency of Issue: Bi-monthly
   No. of issues published annually: 6
   Annual subscription price: $24/36
4) Location of known office of publication: 1330 S. Bascom Ave., Suite D, San Jose, Santa Clara County, California 95128-4502
5) Location of headquarters or general business offices of the publisher: Same as above
6) Publisher: Forth Interest Group, P.O. Box 8231, San Jose, California 95155
   Editor: Martín Ouverson, Same as above
   Business Manager: Georgiana F. Shepherd, Same as above
7) Owner: Forth Interest Group, Same as above
8) Known bondholders, mortgages, and other security holders owning or holding 1% or more total amount of bonds, mortgages, and other securities: None
9) The purpose, function, and non-profit status of this organization and the exempt status for Federal Income Tax purposes have not changed during the preceding 12 months.
10) Extent and nature of circulation

<table>
<thead>
<tr>
<th>Average No. copies/issue during preceding 12 months</th>
<th>Actual No. copies of single issue nearest to filing date</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Total no. copies printed: 2750</td>
<td>2500</td>
</tr>
<tr>
<td>B. Paid/requested circulation:</td>
<td></td>
</tr>
<tr>
<td>1. Sales: 0</td>
<td>0</td>
</tr>
<tr>
<td>2. Mail subscription: 2187</td>
<td>1918</td>
</tr>
<tr>
<td>C. Total paid/requested circulation: 2187</td>
<td>1918</td>
</tr>
<tr>
<td>D. Free distribution by mail, carrier or other means: samples, complimentary and other free copies: 47</td>
<td>45</td>
</tr>
<tr>
<td>E. Total distribution: 2234</td>
<td>1963</td>
</tr>
<tr>
<td>F. Copies not distributed:</td>
<td></td>
</tr>
<tr>
<td>1. Office use, left over, unaccounted, spoiled after printing: 516</td>
<td>537</td>
</tr>
<tr>
<td>2. Return from news agents: 0</td>
<td>0</td>
</tr>
<tr>
<td>G. TOTAL: 2750</td>
<td>2500</td>
</tr>
</tbody>
</table>
11. I certify that the statements made by me above are correct and complete.
   /s/ Georgiana F. Shepherd
Figure One. Schematic diagram of the ball-balancing system.

Forth Implementation
The control law given by (10) was implemented in F83 on an IBM PC. The Tecmar board used for data acquisition is configured to have its memory map in a different segment from that used by the F83 Forth system. Since the version of F83 used does not have instructions to store and fetch addresses outside its 64K byte segment, code words !BUFF, @BUFF, C!BUFF, and C@BUFF are written to work in a similar fashion to !, @, c!, and C@, respectively, to access these addresses. We need to set only the base address, which is $A000 in the IBM implementation. These words are given in Screens 1–2 in Figure Two. For example, 725 C@BUFF means fetch byte at address $A0725 = A000:0725.

In the word ADC in Screen 4 of Figure Two, the channel number is sent to address $A0725 and address $A0726 is set to zero to start the analog-to-digital conversion. The word DAC is used for digital-to-analog conversion where addresses $A0721 + 2*ch and $A0720 + 2*ch are stored with the high byte and the low byte of the output data. The word U.CAL in Screen 5 of Figure Two is the scaled calculation of (10). The initial control input in optimal controllers is usually of high magnitude. The word OPT is used to avoid saturation of the d.c. servo motor amplifier, where the output is scaled to 0.3 of its value whenever U.CAL exceeds ±2V. The sampling time is measured to be eight msec.

The control law (12) was implemented on a Motorola MC68HC11 in order to have a standalone system. The listing for this MaxForth program is given in Figure Three. The word TSTH.CCF in Screen 62 checks the completion of the analog-to-digital conversion. The words X1.CAL, X2.CAL, X3.CAL, and X4.CAL in Screen 64 are the scaled calculations of the states $x_1, x_2, x_3,$ and $x_4$, respectively. In order to ensure faster calculation, the calculation is approximated by arithmetic shifts rather than multiplications. For example, in the calculation of X2.CAL in Screen 64,

$$\text{DUP } 2/ \text{ DUP } 2/ \text{ DUP } 2/ \ 2/ 2/ - + +$$

is equivalent to

$$(1 + (0.5 + (0.25 - 0.03125)) \times 2 = 1.71875 \times 2,$$

which is an approximation for 1.7276X2 in (12).

Conclusion
The response of the system is excellent in both implementations. Setting time for the ball on top of the arc is longer with the MC68HC11 due to the lower precision of the eight-bit A/D compared with the 12-bit A/D on a PC.

(Text continued on page 33.)
Figure Two. Forth words used to balance the ball using F83 with an IBM PC and Tecmar data acquisition board.
Figure Three. Forth words used to balance the ball using MaxForth on a 68HC11.
INCREASE MEMORY FOR THE TI 99/4A

HOWARD H. ROGERS - TORRANCE, CALIFORNIA

The amount of random-access memory left in the TI 99/4A after loading TI-Forth—a
extension of fig-FORTH—is 16K without the editor, but 13K with it. However, most
users load additional TI-Forth code, leaving as little as 6K for use. This paper
Discusses a practical method of increasing that memory by over 8K of RAM,
primarily for use with arrays.

Source of Additional RAM
There is 16K of RAM associated with the video display processor (VDP), an
unused 8K of which is available in all modes (text, graphics, and multi-color)
except Graphics2, a bit-mapped mode. Since Forth normally runs in text mode, no
interferences result from using this memory.

It should be pointed out that VDP memory is accessed byte-by-byte through a
memory-mapped port, and is not in the processor’s address space. Accessing this
memory is done in serial fashion, which is significantly slower than accessing pro-
cessor (CPU) RAM. Forth runs in the CPU RAM obtained from a 32K memory expan-
sion card (required to use Forth on the 99/4A).

System Synonyms
TI-Forth provides four words, summarized below, to access VDP RAM:

VSBW  Writes one byte from the stack to a VDP address.

VMBW  Writes multiple bytes from a CPU address to a VDP address.

VSBR  Reads one byte from a VDP address.

VMBR  Reads multiple bytes from a VDP address to a CPU address.

These words provide the basis for the definitions presented in this paper. Arrays
can be initialized by VFFILL, the equivalent of FILL in fig-FORTH.

Screens
The intent of the code shown in the screens is to provide words analogous to
standard Forth words, but which use VDP RAM instead of CPU RAM. The names
are basically the same as those for standard Forth definitions, but with the letter v
before the name.

Screen 110—This screen provides the equivalents of !, C!, ALLOT, and VARI-
ABLE. Since little memory is used by constants, a VDP equivalent of CONSTANT
was considered unnecessary.

Screen 111—The equivalents of @, C@, (comma), C,, and !,+ are shown. The
words equivalent to (comma) increment the VDP pointer VPTR, which serves a
function similar to HERE. RESETV was defined to allow recovery of VDP RAM
prior to using FORGET with a VARRAY, since FORGET alone does not affect

(Continued on page 30.)

Figure One. Representative screen dump of VDP RAM.

Table 1.

<table>
<thead>
<tr>
<th>Addr</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>1400</td>
<td></td>
</tr>
<tr>
<td>1408</td>
<td></td>
</tr>
<tr>
<td>1410</td>
<td></td>
</tr>
<tr>
<td>1418</td>
<td></td>
</tr>
<tr>
<td>1420</td>
<td></td>
</tr>
<tr>
<td>1428</td>
<td></td>
</tr>
<tr>
<td>1430</td>
<td></td>
</tr>
<tr>
<td>1438</td>
<td></td>
</tr>
<tr>
<td>1440</td>
<td></td>
</tr>
<tr>
<td>1448</td>
<td></td>
</tr>
<tr>
<td>1450</td>
<td></td>
</tr>
<tr>
<td>1458</td>
<td></td>
</tr>
<tr>
<td>1460</td>
<td></td>
</tr>
<tr>
<td>1468</td>
<td></td>
</tr>
<tr>
<td>1470</td>
<td></td>
</tr>
<tr>
<td>1478</td>
<td></td>
</tr>
<tr>
<td>1480</td>
<td></td>
</tr>
<tr>
<td>1488</td>
<td></td>
</tr>
<tr>
<td>1490</td>
<td></td>
</tr>
<tr>
<td>1498</td>
<td></td>
</tr>
</tbody>
</table>

Figure One. Representative screen dump of VDP RAM.

0 ( FIG-Forth: VDP arrays and variablesSCR#1 HHR 1/13/87 )
1
2 BASE->R HEX ( 8664 bytes of VDP RAM available )
3 1400 VARIABLE VPTR ( Usable VDP RAM starts at hex 1400 )
4
5 : VALLOT ( n --- ) VPTR +! ; ( Advances VDP RAM pointer )
6
7 : VC! ( b vaddr --- ) VSBW ; ( More appropriate name )
8
9 : V! ( n vaddr --- ) ( 2 VMBW transfers a word from the top )
10 SP@ 2+ SWAP 2 VMBW DROP ; ( of the stack to VDP RAM )
11
12 ( Address of named variable is stored as a constant )
13 : VVARIABLE ( n --- ) VPTR @ ( =CELLS forces pointer even )
14 =CELLS DUP DUP 2+ VPTR ; ( Pointer VPTR is incremented )
15 CONSTANT V! ; --&gt; ( Store value in variable )

SCR #110

SCR #111

SCR #112

0 ( Alternative V! definitionsSCR#2 HHR 1/13/87 )
1
2 VC@ ( vaddr --- n ) VSBR ; ( More appropriate name )
3
4 : V@ ( vaddr --- n ) ( Analogous to @ )
5 0 SWAP SP@ 2+ 2 VMBR ;
6 ( VMBR transfers a word from VDP RAM to the top of the stack )
7
8 : VC, ( c --- ) VPTR @ VSBW 1 VPTR +! ;
9 : V, ( n --- ) VPTR @ V! 2 VPTR +! ;
10 : V+! ( n addr --- ) DUP V@ ROT + SWAP V! ;
11
12 : RESETV ( --- &lt;name&gt; ) ( Recovers VDP RAM— resets pointer )
13 [COMPILE] ' CFA EXECUTE VPTR ! ;
14
15 R-&gt;BASE

SCR #112

0 ( Alternative V! definitions HHR 1/13/87 )
1
2 HEX ( All use VSBW, a single byte transfer from the stack )
3
4 : V! ( n vaddr --- ) DUP ROT
5 DUP SWPB FF AND ROT VSBW ( SWPB swaps bytes conveniently )
6 SWAP 1+ VSBW ;
7
8 : V! ( n vaddr --- )
9 DUP ROT SP@ CE ROT VSBW ( Uses CE instead of SWPB to )
10 SWAP 1+ VSBW ; ( select correct byte )
11
12 : V! ( n vaddr --- ) SWAP PAD ! ( Similar to above but )
13 DUP PAD CE SWAP VSBW ( uses PAD for storage )
14 PAD 1+ CE SWAP 1+ VSBW ;
15
SCR #113
0 ( Alternative V@ definitions  HHR 1/13/87 )
1
2 HEX ( All use VSBR, a single byte transfer to the stack )
3 ( All use + or OR to create a Word from two bytes )
4
5 : V@ ( vaddr --- n ) DUP VSBR
6 8 SLA SWAP 1+ VSBR + ; ( 8 SLA shifts left 1 byte )
7
8 : V@ ( vaddr --- n ) DUP VSBR ( OR is used as an )
9 8 SLA SWAP 1+ VSBR OR ; ( alternative to + )
10
11 : V@ ( vaddr --- n ) DUP
12 VSBR 100 * SWAP ( * in place of left shift of 1 byte )
13 1+ VSBR OR ;
14
15

SCR #114
0 ( VDUMP SCR# 1  HHR 1/15/87 )
1 0 VARIABLE CNTR  0 VARIABLE TEMP
2
3 : PRT 0 <# #### #> TYPE ; ( VDP RAM addr stored at PAD )
4 ( Lines of data stored at PAD + 2 to PAD + 9 temporarily )
5
6 : GETDATA PAD @ PAD 2+ 8 VMBR ;
7 ( Transfers data from VDP addr to PAD + 2 )
8
9 : STOP ?KEY DUP 2 = ( Any key stops & resumes printing )
10 IF TEMP @ BASE ! 0 CNTR ! QUIT THEN
11 IF KEY DROP THEN ; ( FCTN 4 terminates VDUMP )
12
13 : FILTER 8 0 DO PAD 2+ I + C@ ( Changes byte values < 32 )
14 DUP 32 < SWAP 126 > OR ( & > 126 to ASCII 46, prints . )
15 IF 46 PAD 2+ I + C! THEN LOOP ; -->

SCR #115
0 ( VDUMP SCR# 2  HHR 1/15/87 )
1 : PRTL 0 @ PRT SPACE 2 +LOOP ;
2
3 : PRT-ASCII SPACE PAD 2+ 8 TYPE CR ;
4 ( PRT-ASCII prints filtered ASCII characters )
5 : HEADER CR " Addr 0 1 2 3 4 5 6 7 ASCII" CR ;
6
7 : VDUMP ( vaddr cnt --- ) ( TEMP holds previous base value )
8 CR CR HEADER BASE @ TEMP ! HEX
9 8 + 8 / SWAP ( Print only full 8 byte line )
10 8 / 8 * PAD ! ( start only at an addr divisible by 8 )
11 0 DO GETDATA PAD @ PRT 2 SPACES ( PRT prints address )
12 PRTL 0 FILTER PRT-ASCII ( Prints header each )
13 CNTR @ 19 > IF HEADER 0 CNTR ! THEN ( 20 lines )
14 1 CNTR +! 8 PAD +! STOP ( Increments addr by 8 )
15 LOOP TEMP @ BASE ! 0 CNTR ! ; ( Restores base )
“Contributions from the Forth Community”
A Forth Interest Group Library

SUBMISSION FORM

Name: __________________________
Address: _________________________

Phone Number: (_____) _________________________

SOFTWARE SUBMISSION

Type:  □ New Public Domain Program(s)
       □ New User-supported Program, Suggested Donation $ ________________
       □ Corrected Version of Program Already in Library
          [Disk#]/Filename(s)

Source Forth Standard: □ 83 □ 79 □ fig □ other ________________

Machine: □ IBM/Clone □ Macintosh □ Atari
          □ Amiga □ CP/M □ other ________________

Special Requirements: □ Color □ Monochrome
                      □ other ________________

Material Descriptions:
(1) Please describe each file enclosed in a plain ASCII text file called “FILES.DOC”.
(2) Describe the loading procedure and special notes in a plain ASCII text file called “READ.ME”.
(3) Describe submission, here, in less than 160 characters for inclusion in the FIG Order Form.

(4) A more complete description for special feature in Forth Dimensions:

______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________

These programs and all accompanying materials may be published and distributed under the direction of FIG, without compensation to me and without further permission from me. I have the right or have obtained the right to submit this material. I have the right to also submit this material for publication to other publishers.

Signed __________________________
Date __________________________
Algorithms
- Check Digit
  - Self-Checking Numbers, Vol 10, Issue 5, pg 9
- Linear Automata
  - Linear Automata, Vol 10, Issue 2, pg 23
Applications of Forth
- Time Accounting
  - Time-Keeping Routine, Vol 10, Issue 6, pg 30
Architectures
- 32-bit
  - Shadow Stacks & Double-Precision Numbers, Vol 10, Issue 3, pg 7
  - Letter, Vol 10, Issue 4, pg 6
Assemblers
- A 6502 Assembler, Vol 9, Issue 5, pg 19
  - Letter, Vol 10, Issue 2, pg 9
  - Assembler-only Equates
  - Letter, Vol 10, Issue 2, pg 9
  - Disposing of
    - Have Your Assembler..., Vol 10, Issue 1, pg 22
Bulletin Boards
- BBS Gateways
  - The Best of GENie, Vol 10, Issue 4, pg 29
Commenting Code — see Source Code Styling Conventions
Compilation
- of Executable DOS Files (F83)
  - Standalone Applications, Vol 10, Issue 4, pg 15
  - of Routines with Separated Heads
  - Headerless Local Variables and Constants, Vol 10, Issue 1, pg 19
Compiled Code Decompilers
- The Visible Forth, Vol 9, Issue 3, pg 18
  - Letter, Vol 10, Issue 2, pg 7
Conventions and Exhibitions
Data Objects and Associated Methods
- Designing Data Structures, Vol 10, Issue 2, pg 12
- Designing Data Structures, Vol 10, Issue 5, pg 19
  - Object-Oriented Forth, Vol 10, Issue 2, pg 15
Data Structure Design
  - for Portability
    - Designing Data Structures, Vol 10, Issue 3, pg 31
    - Designing Data Structures, Vol 10, Issue 4, pg 26
  - for Reuse
    - Designing Data Structures, Vol 10, Issue 2, pg 12
  - Arrays
    - Designing Data Structures, Vol 10, Issue 4, pg 26
  - Stacks
    - Convenient Extra Stack, Vol 10, Issue 3, pg 5
    - Shadow Stacks & Double-Precision Numbers, Vol 10, Issue 3, pg 7
    - Letter, Vol 10, Issue 4, pg 6
  - String Stacks
    - Apple II SFORTH (ProDos), Vol 10, Issue 5, pg 14
    - Using a String Stack, Vol 10, Issue 3, pg 15
    - SuperStacks (Arrays of stacks)
      - Letter, Vol 10, Issue 4, pg 5
  - Data Structures In-line with Code
    - Local Variables, Vol 9, Issue 4, pg 9
    - Letter, Vol 10, Issue 1, pg 6
  - Data Types and Associated Operations
    - Dates
      - Conversion Operations
        - Formatting Source Code, Vol 10, Issue 6, pg 10
      - Integers, double
        - Arithmetic operations
        - Formatting Source Code, Vol 10, Issue 6, pg 10
    - Strings
      - Assorted Operations
        - Using a String Stack, Vol 10, Issue 3, pg 15; Issue 4, pg 30
      - Compiling Strings
        - Improved String Handling, Vol 10, Issue 1, pg 15
      - Parsing
        - Improved String Handling, Vol 10, Issue 1, pg 15
  - Database Records and Associated Operations
    - dBASE File Access Method
    - Access to dBASE Files, Vol 10, Issue 1, pg 10
  - Decompiling — see Compiled Code Decompilers
  - Dictionaries and Associated Operations
  - Managing Large Dictionaries
    - Some Words About F83’s Words, Vol 10, Issue 3, pg 26
  - Disk OS Structures and Associated Operations
    - Directories for Screens
    - Simple Screen Directory, Vol 10, Issue 4, pg 8
Now you have a full screen and keyboard manager that makes your application interface truly simple to design and implement. You get full control of all display features—either from within your application or from PMF. All you do is access a file—PMF does the rest.

- **Easy to use:** no source code to include, no subroutines to call. Creates a standard interface for all your applications.
- **Portable:** works with *any* language compiler or interpreter that runs on an IBM PC or compatible.
- **Flexible:** dynamically reconfigurable; removable resident driver uses only 48K. Built-in editor lets you paint panels rapidly.

$39.95

Send check or money order to:

Advanced Functions Technology
1761 Cardinal Drive
Clearwater, FL 34619
(813) 786-8087

Text-based applications only. Requires DOS 3.0 or higher.
Call for Papers

There is a call for papers on the use of Forth technology in Embedded Systems. Papers are limited to 5 pages, and abstracts to 100 words. Longer papers will be considered for review in the refereed Journal of Forth Application and Research.

Please send abstracts by March 15, 1990 and final papers by May 15, 1990.

For more information, contact:

Lawrence P. Forsley
Conference Chairman
Institute for Applied Forth Research, Inc.
70 Elmwood Avenue
Rochester, NY 14611
(716)-235-0168 • (716)-328-6426 (FAX)
Forth programmers can tap a very large pool of talent...

The joining of these two virtual nets into one ForthNet has resulted in a great, single message base for all Forth users. For this, much credit must go to Jerry Shifrin. I was initially porting messages of interest from Don Madsen's North Coast Forth Board to GENie. This helped extend the information input, but the circle was only half complete until Jerry developed methods to make the message bases flow to the East Coast Forth Board (and the other xCFB nodes) from GENie, thereby completing the circuit: everything became a homogeneous message base, whereby users of all the services are able to communicate with one another, an exciting development.

Now Forth programmers are able to tap a very large pool of talent to help resolve problems. A new user of Forth encountering an editor problem received seven responses from four sources in two days. That is networking at its best! As a GENie Forth RoundTable sysop, I remind you that all this is available simply by logging onto the GENie Forth RoundTable; you need not worry about the mechanics of ForthNet. All that is asked in return is your participation. Nothing could be easier.

All is not sunshine and roses, however. Much porting is still done by one or two individuals. The absence of either shows immediately. This is the bane of most virtual networks, though. Usenet functions only by the grace of the volunteer postmasters at its various Unix sites, which is partly why links in that system come and go. We hope to emulate their success and have more links join ForthNet than leave.

Exciting trends and possibilities are afoot. We are currently establishing a link in Australia on Lance Collins' BBS for the PCBoards. We are also trying to establish a Usenet link with other Forth sources. We are porting messages into GENie for distribution from Usenet comp.lang.forth, and are creating an edited ForthNet packet for distribution on Usenet. We have been in touch with several Forth users with whom we previously would have had no real hope of establishing contact. We encourage all who have contacted us to continue to do so, and those who have not to please do so. Each new contact is like a new discovery.

Jack Woehr
jax on GENie
jax@well.UUCP
jax@chariot.UUCP
issue of *Forth Dimensions* and, as usual, enjoyed reading your "Best of GEnie" section. I note you have a quote from Larry Forsley in which he says, "JFAR [The Journal of Forth Application and Research] V.2 will be going to the printer just before Christmas... JFAR V.3 and V.4 papers are now being processed. I expect that volume to be finished by June '89."

The quote was dated December 1988. Did all this really come to pass? We have not received any copies of JFAR since Volume IV.

I thought C.H. Ting’s comparison of Forth and Zen (as well as his quote from Lin Yu-Tang) very apt.

Regards,
Ted Rofe

P.S. What I do with Forth:
I develop software for a graphics board prototype (PC based) with a Texas 34010 processor, and I have the interface from the PC side written in Forth. The interface is only for development purposes, and it includes a mini-debugger for 34010 machine code. The graphics board will be a part of a system for nuclear medicine.

To: Paulo A. D. Ferreira
From: Gary Smith
Subject: Forth BBS’s

As a new FIG member, I saw your article in *Forth Dimensions*, regarding Forth on-line resources. But... I only have access to an X25 system and I have no modem, so could you please tell me the addresses of some BBSs accessible this way? I would be very grateful.

—Paulo Ferreira

---

To: Lance Collins
From: Gary Smith
Subject: JFAR

Thanks for the messages which have been filtering back to me on paper from Ted Rofe. Turns out one of our chapter members can get to Usenet mail, but his research assistant status is on a month-to-month basis at present, so he is not the mailbox to tell you about.

Have had snail mail from Jack Brown re: PCBoard, and have ordered and received a copy. (Please mail the registration card for me.) Have only had a quick play so far, but am not as impressed as I hoped I would be. Seems that, like OPUS, third-party shareware is needed to make it work well. Especially netmail, which is where we started on this. Have written to Jack Brown saying I have ordered PCBoard and asking for the shareware, etc., extras he promised.

On phone costs, a five-minute call could transfer a 30-50K ARC file at 2400 bps for less than ten dollars. For once a week the cost is painful but bearable, at least for a few months as an experiment. We will also have to pay Jerry and Jack for disks and mailing of files.

Then there are 9600 modems, if satellite links actually provide better throughput (moot).

You said to Ted Rofe that you only found out about our BBS recently. I wrote to Kent Safford about it last November. I also asked Marlin Ouverson if he would like an article about how we got our BBS set up. (Tell JAX about this, in *FD XI/2* he says he dreams of every chapter having a BBS on ForthNet).

Following that last thought, if there was a Forth file library and many remote BBSs like ours, maybe FIG could provide an update service for the chapters to subscribe to. Then we could spend our scarce dollars on netmail and let snail mail carry the bulky stuff.

Regards,
Lance
Have written to Jack Brown saying I have ordered... Tell JAX about this...

I am posting this exchange to GEnie/ForthNet. Dennis Ruffer, Jack Brown, Jack Woehr (jax), Jerry Shifrin, et al. will read and know we are indeed creating a necessary service. I will leave it to the FIG directors and the Chapters Coordinator to reply to your points.

> Maybe FIG could provide an update service...

That’s the plan, Lance. That’s the plan. Regards,
Gary

ASSOCIATION JEDI
17, Rue de la Lancette
F - 75012 PARIS (FRANCE)

Dear Gary,

After your letter published in Forth Dimensions (XI/1), we inform you about the French Forth BBS JEDI. To contact us, connect with the Teletel network:

33 36 43 15 15
Access-code: SAM*JEDI
Sysop: Marc PETREMANN (SECRETARE)

Set your modem and communication software to 1200/75 baud with 7 data bits, even parity, and one stop bit. The capacity access of SAM*JEDI is 32 simultaneous ways.

If you call from Houston, you can access SAM*JEDI via the USVIDEOTEL network. For more information, call:

Videodial Inc.
1700 Broadway
New York, New York 10019
Telephone 212-307-5005

With our respect,
Marc

To: ASSOCIATION JEDI
From: Gary Smith
Dear Marc and Association JEDI,

Thank you for your letter of 24 July, which I received today, advising me of the Forth BBS JEDI.

I will post your letter to the GEnie Forth RoundTable and on ForthNet today. I will also include you in our resource listing update in Forth Dimensions. You should realize it will be at least two issues before the information will appear in Forth Dimensions, so please be patient.

If any of your association members have access to Usenet, it would make electronic contact with you much easier. Both the FIG Chapters Coordinator Jack Woehr (jax) and myself (gars) are easy to contact via Usenet. If possible, please do so.

Regards, Gary

To suggest an interesting on-line guest, leave e-mail posted to GARY-S on GEnie (gars on Wetware and the Well), or mail me a note. I encourage anyone with a message to share to contact me via the above or through the offices of the Forth Interest Group.

(Continued from page 21.)

VDP RAM. Both RESETV and FORGET must be used to forget a VDP variable.

Screens 112, 113—Several versions of the VDP definitions were written, in an effort to optimize for speed. Those shown on screens 110–111 were the fastest. Slower versions are shown on screens 112–113. It would probably improve the usefulness of the VDP words in manipulating large arrays if even faster versions could be written.

A convenient timer, used to determine the speed of the various definitions, is based on the VDP interrupt. START was placed at the beginning of a definition, and TIMER at the end. The elapsed time in seconds * 60 was then printed.

: START
  0 33750 ! ;

: TIMER
  33750 @ 2 / . ;

Screens 114, 115—VDUMP was written to provide a convenient dump of VDP RAM to the screen or printer, and is the equivalent of the TI-Forth word DUMP. It provides the VDP addresses and their contents, both in hexadecimal numbers and ASCII equivalents. Unprintable characters are printed as a period. A typical dump is shown in Figure One.

Howard H. Rogers, developer of the nickel-hydrogen battery, is a senior scientist working with satellite batteries for Hughes Aircraft. He earned his Ph.D. at M.I.T. in 1953, and uses Forth "to keep his technical background up-to-date."
for the second annual

**FORTH APPLICATIONS WORKSHOP**

on **REAL TIME DEVELOPMENT**

**COLONY PARKE HOTEL • Dallas, TX • Feb. 16-18, 1990**

The objectives of this workshop are to share, discuss and disseminate recent research on and techniques (hardware and software) in real time development tools, methods and environments. Attendees will hear presentations from industry experts on many topics, including:

- Development Tools
- Programming Environments
- Fault Tolerant Systems
- Development methods
- Execution Monitoring
- Debugging Environments
- Embedded System Considerations
- Forth Engines and Software
- Multitasking/Multiuser Systems
- Engineering considerations
- Development System Architectures
- Programming Methods

Papers for oral and poster presentations are requested from computer professionals and other interested parties. Facilities will be available for scientific and technical demonstrations. Proceedings will be made available to the participants of the workshop. Vendors of software and/or hardware may request exhibit space. Authors should submit an abstract of 250 words or less, typed, double spaced, by the deadline below. Contributed papers should be previously unpublished work. You are not required to present a paper to attend the workshop.

**Please send abstracts and requests for workshop information to:**

*Conference Chairman*
Howard Harkness
3316 Vine Ridge
Bedford, Texas 76021
(214) 580-1515 x545

**TIMETABLE:**
Receipt of abstract: Nov. 1, 1989
Receipt of paper: Dec. 1, 1989

*Sponsored by the ACM Special Interest Group on Forth*

For ACM SIGForth membership information, contact:
ACM, 11 West 42nd St., New York, NY 10036 (212) 869-7440
Forth Interest Group

The Forth Interest Group serves both expert and novice members with its network of chapters, *Forth Dimensions*, and conferences that regularly attract participants from around the world. For membership information, or to reserve advertising space, contact the administrative offices:

Forth Interest Group
P.O. Box 8231
San Jose, California 95155
408-277-0668

Board of Directors
Robert Reiling, President (ret. director)
Dennis Ruffer, Vice-President
John D. Hall, Treasurer
Terri Sutton, Secretary
Wil Baden
Jack Brown
Mike Elola
Robert L. Smith

Founding Directors
William Ragsdale
Kim Harris
Dave Boulton
Dave Kilbridge
John James

In Recognition
Recognition is offered annually to a person who has made an outstanding contribution in support of Forth and the Forth Interest Group. The individual is nominated and selected by previous recipients of the "FIGGY." Each receives an engraved award, and is named on a plaque in the administrative offices.

1979 William Ragsdale
1980 Kim Harris
1981 Dave Kilbridge
1982 Roy Martens
1983 John D. Hall
1984 Robert Reiling
1985 Thea Martin
1986 C.H. Ting
1987 Marlin Ouverson
1988 Dennis Ruffer

ANS Forth
The following members of the ANS X3J14 Forth Standard Committee are available to personally carry your proposals and concerns to the committee. Please feel free to call or write to them directly:

Gary Betts
Unisyn
301 Main, penthouse #2
Longmont, CO 80501
303-924-9193

Mike Nemeth
CSC
10025 Locust St.
Glenndale, MD 20769
301-286-8313

An-trew Kobziar
NCR Medical Systems Group
950 Danby Rd.
Ithaca, NY 14850
607-273-5310

Elizabeth D. Rather
FORTH, Inc.
111 N. Sepulveda Blvd., suite 300
Manhattan Beach, CA 90266
213-372-8493

Charles Keane
Performance Packages, Inc.
515 Fourth Avenue
Watervliet, NY 12189-3703
518-274-4774

George Shaw
Shaw Laboratories
P.O. Box 3471
Hayward, CA 94540-3471
415-276-5953

David C. Petty
Digitel
125 Cambridge Park Dr.
Cambridge, MA 02140-2311
617-576-4600

Forth Instruction
Los Angeles—Introductory and intermediate three-day intensive courses in Forth programming are offered monthly by Laboratory Microsystems. These hands-on courses are designed for engineers and programmers who need to become proficient in Forth in the least amount of time. Telephone 213-306-7412.

On-Line Resources
To communicate with these systems, set your modem and communication software to 300/1200/2400 baud with eight bits, no parity, and one stop bit, unless noted otherwise. GEnie requires local echo.

GEnie
For information, call 800-638-9636
- Forth RoundTable (ForthNet link*)
  Call GEnie local node, then type M710

(Continued on next page)
or FORTH
SysOps: Dennis Ruffer (D.RUFFER), Scott Squires (S.W.SQUIRES),
Leonard Morgenstern (NMORGENSTERN), Gary Smith (GARY-S)
• MACH2 RoundTable
Type M450 or MACH2
Palo Alto Shipping Company
SysOp: Waymen Askey (D.MILEY)

BIX (ByteNet)
For information, call 800-227-2983
• Forth Conference
Access BIX via TymeNet, then type j forth
Type FORTH at the : prompt
SysOp: Phil Wasson (PWASSON)
• LMI Conference
Type LMI at the : prompt
Laboratory Microsystems products
Host: Ray Duncan (RDUNCAN)

Unix BBS's with Forth conferences (ForthNet links*)
• WELL Forth conference
Access WELL via CompuServeNet or 415-332-6106
Fairwitness: Jack Woehr (jax)
• Wetware Forth conference
415-753-5265
Fairwitness: Gary Smith (gars)

References

This paper was presented by R. Haskell at the 1988 Real-Time Programming Convention in Anaheim, California.
Harry S. Truman may have been America’s finest President in the second half of the twentieth century. His critics saw him as a feisty poltroon, but there is no denying that throughout the course of his public life he got results, from the courthouse and paved roads of his home county in Missouri, to the investigation of corrupt defense contractors, to the defeat of Japan and the restructuring of the American economy for peacetime in the aftermath of World War II. Truman’s best-known dictum regarding public office was, “If you can’t stand the heat, get out of the kitchen.”

When John D. Hall turned over the Chapter Coordinator’s office, there was no question that I could stand the heat. Heck, I’m notorious at generating it myself; ask around the xCFB boards! The problem I’m having is that I’m not sure I have found the kitchen.

One drawback to having freedom was explored in the popular novel The Unbearable Lightness of Being: you have to make your own decisions. Forth Interest Group chapters operate in near-total freedom from the dictates of the central organization. Read the Chapters Guide, available from the FIG office, if you have any doubts on this matter. About the only thing you can’t do in a FIG chapter is print your own dollar bills with Chuck Moore’s picture on them or fall under a five-FIGger membership level.

The FIG office handles chapter registration in a methodical and more-or-less automatic fashion. Applications are received, and if they meet the criteria outlined in the Chapters Guide, they are approved. The Chapter Coordinator is notified. Chapter Coordinator scratches head, writes another bi-monthly column, writes another monthly email newsletter, tempus fugit.

What are the services that the Chapters would like to see provided by the central organization? If we had our way, we’d travel around the world and see you all personally but, that being impractical, we have chosen what we feel to be the next best thing, networked telecommunications. It is gratifying to hear from overseas chapters; Ted Rolfe emailed us this month from Australia to say that our description of the ups and downs of Denver FIG sounded like his own chapter. We sense, however, that if FIG is to make a contribution to the viability of the local chapter, more is called for than chit-chat.

One of the most oft-mentioned suggestions for FIG is that its membership rates need restructuring. The Board is in the process of considering that request and I believe we’ll see action on it before long. But aside from making membership fees more suited to the type of individual member, what other efforts can we make to augment the success of your chapter?

I believe that chapters exist primarily for the mutual edification of Forth programmers. After winning the World Chess Championship in 1975, Anatoly Karpov in an interview commented that, “We are all, after all, merely engaged in learning to play better.” I think that applies to Forth programmers also. I know from personal experience that I have seen some of the great minds of Forth in the instant that their face brightened as a new idea struck them in the course of an exchange at a monthly FIG meeting. Harry Truman was fond of saying that an “expert” was a person who was afraid of learning anything new for fear that he would no longer be an expert. There are few experts at FIG meetings.

To the end of mutual edification, FIG offers chapters discount rates on the literature advertised in the center of every issue of Forth Dimensions. Are there other efforts we should be making on your behalf to ensure that the bewildered may receive enlightenment at your monthly meetings?

Many chapters have corresponded electronically with us, and I have requested to be placed on the mailing list for chapter publications. To date, I have received personal correspondence from certain chapters, but only BC FIG has seen fit to reach me with a newsletter, which was easy since their newsletter—a model for such—is distributed on-line. If your chapter has a publication, a meeting notice, or a newsletter, would you please include me on the distribution list?

The subject of a speakers bureau has been bandied about for years. I mentioned it in my last column, pointing out that a new face can be the key to better meeting attendance. Since I write this article only a day after Forth Dimensions arrived, I have not received any feedback on this point yet; however, I expect to receive some. Are we ready for a speakers bureau? Is your local chapter ready to encourage its members who will be visiting other cities to register to speak at another FIG chapter?

Youth being e’er the key to the future, we are considering attempts to organize introductory Forth presentations for secondary schools. Presentations could be made by members of the local chapters with teaching aids provided through the central organization. This would have the dual effect of encouraging young men and
women to explore computer science using Forth methodology, while providing them with the public-domain tools with which to do so. It would also, not insignificantly, provide an interesting, novel and rewarding group activity for chapter members. Furthermore, it is hard to imagine that a Forth presentation at a school would not net at least two or three new FIG members. Is this something your chapter would actively support and participate in, were the teaching aids and introductory letters to secondary institutions forthcoming?

John Hayes is the author of several Forth articles and a key figure in the VLSI Forth microprocessor project at...
IN SEARCH OF A BETTER NUMBER INPUT ROUTINE

MIKE ELOLA - SAN JOSE, CALIFORNIA

I have long sought a number input routine that is simple yet flexible. Over the years, I have created and abandoned a variety of number input routines. The process has helped me understand some of the difficulties of using Forth for a programming project.

The off-then-on-again search started out as a response to a troublesome Forth problem. Forth simply terminates execution if an error is detected during number conversion. Not only did I want to be rid of such stoppages, I also planned to add support for various number input formats, such as dates and currency amounts.

My earliest number input routines relied heavily upon standard Forth routines. Typically, these routines processed strings in three distinct stages: EXPECT obtained an input string from a user; a new routine ensured that the contents of the input string would not trigger early program terminations; and NUMBER converted the string to a double. (The term double indicates a 32-bit integer.)

This approach traps errors before they become fatal, and allows a modicum of non-fatal error processing to be added. However, this belated error processing had its own problems too. It would have been much better to check each keystroke as it was entered, thereby detecting and correcting errors more naturally. At this juncture, the program could accept many keystrokes from the user; then, if NUMBER disliked any one of them, all of them would be tossed.

Errors become more difficult to correct when discovered late. Still, I was glad to have removed the worst offence to the users of my program—having to rerun a program which crashes.

Eventually I was able to develop input routines that checked for invalid keystrokes as each keystroke was entered, but it required substituting Forth's EXPECT routine with another of my own making. Even with these improvements, EXPECT never could be transformed into a straightforward, easy-to-maintain, and easy-to-enhance routine. For example, the effort to enhance EXPECT to display a number sign or a fixed decimal point was far more challenging that it should have been. I ended up with many different versions of EXPECT because any new feature prohibitively increased the difficulty of adding the next feature.

With the need for something entirely different becoming clear, I started the search for completely new input routines to graft onto Forth. The early results are shown in Figure One. The routines GET# and GET2# convert input strings directly to numbers, bypassing the use of NUMBER and EXPECT (but not completely).

This solution came closest to creating a toolset of routines. The most basic of these routines, GET#, obtains just one digit of a number. But the intended progression from the GET# routine to the anticipated GETn# routine never materialized. The routines did not dovetail as well as I had hoped. I had planned at least to use GET# to derive GET2#. However, I simply gave up on this path of development due to the difficulty of implementation.

But I still dreamed of developing a collection of routines, each addressing part of the problem. While examining the available functions for entering and displaying data, I noted how much these functions were like inverses of one another. Table One [page 40] helped illustrate.

Observe that the output routines output the input routines. The output routines are also well factored, and flexible enough to support a variety of number formats, such as currency, dates, and so forth. The same is not true for the input routines. (I suspect that the same type of disparity is shared by many other computer languages.)

Could the source of the disparity be the different levels of factoring which had been applied to the input and output functions? Judge for yourself. The processing steps for the conversion of string inputs to numbers are:

1. Set a pointer to the start of a string and zero the string-length count.
2. Divide the double number on the stack by the current base, converting the remainder into the next digit to be added to the head of the string.
3. Increment the string-length count.
4. Repeat steps 2 and 3 for every call to #.
5. Remove the double number from the stack, replacing it with the address of the string and its length count in preparation for the use of TYPE.

Optional steps for the conversion of numbers to output strings include placing additional, non-numeric characters into...
the string with HOLD. Another common option is to call the routine #S, which repeatedly calls # until the double on the stack is equal to zero, indicating that no more significant digits remain to be converted into ASCII digits.

To correct the imbalance, better-decomposed functions would be needed for number input. Some of the functions that might become more discrete are: check keystroke for errors; update the string that is being displayed; and if the latest keystroke was a digit key, convert it according to the current number base and use it to update the current numeric value entered.

Next, I realized that the input routine did not have to build a string. Since its aim was to obtain a numeric value, it needn’t work with an intermediate string at all. I was beginning to realize that a certain degree of imbalance—and slightly less-parallel structures—might be preferable for the input and output routine toolsets. But I was still trying to correct the lack of parallel structure when I created Table Two.

Each instance of B# would correspond to the entry of a particular digit. The anticipated B# routine requires its own error handling (see Figure Two).

In the same way, slightly different versions of B# could handle the entry of a minus sign as an error or as a legal entry, depending on the position; at the beginning digit position, a minus sign might be permissible. Different processing can take place at each digit position, which closely parallels Forth’s number-to-output-string toolset.

But the remaining obstacles were not going to permit me to realize such parallel structures. One obstacle is the inability to undo B# functions in order to support backspace deletions.

By placing even more functionality inside of a keystroke-handling routine, I could ease the difficulties of the implementation. Each time around the key-input loop, the numeric value accumulated and the corresponding string are recomputed and redisplayed. Here is the flow of steps for the envisioned routine:

1. Get a single, undisplayed character.
2. Reject invalid key codes, returning to step 1 as necessary.
3. Convert the ASCII character to a number and accumulate it into the double on the stack.
4. Duplicate the double on the stack.
5. Convert the duplicated value to an ASCII string and display at a fixed location on the display screen.
6. Loop back to beginning.

Each character is lost after its accumulation into the double on the stack. So, each time through the key-input loop, the new routine would recompute the ASCII string that represented the entry underway. This is no small addition to a routine that is primarily concerned with providing an input function. However, the existing Forth output routines are able to build this string easily, and in just about any format desired. This approach imparted the flexibility that no other approach had offered, so that dates, currency amounts, and the like could be handled by the same input routine (as exemplified in the accompanying sidebar).

Figure Three shows the necessary source code for two versions of a number input routine (with shared primitives). The first version of DINPUT allows backspace editing of the default value, while the sec-

---

**TDS 9090 FORTH COMPUTER**

**Ideal for starter, teaching or target system**

- **build into your product for rapid completion!**
  - complete Fig-Forth system
  - 30K RAM; 16K EPROM
  - complete program with IBM-PC
    - connect to keyboard, lcd display, RS 232
    - 35 I/O lines; 10 bit A/D option
    - low power - down to 3 ma @ 6-18V

Connect the 4" x 3" TDS 9090 single-board computer to an IBM-PC or compatible and start writing Forth code immediately! Lots of ready made application programs come with the kit to do interrupt-driven I/O, graphics I/O, driver, frequency measurement, solid-state speech and data-logging. The board includes a ROM-resident Forth language kernel and an assembler. By storing generated code in either non-volatile RAM or EPROM, the board can be used in a target system or stand-alone product. Based on the CMOS Hitachi HD 63A03Y microprocessor, it has two timers, two serial ports and interrupts, which are available via Forth instructions. Also included on board are 30K RAM for storing source code or data, 16K EPROM/hirom for firmware, 256 bytes EEPROM, 35 I/O lines, two RS 232 serial interfaces, a watchdog timer to insure recovery from crashes, and an expansion bus. Interface the TDS 9090 to an 8 x 8 keyboard or an LCD display, or use two of the I/O lines as an I2C interface. The ROM-resident Forth is an extended version of Fig-Forth with Forth words to support all the onboard peripherals, as well as the keyboard and LCD interfaces. Pre-programmed software in PROM and it starts to run as soon as power is applied. Made in England by Triangle Digital Services, and well-known in Europe, the TDS 9090 is now supported in the USA and is available with less than two week delivery at only

**$219** (25qty)

The Saelig Company 1193 Moseley Rd Victor NY 14564 USA
tel: (716) 425-4367 or fax (716) 425-7381

---

**WE'RE LOOKING FOR A FEW GOOD HEADS.**

Dash Find Associates

Forth Recruiters

70 Elmwood Ave. Rochester, NY 14611 (716) 235-0168
ond discards the default and uses a zero starting value if any key other than Return is pressed as the first keystroke.

The keystroke-handling code spans both the DINPUT and CHAR>D+ routines. The keystroke interpreter is bound by BEGIN ... REPEAT or BEGIN ... UNTIL constructs in the two versions of input routine offered.

Because of the complexity that would have been introduced to support backspace deletion, the very graphically informative B# notation was abandoned. However, a similarly graphic notation was recovered through the use of a new number display routine, the details of which are described in the sidebar that accompanies this article.

This routine lets the usual Forth number output toolset routines (such as # and HOLD) do the work of building the output string, but the order of operations is all under the control of yet another string, which I call a picture string.

Structural Considerations

I was most surprised the find that the structure of my first acceptable number input routines was so unified. I had come to expect that the real solution would be a diverse collection of mix-and-match routines. Even though I favored a solution that paralleled Forth's number output routines, I became satisfied with the structure of the new routine once I understood it better.

Rather than create input routines to parallel the Forth output counterparts, the input routines have subsumed the output routine functions. So, in at least one sense, DINPUT is composed of mix-and-match routines. These reusable routines were not developed anew, but borrowed from the already existing output routines.

Accordingly, DINPUT incorporates a great deal of functional scope. Normally, I would take this as evidence of incomplete factoring. However, input actions are an unusual combination. Routines for accepting user-supplied data must incorporate an output function to let user see their own progress as they press keys.

By coming to this understanding, I began to see how my original efforts to use EXPECT were not theoretically sound. In terms of programming philosophy, the decomposition of input functions through EXPECT and NUMBER were the source of my difficulties all along.

EXPECT clearly belongs to a string class of actions. My need was to obtain avoiding a string representation of those characters, the types of actions performed by DINPUT are better focused upon numeric input.

Proper functional decomposition of a program typically streamlines and simplifies code, making it more reusable. One of the goals of structuring code properly is that unnecessary conditionals can be discovered and removed. Often, conditionals help support modes that must be accounted for in diverse areas in the code. Proper structuring of code should help reduce this complexity.

There are typically many tests and modes within input routines. Among them

Figure One. First efforts yielded these routines.

: B# ( working-value -- new-working-value )
BEGIN KEY
Error? NOT UNTIL
Display-Key
Char>Number ;

Figure Two. Error handling for B#.
are tests such as whether the key just pressed was the Delete key, a minus sign, or a digit. Typically, the routine must also track the previous keystrokes, since the outcome of pressing the Delete key is different if nothing has been entered. Supporting these modes often requires multiple tests of the same condition, indicating that the structure of the code is less than optimal.

Most modal problems have been ironed out successfully within DINPUT. Although modes still exist, they are not examined over and over in diverse areas. For example, when the double value is zero, the Delete key is not honored. Rather than maintaining a string counter that decrements to zero as Delete is depressed and testing the value of that variable, DINPUT tests the numeric value that is under construction on the stack!

Likewise, better functional decomposition supports better error handling. Consider the routine CHAR>D+. This routine converts a character code to the appropriate digit value and then updates the double on the stack accordingly. This is a more incremental way to perform string-to-number conversion, because it takes place one character at a time. The Forth CONVERT routine (called by NUMBER) works on a substring-by-substring basis. CONVERT obstructs error processing at the keystroke level.

Conclusions

The most dramatic end-user benefit imparted by DINPUT is its more forgiving user interface, made possible by improvements in error handling. The routine also has the merit of easy modification in order to display properly formatted numbers even while they are being entered.

Error handling is poor in many Forth programs as a direct result of the use of CONVERT. Had there existed a routine like CHAR>D as part of Forth, I am sure that a number input routine like my DINPUT would have arrived on the scene much sooner.

Nevertheless, I will be the first to assert that CONVERT is completely adequate for supporting the interpreter and compiler functions inside Forth. And I can see why this pleases most Forth programmers: they typically favor the simplest solution for the problem at hand.

Still, to allow increased flexibility and better support for user-interactive applications, we should choose to extend Forth with routines like I have described. While CONVERT is the shortest path to an implementation of Forth, it does not lead to a suitable user interface for an application requiring formatted number input.

To encourage the development of new algorithms and their discussion in articles such as this one, Forth is wonderful. Through its lack of sophistication in many areas, Forth becomes the preferred language for shaping new algorithms. By pursuing the same kind of bare-bones simplicity typical of the supplied Forth routines, more efficient and effective solutions are likely to be found.

Forth is an accommodating language for development along two types of paths: paths directed upwards towards higher levels of functionality within single routines. What I have learned with regard to the development of number input routines is that progress towards higher levels of functionality was made possible through seemingly backwards progress towards more precisely formulated, infinitesimal levels of functionality.

I had to feel comfortable going backwards before I could move forwards.

In my seven or eight years of working with Forth, I always felt too uncomfortable redefining the supplied routines. Perhaps I instinctively felt that the proper use of a high-level language was not to turn it against itself. Or perhaps I admired too much the sage Forth programmers who could implement Forth on new machines, so I felt uncomfortable tweaking their vision of what the language should be on a particular machine.

However, if this is one of Forth’s most

(Continued on next page.)
### Table One

<table>
<thead>
<tr>
<th>Input Resources</th>
<th>Output Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data</strong></td>
<td></td>
</tr>
<tr>
<td>numbers</td>
<td>D.R &lt; # # HOLD #S #&gt; TYPE</td>
</tr>
<tr>
<td>dates</td>
<td>## # HOLD # &gt; TYPE</td>
</tr>
<tr>
<td>strings</td>
<td>COUNT TYPE</td>
</tr>
</tbody>
</table>

### Table Two

<table>
<thead>
<tr>
<th>ASCII string corresponding to number value is built one character at a time for each call to #, proceeding from the tail to the head of the string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>The number atop the stack is updated to reflect each character obtained with B#. or The number top the stack is updated to reflect each digit left atop the stack by B#.</td>
</tr>
</tbody>
</table>
unique strengths, an ability to accommodate development directed at higher and lower levels of functionality equally well, then it should be recognized as a strength.

One of the more disturbing implications of this flexibility is that more advanced Forth programmers will be tempted to displace many kernel routines. I think we have ample evidence of this because the top Forth programmers typically use custom versions of the language. However, the rest of the programming community considers this flexibility and consequent non-standardization as confirmation of Forth's reputed unreadability and unmaintainability.

To help make Forth more than a language laboratory, we'll need to decompose the functions of the language better, and then standardize those miniscule—and seemingly unimportant—functions. In fact, this should be one of the highest priorities of organizations developing Forth standards.

Bibliography

Mike Eloba is a published Forth programmer and a full-time writer at Apple Computer. Over the years, Mike feels, Forth has tricked him into believing he is a computer scientist.
The FIG Chapters listed below are currently registered as active with regular meetings. If your chapter listing is missing or incorrect, please contact Kent Safford at the FIG office's Chapter Desk. This listing will be updated in each issue of Forth Dimensions. If you would like to begin a FIG Chapter in your area, write for a "Chapter Kit and Application." Forth Interest Group, P.O. Box 8231, San Jose, California 95155.

U.S.A.
- ALABAMA
  Huntsville Chapter
  Tom Konantz
  (205) 881-6483
- ALASKA
  Kodiak Area Chapter
  Ric Shepard
  Box 1344
  Kodiak, Alaska 99615
- ARIZONA
  Phoenix Chapter
  4th Thurs., 7:30 p.m.
  Arizona State Univ.
  Memorial Union, 2nd floor
  Dennis L. Wilson
  (602) 381-1146
- ARKANSAS
  Central Arkansas Chapter
  Little Rock
  2nd Sat., 2 p.m. &
  4th Wed., 7 p.m.
  Jungkind Photo, 12th & Main
  Gary Smith (501) 227-7817
- CALIFORNIA
  Los Angeles Chapter
  4th Sat., 10 a.m.
  Hawthorne Public Library
  12700 S. Grevelia Ave.
  Phillip Wasson
  (213) 649-1428
  North Bay Chapter
  2nd Sat., 10 a.m. Fort Collins
  12 Noon Tutorial, 1 p.m. Fort
  South Berkeley Public Library
  George Shaw (415) 276-5953
  Orange County Chapter
  4th Wed., 7 p.m.
  Fullerton Savings
  Huntington Beach
  Noshir Jesung (714) 842-3032
  Sacramento Chapter
  4th Wed., 7 p.m.
  1708-59th St., Room A
  Tom Ghormley
  (916) 444-7775
  San Diego Chapter
  Thursdays, 12 Noon
  Guy Kelly (619) 454-1307
  Silicon Valley Chapter
  4th Sat., 10 a.m.
  H-P Cupertino
  Bob Barr (408) 435-1616
  Stockton Chapter
  Doug Dillon (209) 931-2448
- COLORADO
  Denver Chapter
  1st Mon., 7 p.m.
  Clifford King (303) 693-3413
- CONNECTICUT
  Central Connecticut Chapter
  Charles Krajewski
  (203) 344-9996
- FLORIDA
  Orlando Chapter
  Every other Wed., 8 p.m.
  Herman B. Gibson
  (305) 855-4790
  Southeast Florida Chapter
  Coconut Grove Area
  John Forsberg (305) 252-0108
  Tampa Bay Chapter
  1st Wed., 7:30 p.m.
  Terry McNay (813) 725-1245
- GEORGIA
  Atlanta Chapter
  3rd Tues., 6:30 p.m.
  Western Sizzlen, Doraville
  Nick Hennenfent
  (404) 393-3010
- ILLINOIS
  Cache Forth Chapter
  Oak Park
  Clyde W. Phillips, Jr.
  (312) 386-3147
  Central Illinois Chapter
  Champaign
  Robert Illyes (217) 359-6039
- INDIANA
  Fort Wayne Chapter
  2nd Tues., 7 p.m.
  I/P Univ. Campus, B71 Neff
  Hall
  Blair MacDermid
  (219) 749-2042
- IOWA
  Central Iowa FIG Chapter
  1st Tues., 7:30 p.m.
  Iowa State Univ., 214 Comp. Sci.
  Rodrick Eldridge
  (515) 294-5659
- MARYLAND
  MDFIG
  Michael Nemeth
  (301) 262-8140
- MASSACHUSETTS
  Boston Chapter
  3rd Wed., 7 p.m.
  Honeywell
  300 Concord, Billerica
  Gary Chanson (617) 527-7206
- MICHIGAN
  Detroit Ann Arbor Area
  4th Thurs.
  Tom Chrapkiewicz
  (313) 322-7862
- MINNESOTA
  MNFIG Chapter
  Minneapolis
  Fred Olson
  (612) 588-9532
- MISSOURI
  Kansas City Chapter
  4th Tues., 7 p.m.
  Midwest Research Institute
  MAG Conference Center
  Linus Orth (913) 236-9189
  St. Louis Chapter
  1st Tues., 7 p.m.
  Thornhill Branch Library
  Robert Washam
  91 Weir Drive
  Ellisville, MO 63011
- NEW JERSEY
  New Jersey Chapter
  Rutgers Univ., Piscataway
  Nicholas Lordi
  (201) 338-9363
- Fairfield FIG Chapter
  4th Day, 8:15 p.m.
  Gurdy Leete (515) 472-7077
- FORTH DIMENSIONS
  Volume XI, Number 4
NEW MEXICO
Albuquerque Chapter
1st Thurs., 7:30 p.m.
Physics & Astronomy Bldg.
Univ. of New Mexico
Jon Bryan (505) 298-3292

OHIO
Cleveland Chapter
4th Tues., 7 p.m.
Chagrin Falls Library
Gary Bergstrom
(216) 247-2492

Columbus FIG Chapter
4th Tues.
Kal-Kan Foods, Inc.
(614) 878-7241

VERMONT
Vermont Chapter
Vergennes
3rd Mon., 7:30 p.m.
Vergennes Union High School
RM 210, Monkton Rd.
Hal Clark (802) 453-4442

VIRGINIA
First Forth of Hampton Roads
William Edmonds
(804) 898-4099

Richmond Forth Group
2nd Wed., 7 p.m.
154 Business School
Univ. of Richmond
Donald A. Full
(804) 739-3623

WISCONSIN
Lake Superior Chapter
2nd Fri., 7:30 p.m.
1219 N. 21st St., Superior
Allen Anway (715) 394-4061

INTERNATIONAL
• AUSTRALIA
Melbourne Chapter
1st Fri., 8 p.m.
Lance Collins
65 Martin Road
Glen Iris, Victoria 3146
03/29-2600
BBS: 61 3 299 1787

Sydney Chapter
2nd Fri., 7 p.m.
John Goodsell Bldg., RM LG19
Univ. of New South Wales
Peter Tregear
10 Binda Rd.
Yowie Bay 2228
02/524-7490
Usenet
tedt@usage.cs.d.unsw.oz

• BELGIUM
Belgium Chapter
4th Wed., 8 p.m.
Luk Van Loock
Lariksdrf 20
2120 Schoten
03/658-6343

Southern Belgium Chapter
Jean-Marc Bertinchamps
Rue N. Monnom, 2
B-6290 Malinnes
071/213858

• CANADA
BC FIG
1st Thurs., 7:30 p.m.
BCIT, 3700 Willingdon Ave.
BBY, Rm. 1A-324
Jack W. Brown (604) 596-9764
BBS (604) 434-5886

Northern Alberta Chapter
4th Sat., 10 a.m.-noon
Tony Van Muyden
(403) 962-2203 (eves.)
(403) 486-6666 (days)

Southern Ontario Chapter
Quarterly, 1st Sat., Mar., Jun.,
Sep., Dec., 2 p.m.
Genl. Sci. Bldg., RM 212
McMaster University
Dr. N. Soltzseff
(416) 525-9140 x3443

• ENGLAND
Forth Interest Group-UK
London
1st Thurs., 7 p.m.
Polytechnic of South Bank
RM 408
Borough Rd.
D.J. Neale
58 Woodland Way
Morden, Surrey SM4 4DS

• FINLAND
FinFIG
Janne Kotiranta
Arkkitehdinkatu 38 c 39
33720 Tampere
+358-31-184246

• HOLLAND
Holland Chapter
Vic Van de Zande
Finmark 7
3831 JE Leusden

• ITALY
FIG Italia
Marco Tausel
Via Gerolamo Forni 48
20161 Milano
02/435249

• JAPAN
Japan Chapter
Toshi Inoue
Dept. of Mineral Dev. Eng.
University of Tokyo
7-3-1 Hongo, Bunkyo 113
812-2111 x7073

• NORWAY
Bergen Chapter
Kjell Birger Faeraas,
47-518-7784

• REPUBLIC OF CHINA
R.O.C. Chapter
Chin-Fu Liu
5F, #10, Alley 5, Lane 107
Fu-Hsin S. Rd. Sec. 1
TaiPei, Taiwan 10639

• SWEDEN
SweFIG
Per Alm
46/8-926931

• SWITZERLAND
Swiss Chapter
Max Hugelshofer
Industrieberatung
8152 Opfikon
01 810 9289

SPECIAL GROUPS
• NC4000 Users Group
John Carpenter
1698 Villa St.
Mountain View, CA 94041
(415) 960-1256 (eves.)
NEW FIG DISK LIBRARY
"Contributions From the Forth Community"

FLOAT4th.BLK, V1.02, Robert L. Smith

PocketForth: V1.4, Chris Heilman (1 disk)
Smallest complete Forth for the Mac. Access to all Mac functions, files, graphics, floating point, macros, create stand-alone applications and DA's, based on fig & Starting Forth

F83: V2.01, Mike Perry & Henry Laxen
The newest version that has been ported to a variety of machines. Editor, assembler, decompiler, meta-compiler. Source and shadow screens. Base for other F83 applications. IBM, 83 (1 disk).

VP-Planner Floating Point for F-PC, V1.01, Jack Brown (1 disk) - Floating point engine behind the VP-Planner spreadsheet. 80-bit (temporary-real) routines with Transcendental Functions, NUMBER I/O support, vectors to support numeric coprocessor overlay and user NAN checking. IBM.

F-PC: V2.25, Tom Zimmer
A full Forth system with pull-down menus, sequential files, editor forward assembler, meta-compiler, floating point. Complete source and Help files. Base for other F-PC applications. Hard disk recommended. IBM, 83 (4 disks).

F-PC: TEACH, Lessons 0-5, J. Brown
Forth classroom on disk. First five lessons from Jack Brown of BC Institute of Technology on learning Forth. IBM, F-PC (2 disks).

F-PC: TEACH, Lessons 0-5, J. Brown
Forth classroom on disk. First five lessons from Jack Brown of BC Institute of Technology on learning Forth. IBM, F-PC (2 disks).

JLISP V1.0, Nick Didkovsky (1 disk)
LISP interpreter invoked from Amiga JForth. The nucleus of the interpreter is the result of Martin Tracy's work. It has been extended to allow the LISP interpreter to link to and execute JForth words. It can communicate with JForth's ODE (Object Development Environment). AMIGA, 83.

$6.00 per disk or 5 disks for $25.00
NOW AVAILABLE
FROM THE FORTH INTEREST GROUP

Forth Interest Group
P.O. Box 8231
San Jose, CA 95155