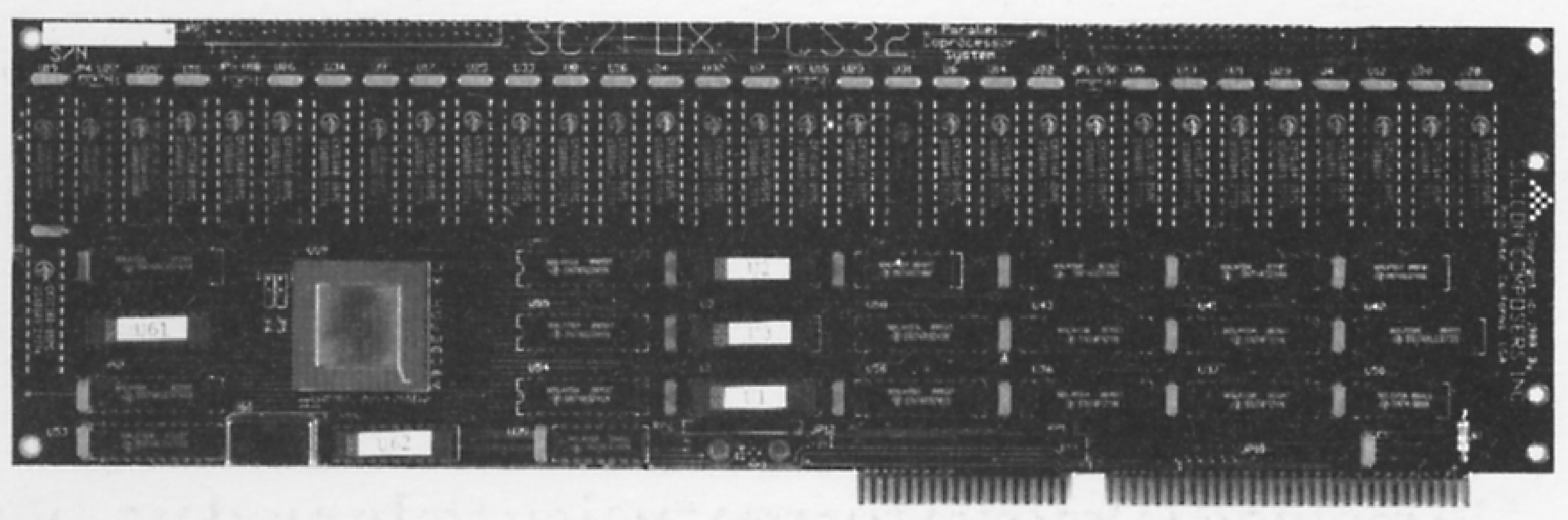
VOLUME XI, NUMBER 4 NOVEMBER/DECEMBER 1989

FORTH DIMENSIONS

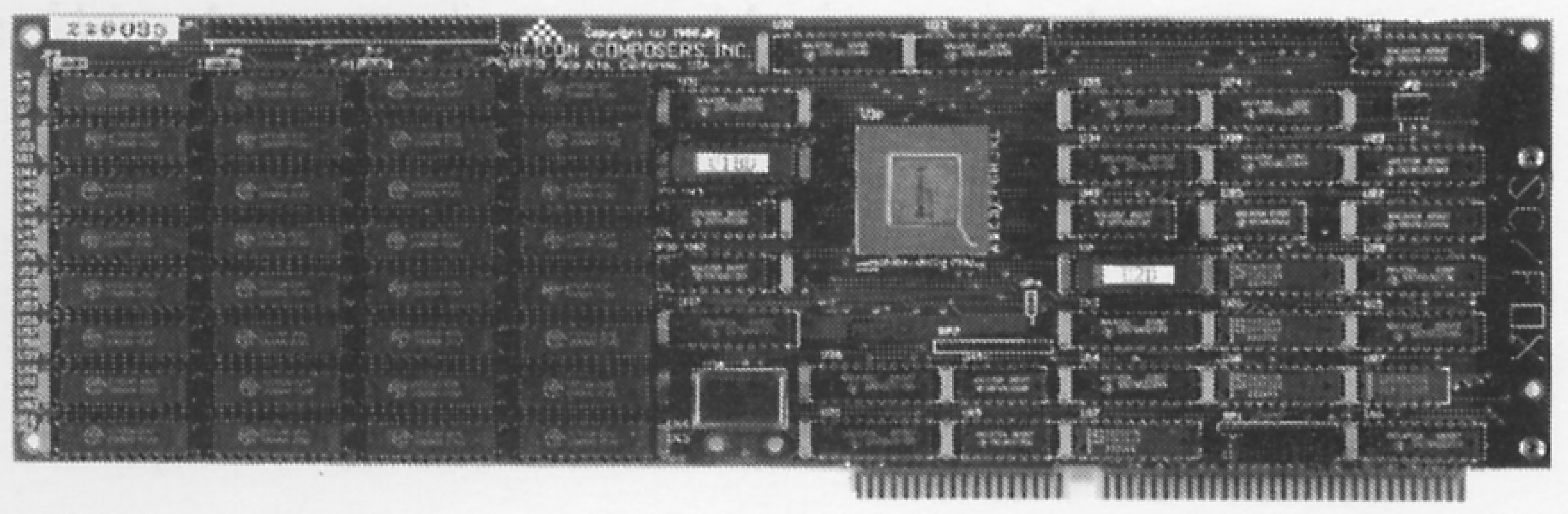
SILICON COMPOSERS

Performance, Quality, Service

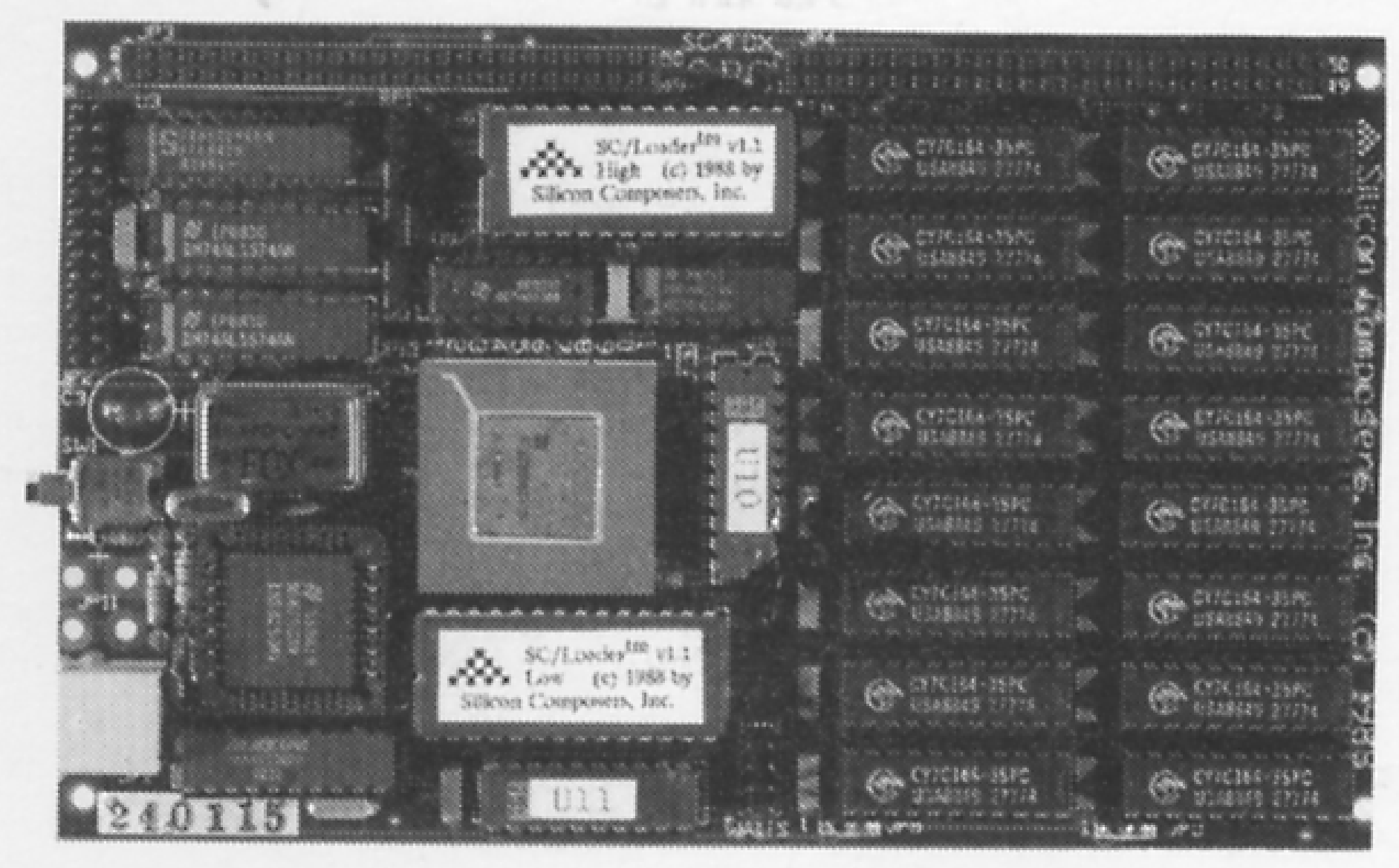
SC/FOX PCS32 Parallel Coprocessor System32
Uses the 32-bit SC32tm Forth CPU.
System speed options: 8 or 10 MHz.
Full-length 8- or 16-bit PC/XT/AT plug-in board.
64K to 1M byte, 0-wait-state static RAM.
Hardware expansion, two 50-pin strip headers.
Includes SC/Forth32, based on the Forth-83 Standard.



SC/FOX PCS Parallel Coprocessor System
Uses Harris RTX 2000tm 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: 100mm by 160mm.
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/Forthtm Language
Based on the Forth-83 Standard.
15-priority time-sliced multitasking.
Supports user-defined PAUSE.
Automatic optimization and μcode 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.

SC32 Forth Microprocessor
32-bit CMOS microprocessor, 34,000 transistors.
One-clock cycle instruction execution.
Non-multiplexed 32-bit address bus and data bus.
16 gigabyte non-segmented data space.
2 gigabyte non-segmented code space.
8 or 10 megahertz full-static operation.
Stack depths limited only by available memory.
Interrupt and interrupt acknowledge lines.
Bus request and bus grant lines with on-chip tristate.
Wait state line for slow memory and I/O devices.
85-pin PGA package.

RTX 2000 Forth Microprocessor
16-bit CMOS microprocessor in 84-pin PGA package.
1-cycle 16x16 parallel multiplier.
14-prioritized interrupts, one NMI.
Two 256-word stacks.
Three 16-bit timer/counters.
8-channel multiplexed 16-bit I/O bus.

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

FORTH
D I M E N S I O N S

EDITORIAL

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 time we ended up using some of his original equations for photostatic reproduction here. It reminded us how far some type-handling utilities 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 microcomputers?

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.

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."

LETTERS

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 act 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
```

```
   FI T
   RUN" I \ "
   LOOP  ;
```

Or you can delete the same commenting by typing:

```
:: 11 5 DO
   FI T
   RUN" D \ "
   LOOP  ;
```

Indentation can also be easily added or removed using RUN". A global replace (up to screen 100, say) can be performed by typing:

```
100 :: BEGIN
     RUN" S old"
     RUN" R new"
     AGAIN  ;
```

(All of these examples work in Laxen and Perry's F83. The word :: will HIDE the last definition, so if you're experimenting with these definitions, you may have to define some junk word just before using ::.)

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.*

```
: (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 ::
```

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

forthWINDOWS+

[illegible]

HORIZONTAL STYLE LIGHT BAR MENUS

MEMORY
RESIDENT
FORTH
16K RAM

POPUP WINDOWS

OVERLAPPING

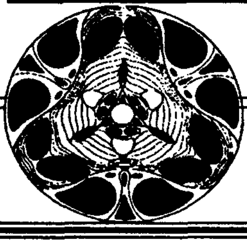
WINDOWS

$49.95

VERTICAL
STYLE
LIGHT BAR
MENUS

MOUSE SUPPORT

Send check or money order
U.S. Funds only



CONCEPT 4 P.O. BOX 204
LINCOLN CA. 95648

ANS FORTH REQUIRED WORDS

JOHN R. HAYES - LAUREL, MARYLAND

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>R`, `2DROP`, `2DUP`, `2OVER`, `2R>`, and `2SWAP`. `2!`, `2@`, 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 a 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 `EXIT`ed 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 Forths 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

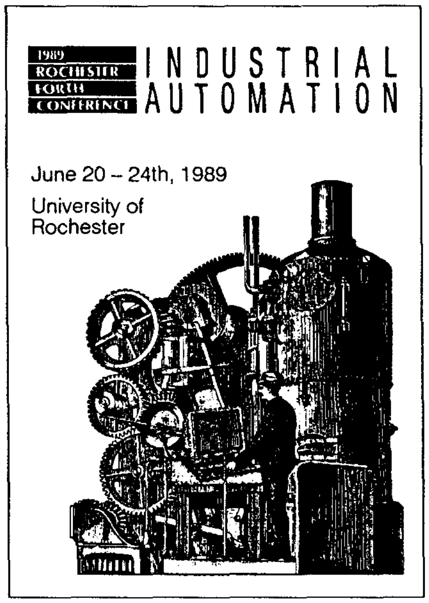
Forth-based Control of an Ion Implanter

```
: 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



1989 ROCHESTER FORTH CONFERENCE INDUSTRIAL AUTOMATION

June 20 – 24th, 1989
University of Rochester

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. Nieuwenhuÿzen, *European Editor*
University of Utrecht, The Netherlands

Upcoming Papers:

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

THE JOURNAL OF FORTH APPLICATION AND RESEARCH

Volume 6 1990

NOW'S THE TIME!

Volume VI Subscriptions

	Individual	**Corporate**
USA	$50.00	$125.00
Canada/Mexico	$60.00	$125.00
Europe/Asia	$70.00	$140.00

1989 Rochester Conference Proceedings
Industrial Automation $25, plus $5 S/H

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.)

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 `D>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 `D>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[1].

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 Forth-83 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 Forth-83 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, `COMPILE` and `[COMPILE]` have been replaced by the single word `POSTPONE`. `COMPILE` 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[2]. In all but a few rare cases, `POSTPONE` may be used instead of `COMPILE` or `[COMPILE]`. For backward compatibility, `COMPILE` may be defined as:

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
  POSTPONE POSTPONE ;
  IMMEDIATE
```

[COMPILE] may be defined (identically) as:

```
: [COMPILE]
  POSTPONE POSTPONE ;
  IMMEDIATE
```

CMOVE and CMOVE> 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:

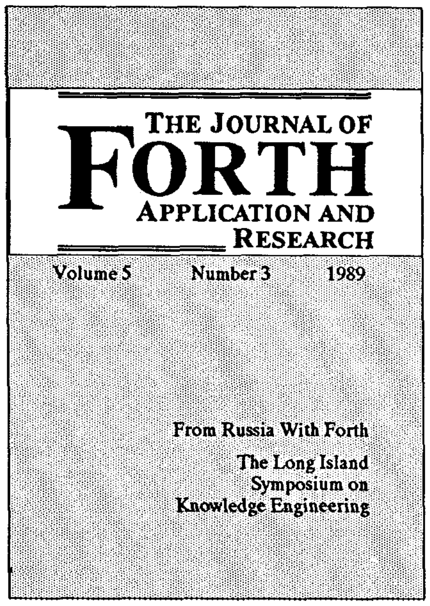
```
: PICK
  ?DUP IF SWAP >R
  1- RECURSE R>
  SWAP ELSE DUP THEN ;

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

Summary

ANS Forth is a descendant of Forth-83. Consequently, knowledge gained about

(Continued on page 35.)

WE'RE LOGGING OUT!


THE JOURNAL OF FORTH APPLICATION AND RESEARCH
Volume 5 Number 3 1989
From Russia With Forth
The Long Island Symposium on Knowledge Engineering

JFAR Volume 5 Number 3

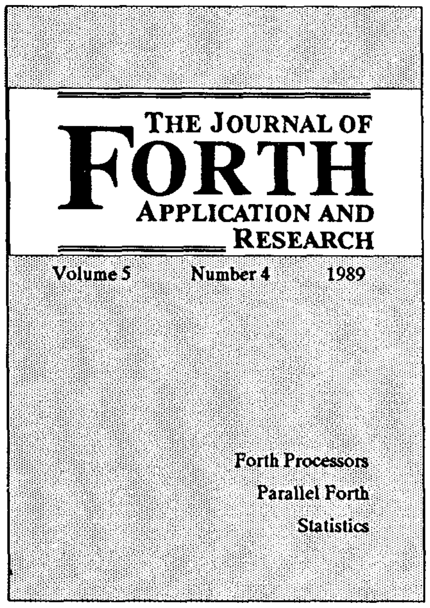
Forth in the U.S.S.R.
Forth for IBM Mainframe Computers
Symbolic Computations on a Personal Computer
S.N. Baranoff, *Leningrad Inst. for Inform.*

Alternative Knowledge Acquisition: Developing A Pulse-Coded Neural Network
W. B. Dress
Oak Ridge National Laboratory

List Processing and Object-Oriented Programming Using Forth
The Prolog Interpreter Algorithm
Dennis L. Feucht
Innovatia Laboratories

Symbolic Stack Addressing
Adin Tevet

An Efficient Algorithm for Locating the Global Maximum of an Arbitrary Univariate Function
Richard E. Haskell


THE JOURNAL OF FORTH APPLICATION AND RESEARCH
Volume 5 Number 4 1989
Forth Processors
Parallel Forth
Statistics

JFAR Volume 5 Number 4

Language Coprocessor Boosting the Execution Speed of Threaded Code Programs
Eddy H. Debaere,
Electronics Laboratory, State Univ. of Ghent

Parallel Forth
John E. Dorband,
NASA/Goddard Space Flight Center

An Arithmetic-Stack Processor for High Level Language Execution
Rodney M. Goodman,
California Institute of Technology
Anthony J. McAuley,
Bell Communications Research

The Architecture of the SC32 Forth Engine
John Hayes and Susan Lee,
Johns Hopkins University / APL

Error-Free Statistics in Forth
Leonard F. Zettel,
Research Staff, Ford Motor Co.

THERE'S STILL TIME

Volume V Subscriptions

	Individual	Corporate
USA	$50.00	$125.00
Canada/Mexico	$60.00	$125.00
Europe/Asia	70.00	$140.00

Single Issues available, write or call.

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

FIBONACCI RANDOM NUMBER GENERATOR

NATHANIEL GROSSMAN - LOS ANGELES, CALIFORNIA

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.[1] 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.[2] 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 TEXperts can work miracles. The actual formatting used to produce the manuscript came from the LATEX 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_k from a 'seed' x_0, developing the sequence according to the formula

$$x_{k+1} = ax_k + c \pmod{M}.$$

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 $ax_k + c$ 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:

x_0	a	c	M
256	7^5	0	M_{31}
?	31421	6927	2^{16}
0	3141592653	2718281829	2^{35}

(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 $x_{k+1} = x_k + x_{k-1}$, 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

$$x_{k+1} = x_{k-17} - x_{k-5}. \tag{1}$$

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-

[1]Turing was able to generate 'truly' random numbers by reading the quantum noise in electron tubes.
[2]I will be specific to TEX, but the same ideas will apply to other formatter or formatting-capable word processor.

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 $x_0, ..., x_{17}$. 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

$$x_k = \sum_{i=0}^{17} c_i r_i^k, \qquad (2)$$

where the numbers $r_0 ..., r_{17}$ are the roots, real and complex, of the secular equation

$$r^{18} + r^{12} - 1 = 0$$

and the coefficients $c_0, ..., c_{17}$ may be chosen arbitrarily provided that the sum (2) is real-valued.[3] With optimal choices for the initial values $x_0, ..., x_{17}$, the theoretical number of elements generated by (2) before the sequence recycles can be very large: if μ 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 $p = (2^{18} - 1)2^{\mu-1}$.[4] If $\mu = 55$, say, corresponding to a seven-byte mantissa with one sign bit, then $p \approx 2^{72}$, almost 5×10^{21}.

Generator Startup

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

Mantissa width

We have defined μ, 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 μ for the particular floating-point enhancement that she is using. We will present a Forth word **f.mu** that automatically determines μ when the random number generator is loaded. For maximum portability, **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 μ were counted one too large, the only loss would be a tiny waste of processing time in seeding 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 μ 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:

```
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.

```
: F.MU=?    ( -- )
  0e         \ 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 **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

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

[3]The equation $r^{18} + r^{12} - 1 = 0$ has eighteen roots, just two of which are real, namely

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

and $-\rho$, together with 14 non-real roots. Of these last, four are arranged with ρ 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 $\rho^{-1/2}$ centered at the origin.

[4]Why the exponent μ-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.

wrapping around is carried out by the offsetting words themselves. There is no call here for a generic array with generic words, because we will work solely with an array of 18 elements and only three out of 17 possible offsets. Because our calculations are nongeneric, we can make several simplifying and optimizing definitions.

First we introduce some convenient arithmetic words.

```
: 5-   2- 2- 1- ;
: 18+  18 + ;
: 17>  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 $k - 5 \geq 0$. 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 $k - 5 \geq -17$. 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 $n \geq 5$ 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.

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

Note that **1+** is intrinsic and faster than **17 -**.

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-dimensional.' This will give the maximum cycle period. In lieu of carrying out extensive numerical experiments, we adopt the shortcut used by [Kah89]. We push a 'random' sequence of zeroes and ones into the mantissas of $x_0, ..., x_{17}$, making sure at the same time that all the exponents are 0.[5]

We explained above that the first 18 elements of the x sequence will be pushed onto the circular array as random mantissa sequences of zeros and ones with zero exponents. For this we need the rudiments of an integer linear congruential random number generator. Actually, we need only a generator with two randomly occupied states. These could be + and –, but we think of them as <0 and ≥0. The signed integers in every standard Forth fall into the two sets of 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[6] and store its least significant digits into **seed.**

```
: NEW.SEED  ( -- )
  gettime   ( d ) \ from syst
clock
  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.

[5]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].

[6]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.

```
: 0.OR.1    ( -- | r )
  sf.rand  \ n on stack
  0<
  if
    0e
  else
    1e
  then  ;
```

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.

```
: ONE.RANDOM.REAL.MOD.1
  ( -- | r )
  0e         \ zero exponent
  f.mu @ 0 do\ every mant bit
    2e f/    \ slide to right
    0.or.1   \ ran bit, left
    f+       \ push onto left
  loop
  fdup 1e f< not
             \ 1 or bigger?
  if         \ >=1 and <2
    1e f-    \ reduce mod 1
  then  ;
```

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.

```
: 18.RANDOM.REALS.MOD.1 ( -- )
  18 0 do
    one.random.real.mod.1
    i pisa f!
  loop  ;
```

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_{-17}, x_{-16}, ..., x_0. Then the calculation begins with the latest subscript set to zero.

```
: INITIALIZE.PISA.PIE  ( -- )
  18.random.reals.mod.1
  0 subscript !  ;
```

Initialization will be automatic upon loading:

```
initialize.pisa.pie
```

To start the clock anew, substitute the following word:

```
: INITIALIZE.NEW.PISA.PIE
  (--)
  new.seed
  initialize.pisa.pie  ;
```

We need a word to carry out subtraction of two modulo 1 reals and express the answer as a real modulo 1. When both r_1 and r_2 lie in [0,1), then the difference $r_1 - r_2$ must lie in (–1,1). If it lies in (–1,0), then the reduction modulo 1 is carried out not by division but simply by adding one.

```
: F-.MOD.1
  ( r1 r2 -- r1-r2 mod 1 )
  f-
  fdup f0<
  if          \ -1 < diff < 0
    1e f+     \ add 1
  then  ;
```

Now we have arrived at the generator itself.

```
: FIBRAND.MOD.1  ( -- | r )
  subscript @
  dup          \ ( k k )
  offset.by.17\ same as +1
  dup >r
  pisa f@      \ x(k-17)
  offset.by.5
  pisa f@      \ x(k-5)
  f-.mod.1     \ x(k-17)-x(k-5)
  fdup
  r@ pisa f! \ store x(k+1)
  r>
  subscript !  ;  \update
```

This generator will not be an end in itself. Therefore, the random real modulo 1 is left atop the floating-point stack for use in the main computation.

Documenting in Style

Dr. C.H. Ting recommended at the 1988 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*.[7] 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 UR/Forth, 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. This

[7]See remarks by Glen B. Haydon, *Forth Dimensions* Vol. X, No. 3, p. 13.

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 TEX 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 TEX 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 TEX. 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 backslashed words would work the expected way with Forth compilation. For TEX 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 TEX 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, TEX composition is best done in small morsels, with the latest batch put through the TEX compiler to catch the bugs. TEX does not crash as Forth does, but even a TEXpert will sometimes write formatting that sinks under an overload of error messages. Writing and testing Forth code and preparing a manuscript for TEX 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 TEX 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,[8] and I usually have two or three open in windows. I set up one 'text' file to contain the debugged code and the interwoven 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 TEX font-formatting commands.[9] The result is a file always ready for TEX 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 TEX ilk by **comment:** and **comment;** and making a small, technically required alteration to the first line of the TEX 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, TEX uses the underscore (e.g., under_score) 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 TEX, but I found that with care I could avoid using it where TEX 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

[Bro87] L. Brodie, *Starting Forth*, 2nd edition, Prentice-Hall, 1987.

[Che85] W. Cheney and D. Kincaid, *Numerical Mathematics and Computing*, Brooks/Cole, 1985.

[Jans66] B. Jansson, *Random Number Generators*, Stockholm: Victor Petersons Bokindustri Aktiebolag, 1966.

[Kah89] D. Kahaner, C. Moler, and S. Nash, *Numerical Methods and Software*, Prentice-Hall, 1989.

[Knu69] D.E. Knuth, *The Art of Computer Programming*, Vol. 2, Addison-Wesley, 1969.

[Mod87] D.T. Modianos, R.C. Scott, and L.W. Cornwell, "Testing Random Number Generators," *BYTE*, January 1987, p. 175.

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

[8]And it can restore all of those files to the 'desktop' at the start of a new session.
[9]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.
[10]Dr. Ting already has exploited this 'vertical' format in his *F-PC Technical Reference Manual*.

HARVARD SOFTWORKS

NUMBER ONE IN FORTH INNOVATION

(513) 748-0390 P.O. Box 69, Springboro, OH 45066

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.)

HS/FORTH with FOOPS - The only flexible full multiple inheritance object oriented language under MSDOS!

Seeing is believing, OOL's really are incredible at simplifying important parts of any significant program. So naturally the theoreticians drive the idea into the ground trying to bend all tasks to their noble mold. Add on OOL's provide a better solution, but only Forth allows the add on to blend in as an integral part of the language and only HS/FORTH provides true multiple inheritance & membership.

Lets define classes BODY, ARM, and ROBOT, with methods MOVE and RAISE. The ROBOT class inherits:

```
  INHERIT> BODY
  HAS> ARM RightArm
  HAS> ARM LeftArm
```

If Simon, Alvin, and Theodore are robots we could control them with:

```
Alvin 's RightArm RAISE        or:
+5 -10 Simon MOVE              or:
+5 +20 FOR-ALL ROBOT MOVE
```

Now that is a null learning curve!

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 PCDOS, 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,ega,vga, 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, decompiler; music; compile 40,000 lines per minute, stacks; file search paths; formats into strings.

PERSONAL LEVEL **$245.**
software 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.**
hardware floating point - data structures for all data types from simple thru complex 4D var arrays - operations complete thru complex hyperbolics; turnkey, seal; interactive dynamic linker for foreign subroutine libraries; round robin & interrupt driven multitaskers; dynamic string manager; file blocks, sector mapped blocks; x86&7 assemblers.

PRODUCTION LEVEL **$495.**
Metacompiler: DOS/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.**
286FORTH or **386FORTH** **$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.
BTRIEVE for HS/FORTH (Novell) **$199.**
ROMULUS HS/FORTH from ROM**$ 95.**
FFORTRAN translator/mathpak **$ 75.**
Compile Fortran subroutines! Formulas, logic, do loops, arrays; matrix math, FFT, linear equations, random numbers.

FORTH IN OPTIMAL CONTROL

J.B. HO, P.Y. KOKATE, M. HUDA, R. HASKELL, N.K. LOH
ROCHESTER, MICHIGAN

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 $u = -\Sigma k_i x_i$, where the k_i'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:

$$\dot{\underline{x}}(t) = A\underline{x}(t) + B\underline{u}(t),$$

$$\underline{y}(t) = C\underline{x}(t), \qquad (1)$$

where, $\underline{x}(t) \varepsilon R^n$ is the state vector, $\underline{u}(t) \varepsilon R^m$ is the input vector, $\underline{y}(t) \varepsilon R^p$ 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.,

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

where rank[.] denotes the rank of [.].

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

$$J = {}_0\!\int^{\infty} [\underline{x}^T(t)Q\underline{x}(t) + \underline{u}^T R\underline{u}(t)]dt \qquad (3)$$

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

$$\underline{u}_{opt}(t) = -R^{-1}B^TK\underline{x}(t), \qquad (4)$$

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

$$KA + A^TK - KBR^{-1}B^TK - Q = 0. \qquad (5)$$

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 $\underline{x}(t)$ can be obtained directly from (1) as $\underline{x}(t)=[C]^{-1}\underline{y}(t)$. 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.,

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

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

$$\begin{bmatrix} \dot{z}(t) \\ \ddot{z}(t) \\ \dot{\theta}(t) \\ \ddot{\theta}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -3.88 & -0.124 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 9.23 & 23.62 & 0 \end{bmatrix} \begin{bmatrix} z(t) \\ \dot{z}(t) \\ \theta(t) \\ \dot{\theta}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 2.77 \\ 0 \\ -6.6 \end{bmatrix} u(t)$$

$$\overset{\Delta}{=} A\underline{x}(t) + Bu(t), \qquad (7)$$

where $\underline{x}^T = [x_1\ x_2\ x_3\ x_4] = [z\ \dot{z}\ \Theta\ \dot{\Theta}]$,

$z(t)$	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

$$J = \int_0^\infty [10x_1^2(t) + 0.1x_3^2(t) + u^2(t)]dt. \quad (8)$$

The optimal control which minimizes J is given by

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

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

$$u_{opt}(t) = 0.2161x_1(t) - 0.4398x_2(t) + 0.59x_3(t) + x_4(t), \quad (10)$$

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

$$x_4(t) = 0.4311x_{4_{old}} + 6.3324(x_3(t) - x_{3_{old}}) \quad (11)$$

where $x_{i_{old}} = x_i(t-\Delta)$, Δ 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,

$$u_{opt}(t) = 0.4237x_1(t) - 1.7276x_2(t) + 0.6941x_3(t) + x_4(t), \quad (12)$$

where $x_4(t)$ is calculated as,

$$x_4(t) = 0.5384x_{4_{old}} + 8.008(x_3(t) - x_{3_{old}}).(13)$$

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: Marlin 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

	Average No. copies/ issue during preceding 12 months	Actual No. copies of single issue nearest to filing date
A. Total no. copies printed:	2750	2500
B. Paid/requested circulation:		
1. Sales:	0	0
2. Mail subscription	2187	1918
C. Total paid/requested circulation:	2187	1918
D. Free distribution by mail, carrier or other means: samples, complimentary and other free copies:	47	45
E. Total distribution:	2234	1963
F. Copies not distributed:		
1. Office use, left over, unaccounted, spoiled after printing:	516	537
2. Return from news agents:	0	0
G. TOTAL:	2750	2500

11. I certify that the statements made by me above are correct and complete.
/s/ Georgiana F. Shepherd

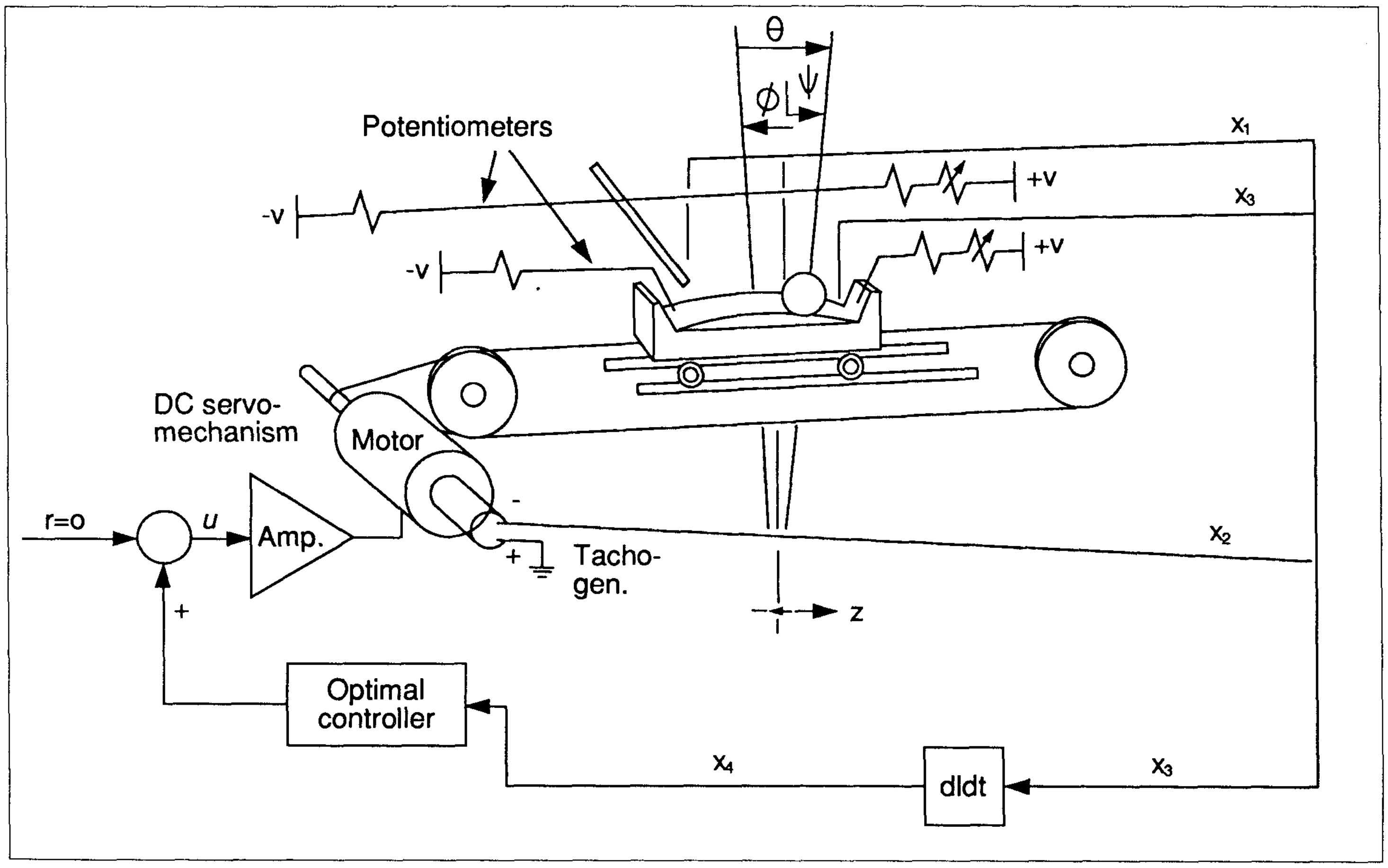


Figure One. Schematic diagram of the ball-balancing system.

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,

```
DUP 2/ DUP 2/ DUP 2/
2/ 2/ - + +
```

is equivalent to

$(1 + (0.5 + (0.25 - 0.03125)))X2 = 1.71875X2$,

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

Conclusion

The response of the system is excellent in both implementations. Settling 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.)

```
    0
Forth in Optimal Control                              reh 01Jan80 \

J.B. Ho, P.Y. Kokate, M. Huda, R. Haskell, N.K. Loh

Center for Robotics and Advanced Automation
School of Engineering and Computer Science
Oakland University
Rochester, Michigan 48309
```

```
    1
\                                                      reh 15Feb89 \
 2VARIABLE SVBX
 2VARIABLE SVAX
 2VARIABLE BASE-ADD
 HEX
 A000 BASE-ADD !
 DECIMAL
 CODE !BUFF      ( data\offset -- )
    AX SVAX #) MOV BX SVBX #) MOV BASE-ADD #) AX MOV
    AX ES MOV BX POP AX POP ES: AX 0 [BX] MOV SVAX #) AX
    MOV SVBX #) BX MOV NEXT END-CODE
 CODE @BUFF      ( add -- data )
    AX SVAX #) MOV BX SVBX #) MOV BASE-ADD
    #) AX MOV AX ES MOV BX POP ES: 0 [BX] AX MOV
    1PUSH SVAX #) AX MOV SVBX #) BX MOV NEXT
 END-CODE
```

```
    2
\                                                      reh 15Feb89 \
 CODE C!BUFF     ( data\offset -- )
    AX SVAX #) MOV BX SVBX #) MOV BASE-ADD #) AX MOV
    AX ES MOV BX POP AX POP ES: AL 0 [BX] MOV SVAX #) AX
    MOV SVBX #) BX MOV NEXT
 END-CODE

 CODE C@BUFF     ( add -- data )
    AX SVAX #) MOV BX SVBX #) MOV BASE-ADD
    #) AX MOV AX ES MOV BX POP ES: 0 [BX] AX MOV
    AH AH SUB 1PUSH SVAX #) AX MOV SVBX #) BX MOV NEXT
 END-CODE
```

```
    3
                                                             HJB 01Jan80
VARIABLE X1
VARIABLE X2
VARIABLE X3
VARIABLE X4
VARIABLE X3_OLD
VARIABLE X4_OLD

0 X3_OLD !
0 X4_OLD !
```

```
    4
                                                             reh 15Feb89
HEX
: ADC     ( Chan -- Data )
   725 C!BUFF 0 726 C!BUFF \ Select channel. Start conversion.
   BEGIN 724 C@BUFF 7F > UNTIL \ Check for end of conversion.
   725 @BUFF ;
: DAC     ( Data\Chan -- )
   2* SWAP
   -800 MAX 7FF MIN      \ Check for 12 bit saturation.
   DUP -ROT 2/ 2/ 2/     \ Separate into H and L bytes.
   2/ 2/ 2/ 2/ 2/ OVER
   721 + C!BUFF          \ Output H byte.
   720 + C!BUFF ;        \ Output L byte and start conversion
```

```
    5
                                                            Hreh 15Feb89
DECIMAL
: U.CAL ( -- U_DATA )
   2 ADC X1 ! 3 ADC X2 ! 5 ADC X3 !
   X4_OLD @ 431 *D X3 @ X3_OLD @ - 6333 *D D+ 1000 M/MOD
   SWAP DROP DUP X4 ! X1 @ 216 *D X2 @ -440 *D D+
   X3 @ 583 *D D+ 1000 M/MOD SWAP DROP +
   X3 @ X3_OLD ! X4 @ X4_OLD ! ;

: OPT ( -- )
   0 1 DAC 128 1828 C!BUFF
   BEGIN  U.CAL DUP 408 > OVER -408 < OR
        IF 3 * 10 / 1 DAC
        ELSE 1 DAC THEN
   AGAIN  ;
```

Figure Two. Forth words used to balance the ball using F83 with an IBM PC and Tecmar data acquisition board.

```
    60
(                                         reh 15Feb89
  HEX
  1004 1C !
  50 1E !
  1060 22 !
  FORGET TASK
  1080 DP !
  DECIMAL

: \   ( -- )
   BLK @
   IF
     >IN @ 64 / 1+ 64 *
   ELSE #TIB @
   THEN >IN ! ; IMMEDIATE

```

```
    61
\                                         reh 15Feb89
HEX
B000 CONSTANT REG
REG  4 + CONSTANT PORTB    \ Output Port B
REG 30 + CONSTANT ADCTL    \ A/D Control Register
REG 31 + CONSTANT ADR1     \ A/D Result Register 1
REG 32 + CONSTANT ADR2     \ A/D Result Register 2
REG 33 + CONSTANT ADR3     \ A/D Result Register 3
REG 39 + CONSTANT OPTION   \ System Configuration Options

VARIABLE X3
VARIABLE X3_OLD
VARIABLE X4_OLD
0 X3_OLD !
0 X4_OLD !

```

```
    62
\                                         reh 15Feb89
CODE TSTH.CCF  \ Test ADC Conversions Complete Flag High ( -- )
   3C C,                  ( PSHX )
   CE C, B0 C, 00 C,      ( LDX #$B000 )
   1F C, 30 C, 80 C, FC C,  ( L1:  BRCLR $30,X $80 L1 )
   38 C,                  ( PULX )
   7E C, FE C, 4A C,      ( JMP NEXT )
END-CODE

: ADC.ON      ( -- )
    80 OPTION C! 5 0 DO LOOP ;

: ADC.MULTI    ( -- )
    ADC.ON 10 ADCTL C! ;


```

```
    63
\                                         reh 15Feb89
CODE 3.1MSEC   ( -- )          ( 3.1 MSEC DELAY )
  3C C,                        ( PSHX )
  CE C, 04 C, 06 C,            ( LDX #$0406 )
  09 C,                        ( L1 : DEX )
  26 C, FD C,                  ( BNE L1 )
  38 C,                        ( PULX )
  7E C, FE C, 4A C,            ( JMP NEXT )
END-CODE
```

```
    64
\                                         reh 15Feb89
: X1.CAL       ( -- n )
    ADR1 C@ 80 -  NEGATE 2/ DUP 2/ 2/ 2/ DUP 2/ 2/ + - ;

: X2.CAL       ( -- n )
    ADR2 C@ 80 - NEGATE DUP 2/ DUP 2/ DUP 2/ 2/ 2/ - + + ;

: X3.CAL       ( -- n )
    ADR3 C@ 86 - NEGATE DUP X3 ! 2/ DUP 2/ 2/ DUP 2/ + + ;

: X4.CAL       ( -- n )
    X4_OLD @ 2/ DUP 2/ 2/ 2/ 2/ +
    X3 @ X3_OLD @ - 8 * +
    DUP X4_OLD ! X3 @ X3_OLD !  ;
```

```
    65
\                                        Hreh 15Feb89
80 PORTB C!

: BALANCE
    ADC.MULTI
    BEGIN
      TSTH.CCF X1.CAL  X2.CAL -  X3.CAL + X4.CAL +
      80 + DUP E4 > OVER 1C < OR
      IF  2/ PORTB C!
      ELSE  PORTB C!
      THEN
      10 ADCTL C! 3.1MSEC
    AGAIN  ;

DECIMAL
```

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—an 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 processor (CPU) RAM. Forth runs in the CPU RAM obtained from a 32K memory expansion card (required to use Forth on the 99/4A).

System Synonyms

TI-Forth[1] 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 to the stack.
`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 `VFILL`, 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 `VARIABLE`. 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 `VVARIABLE`, since `FORGET` alone does not affect

(Continued on page 30.)

```
Addr  0 1  2 3  4 5  6 7   ASCII
1400  0002 0004 0006 0008  ........
1408  000A 000C 000E 0010  ........
1410  0012 0014 0016 0018  ........
1418  001A 001C 001E 0020  .......
1420  0022 0024 0026 0028  .".$.&.(
1428  002A 002C 002E 0030  .*.,...0
1430  0032 0034 0036 0038  .2.4.6.8
1438  003A 003C 003E 0040  .:.<.>.@
1440  2021 2223 2425 2627   !"#$%&'
1448  2829 2A2B 2C2D 2E2F  ()*+,-./
1450  3031 3233 3435 3637  01234567
1458  3839 3A3B 3C3D 3E3F  89:;<=>?
1460  4041 4243 4445 4647  @ABCDEFG
1468  4849 4A4B 4C4D 4E4F  HIJKLMNO
1470  5051 5253 5455 5657  PQRSTUVW
1478  5859 5A5B 5C5D 5E5F  XYZ[\]^_
1480  6061 6263 6465 6667  `abcdefg
1488  6869 6A6B 6C6D 6E6F  hijklmno
1490  7071 7273 7475 7677  pqrstuvw
1498  7879 7A7B 7C7D 7E7F  xyz{|}~.
```

Figure One. Representative screen dump of VDP RAM.

1. O'Hagen, L., Tietz, L., and Yantis, J.T. *TI-Forth Instruction Manual*, Texas Instruments, Inc., 1983.

```
SCR #110
( FIG-Forth: VDP arrays and variables   SCR#1     HHR 1/13/87 )

BASE->R   HEX              ( 8664 bytes of VDP RAM available )
1400 VARIABLE VPTR         ( Usuable VDP RAM starts at hex 1400 )

: VALLOT  ( n --- )  VPTR +!   ;    ( Advances VDP RAM pointer)

: VC! ( b vaddr --- )   VSBW   ;    ( More appropriate name )

: V!  ( n vaddr --- )  ( 2 VMBW transfers a word from the top )
     SP@ 2+ SWAP 2 VMBW DROP ;       ( of the stack to VDP RAM )

          ( Address of named variable is stored as a constant )
: VVARIABLE ( n --- ) VPTR @      ( =CELLS forces pointer even )
     =CELLS DUP DUP 2+ VPTR !     ( Pointer VPTR is incremented )
     CONSTANT V!   ;      -->     ( Store value in variable )

SCR #111
( FIG-Forth VDP arrays and variables   SCR#2      HHR 1/13/87 )

: VC@ ( vaddr --- n ) VSBR   ;       ( More appropriate name )

: V@ ( vaddr --- n )                 ( Analogous to @ )
    0 SWAP SP@ 2+ 2 VMBR   ;
 ( VMBR transfers a word from VDP RAM to the top of the stack )

: VC, ( c --- ) VPTR @ VSBW 1 VPTR +!   ;
: V,  ( n --- ) VPTR @ V!   2 VPTR +!   ;
: V+! ( n addr --- ) DUP V@ ROT + SWAP V!   ;

: RESETV ( --- <name> )  ( Recovers VDP RAM-  resets pointer )
     [COMPILE] ' CFA EXECUTE VPTR !   ;

R->BASE

SCR #112
( Alternative V! definitions                          HHR 1/13/87 )

HEX     ( All use VSBW, a single byte transfer from the stack )

: V!    ( n vaddr --- )  DUP ROT
     DUP SWPB FF AND ROT VSBW  ( SWPB swaps bytes conveniently )
     SWAP 1+ VSBW   ;

: V!    ( n vaddr --- )
     DUP ROT SP@ C@ ROT VSBW      ( Uses C@ instead of SWPB to )
     SWAP 1+ VSBW   ;             (         select correct byte )

: V!  ( n vaddr --- )   SWAP PAD !     ( Similar to above but )
     DUP   PAD C@ SWAP VSBW            ( uses PAD for storage )
     PAD 1+ C@ SWAP 1+ VSBW ;
```

```
SCR #113
( Alternative V@ definitions                        HHR 1/13/87 )

HEX      ( All use VSBR, a single byte transfer to the stack )
         ( All use + or OR to create a Word from two bytes )

: V@    ( vaddr --- n ) DUP VSBR
   8 SLA SWAP 1+ VSBR +     ;     ( 8 SLA  shifts left 1 byte )

: V@    ( vaddr --- n ) DUP VSBR                ( OR is used as an )
   8 SLA SWAP 1+ VSBR OR    ;                   ( alternative to + )

: V@ ( vaddr --- n )    DUP
   VSBR 100 * SWAP      ( * in place of left shift of 1 byte )
   1+ VSBR OR     ;


SCR #114
( VDUMP     SCR# 1                                  HHR 1/15/87 )
0 VARIABLE CNTR    0 VARIABLE TEMP

: PRT 0 <# # # # # #> TYPE    ;    ( VDP RAM addr stored at PAD )
  ( Lines of data stored at PAD + 2 to PAD + 9 temporarily )

: GETDATA   PAD @ PAD 2+ 8 VMBR    ;
  ( Transfers data from VDP addr to PAD + 2 )

: STOP   ?KEY DUP 2 =        ( Any key stops & resumes printing )
    IF TEMP @ BASE ! 0 CNTR ! QUIT THEN
       IF KEY DROP THEN   ;           ( FCTN 4 terminates VDUMP )

: FILTER    8 0 DO PAD 2+ I + C@     ( Changes byte values < 32 )
     DUP 32 < SWAP   126 > OR ( & > 126 to ASCII 46, prints . )
     IF 46 PAD 2+ I + C! THEN LOOP    ;       -->

SCR #115
( VDUMP   SCR# 2                                    HHR 1/15/87 )
: PRTLINE 8 0 DO PAD 2+ I + @ PRT SPACE 2 +LOOP    ;

: PRT-ASCII   SPACE   PAD 2+ 8 TYPE CR    ;
    ( PRT-ASCII prints filtered ASCII characters )
: HEADER    CR ." Addr  0 1  2 3  4 5  6 7    ASCII" CR     ;

: VDUMP ( vaddr cnt --- )    ( TEMP holds previous base value )
     CR CR HEADER   BASE @ TEMP !   HEX
     8 + 8 / SWAP                ( Print only full 8 byte line )
     8 / 8 * PAD !    ( start only at an addr divisible by 8 )
     0 DO GETDATA PAD @ PRT 2 SPACES     ( PRT prints address )
     PRTLINE    FILTER   PRT-ASCII        ( Prints header each )
     CNTR @ 19 > IF HEADER 0 CNTR ! THEN        (     20 lines )
     1 CNTR +!   8 PAD +!   STOP       ( Increments addr by 8 )
     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 ____________________

VOLUME XI INDEX

MIKE ELOLA - SAN JOSE, CALIFORNIA

Panel Management Facility

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:

AFT OF FLORIDA, INC.
Advanced Functions Technology of Florida, Inc.

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

Text-based applications only. Requires DOS 3.0 or higher.

Total control with *LMI FORTH*™

For Programming Professionals: an expanding family of compatible, high-performance, compilers for microcomputers

For Development:
Interactive Forth-83 Interpreter/Compilers for MS-DOS, OS/2, and the 80386

- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 500 page manual written in plain English
- Support for graphics,floating point, native code generation

For Applications: Forth-83 Metacompiler

- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8088, 68000, 6502, 8051, 8096, 1802, 6303, 6809, 68HC11,34010, V25, RTX-2000
- No license fee or royalty for compiled applications

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone Credit Card Orders to: (213) 306-7412
FAX: (213) 301-0761

Generic File Operations
    The Best of GEnie, Vol 10, Issue 2, pg 29
Documenting Source Code — see Source Code Styling Conventions
Education
    Computer Science and Forth
        The Best of GEnie, Vol 10, Issue 3, pg 35
    Forth Textbooks
        Letter, Vol 10, Issue 4, pg 6
Forth Interest Group (FIG)
    Chapters
        The Value of Chapters, Vol 10, Issue 5, pg 36
Forth's Marketing and Promotion
    The Greening of Forth, Vol 10, Issue 6, pg 40
    WISC and the Forth Dilemma, Vol 10, Issue 3, pg 12
Fractals
    Fractal Landscapes, Vol 9, Issue 1, pg 12
        Letter, Vol 10, Issue 1, pg 5
Games and Recreation
    Capture!, Vol 10, Issue 6, pg 20
Implementing Forth
    CPU-Specific Optimization
        Using Registers in Data Stacks, Vol 10, Issue 4, pg 19
    on the Macintosh
        Best of GEnie, Vol 10, Issue 4, pg 29
Interpreters
    Inner interpreters
        A Faster Next Loop, Vol 9, Issue 6, pg 16
            Letter, Vol 10, Issue 1, pg 6
Libraries
    Creating Public-Domain Libraries
        Letter, Vol 10, Issue 4, pg 6
    Object-Oriented Libraries
        Designing Data Structures, Vol 10, Issue 5, pg 19
Linear Automata — see Algorithms
Local Variables
    Letter, Vol 9, Issue 5, pg 5
        Letter, Vol 10, Issue 5, pg 5
Maintainability of Code — see Source Code Styling Conventions
Memory
    Extended Addressing
        Relocatable F83 for the 68000, Vol 9, Issue 6, pg 20
            Letter, Vol 10, Issue 2, pg 7
Microprocessors
    Super 8, FRISC
        The Best of GEnie, Vol 10, Issue 3, pg 35
    Multitasking
        68000 Microprocessor
            Letter, Vol 10, Issue 2, pg 7
Object-Oriented Forth for Portability — see Data Structure Design
Object-Oriented Programming — see Data Objects & Associated Methods
Portability Issues
    The Best of GEnie, Vol 10, Issue 1, pg 27
Portability of Data — see Data Structure Design
Programming Environments

1990 ROCHESTER FORTH CONFERENCE ON EMBEDDED SYSTEMS

June, 1990
University of Rochester
Rochester, New York

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)

BEST OF GENIE

GARY SMITH - LITTLE ROCK, ARKANSAS

News from the GEnie Forth RoundTable—As promised at the close of last issue's column, this time we will re-examine ForthNet. ForthNet is a virtual Forth network that links designated message bases of several computer bulletin boards and information services, in an attempt to provide greater distribution of Forth-related information. It is provided courtesy of the sysops of its various links. Readers of this column may recall that ForthNet was talked of as a dream yet unfulfilled in *Forth Dimensions* (X/4). At that time, there was cause to wonder if it would even survive. Some questions still remain—serious questions—but as the evolution continues, it is increasingly evident that a purpose is being served; and where there is purpose, there is often the will to succeed.

For historical perspective, the term ForthNet was coined in early 1986 when the Forth Interest Group was still searching for a home for its electronic message base that would provide easier and greater access than the existing FIGTree BBS. Discussions at that time were being conducted on Delphi, and consisted primarily of establishing guidelines and expectations. After the GEnie Forth RoundTable was established and I was included as one of the sysops, I again began to explore the possibility of linking with other Forth message bases. With the encouragement of lead GEnie sysop Dennis Ruffer and considerable help from Jack Woehr, we established the first link between the GEnie Forth RoundTable and the forth.conf topic of the Well, of which Jack is the Fairwitness. Later, I brought forth.conf from the Wetware Unix BBS into this link.

Concurrently, Jerry Shifrin had established the East Coast Forth Board as a premier source of Forth information exchange, and had linked it via PCBoard to the West Coast, and later the North Coast, Forth Boards. West Coast and North Coast have both since been dissolved by their operators, but in their places have risen the British Columbia Forth Board and the Real-Time Control Forth Board, both linked to Jerry's East Coast Forth Board. Also linked to this group is Metrolink Bulletin Boards, which themselves maintain a loose network similar to Opus and Fido.

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

Gary Smith
well!gars@lll-winken.arpa (also for jax)
gars@wet.UUCP
gars@chinet.UUCP

* * *

The following are excerpts from recent letters and e-mail :

From: Paulo A. D. Ferreira
Inesc Norte (CG & CAD)
Largo de Mompilher 22
4000 Porto
Portugal
Paulo Ferreira p_ferreira@inescn.rccn.pt
To: Gary Smith
well!gars@lll-winken.arpa
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

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 part of a system for nuclear medicine.

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

By all means—Please start with the Forth conference here on the WELL, where you reached me. The WELL is accessible via X.25—leave e-mail to Eric Fair (fair@well) for details, if you need access help. Eric is the WELL's resident usenet/UUCP/inet guru. Thanks for the info, it is super to hear from a distant Forther. I will forward your message to jax, also, and see if he can put you on his FIG Chapter e-mail list.—gars

From: Ted Rofe
munnari!usage.csd.unsw.oz.au!tedr
To: Gary Smith
well!gars

Gary, I just received the May/June 1989 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

To: Ted Rofe
From: Gary Smith
Subject: JFAR

Ted, thanks for the kudos. It is great to know I'm not writing to a vacuum. Yes, *JFAR finally* came to pass. I have had mine about three weeks, so you should receive yours shortly.

Re: Dr. Ting. He is also one of the nicest people it has been my great pleasure to talk to since becoming involved in Forth. That says a *lot* because, in my opinion, Forth users are a pretty class act, despite our collective reputation as maverick rowdies

Please stay in touch, Ted. We want *very* much to get Australia involved electronically. —Gars

From: Lance Collins
To: Gary Smith

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

From: Gary Smith
To: Lance Collins
> Thanks for the messages...

No. Thank you. Your continued contact has helped sustain momentum. This ForthNet concept is not totally endorsed by all... yet.

> can get to Usenet mail but... he is not the mailbox...

Why not? Even a month would help determine the value of Usenet mail. There is an added reason to test Usenet. It appears (1) GEnie will support Usenet mail via a gateway. (2) I am now attempting to solicit some help in the way of someone who will edit a packet of ForthNet messages for me to post to comp.lang.forth. We can presume your having access to this Usenet/ForthNet port would cut down on your need to carry so much traffic via PCBoard.

> Please mail the registration card for me.
Done.

> You said... you only found out about our BBS recently.

I must confess that, having seen the BBS note in the chapter listings, I gave it little credence until its activity was confirmed by you.

> 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
(SECRETAIRE)

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."

MAKE YOUR SMALL COMPUTER
THINK BIG
(We've been doing it since 1977 for IBM PC, XT, AT, PS2, and TRS-80 models 1, 3, 4 & 4P.)

FOR THE OFFICE — Simplify and speed your work with our outstanding word processing, database handlers, and general ledger software. They are easy to use, powerful, with executive-look print-outs, reasonable site license costs and comfortable, reliable support. Ralph K. Andrist, author/historian, says: "**FORTHWRITE** lets me concentrate on my manuscript, not the computer." Stewart Johnson, Boston Mailing Co., says: "We use **DATAHANDLER-PLUS** because it's the best we've seen."

MMSFORTH System Disk from $179.95
Modular pricing — Integrate with System Disk only what you need:

FORTHWRITE - Wordprocessor	$99.95
DATAHANDLER - Database	$59.95
DATAHANDLER-PLUS - Database	$99.95
FORTHCOM - for Communications	$49.95
GENERAL LEDGER - Accounting System	$250.00

MMSFORTH

MILLER MICROCOMPUTER SERVICES
61 Lake Shore Road, Natick, MA 01760
(508/653-6136, 9 am - 9 pm)

FOR PROGRAMMERS — Build programs FASTER and SMALLER with our "Intelligent" MMSFORTH System and applications modules, plus the famous MMSFORTH continuing support. Most modules include source code. Ferren MacIntyre, oceanographer, says: "Forth is the language that microcomputers were invented to run."

SOFTWARE MANUFACTURERS — Efficient software tools save time and money. MMSFORTH's flexibility, compactness and speed have resulted in better products in less time for a wide range of software developers including Ashton-Tate, Excalibur Technologies, Lindbergh Systems, Lockheed Missile and Space Division, and NASA-Goddard.

MMSFORTH V2.4 System Disk from $179.95
Needs only 24K RAM compared to 100K for BASIC, C, Pascal and others. Convert your computer into a Forth virtual machine with sophisticated Forth editor and related tools. This can result in 4 to 10 times greater productivity.

Modular pricing — Integrate with System Disk only what you need:

EXPERT-2 - Expert System Development	$69.95
FORTHCOM - Flexible data transfer	$49.95
UTILITIES - Graphics, 8087 support and other facilities.	

and a little more!

THIRTY-DAY FREE OFFER — Free MMSFORTH GAMES DISK worth $39.95, with purchase of MMSFORTH System. CRYPTOQUOTE HELPER, OTHELLO, BREAKFORTH and others.

Call for free brochure, technical info or pricing details.

SIG FORTH '90

CALL FOR PAPERS

acm

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

REFERENCE SECTION

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

Andrew 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
Watervleit, 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)

(Reference Section continued)

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)

CompuServe
For information, call 800-848-8990

- Creative Solutions Conference
  Type !Go FORTH
  SysOps: Don Colburn, Zach Zachariah, Ward McFarland, Jon Bryan, Greg Guerin, John Baxter, John Jeppson
- Computer Language Magazine Conference
  Type !Go CLM
  SysOps: Jim Kyle, Jeff Brenton, Chip Rabinowitz, Regina Starr Ridley

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)

PC Board BBS's devoted to Forth (ForthNet links)*

- East Coast Forth Board
  703-442-8695
  SysOp: Jerry Schifrin
- British Columbia Forth Board
  604-434-5886
  SysOp: Jack Brown
- Real-Time Control Forth Board
  303-278-0364
  SysOp: Jack Woehr
- Melbourne FIG Chapter
  Lance Collins
  (03) 299-1787 in Australia
  61-3-299-1787 international

(Continued from page 18)

References

1. D. E. Kirk, *Optimal Control Theory*, Prentice Hall, 1970.
2. K.C.Cheok, N.K.Loh and R.R.Beck, "Microprocessor-Based State Estimators and Optimal Controllers", Proc. 23rd Midwest Symp. Circuits and Systems, Univ. of Toledo, Toledo, OH, pp. 318-324, Aug 1980.
3. B.D.O. Anderson and John B. Moore, *Optimal Filtering*, Prentice Hall, 1979.
4. K.C.Cheok and N.K.Loh, "A Ball Balancing Demonstration of Optimal and Disturbance-Accommodating Control", *IEEE Control Systems Magazine*, pp. 54-57, 1987.

This paper was presented by R. Haskell at the 1988 Real-Time Programming Convention in Anaheim, California,

Advertisers Index

Advanced Functions Technology	26
Concept 4	6
Dash, Find Associates	37
Forth Interest Group	44
Harvard Softworks	15
Institute for Applied Forth Research	8, 9, 27
Laboratory Microsystems	26
Miller Microcomputer Services	30
Mountain View Press	40
Next Generation Systems	35
Saelig Company	37
Silicon Composers	2
SIG Forth	31

Chapter Coordinator's Kitchen

JACK WOEHR - 'JAX' ON GEnie

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?

It seems to me that there are several opportunities to maintain the free character of the Forth Interest Group while providing more services from the central organization at little or no cost; all that is required is the dedicated participation of the chapters. Let me know what you think. I'll be down the hall in kitchen...

(Continued from page 9.)

Forth-83 will be applicable to ANS Forth. In my opinion, the most important improvements over Forth-83 are the increased implementation options available for ANS Forth and the greater variety of computers that can efficiently host the language. At the same time, Forth-83 programs will need only slight modifications to run on 16-bit implementations of ANS Forth.

John Hayes is the author of several Forth articles and a key figure in the VLSI Forth microprocessor project at

Table One. Additions to Required Word Set.

Word	*Purpose of Change*
`2>R 2DROP 2DUP 2OVER 2R> 2SWAP`	Completes set of cell-pair words
`2! 2@ 2*`	Essential
`C,`	Completes set of character words
`" CHAR [CHAR]`	String and character literals
`RECURSE UNLOOP`	Improves control flow
`EVALUATE`	The Forth interpreter
`CELL+ CELLS`	Portable addressing
`BYTE+ BYTES`	Portable addressing
`ALIGN REALIGN`	Portable address alignment
`S>D D>S`	Portable conversion
`POSTPONE`	Replaces `COMPILE [COMPILE]`
`MOVE`	Replaces `CMOVE CMOVE>`

Table Two. Deletions from Required Word Set.

Word	*Purpose of Change*
`BLOCK BLK BUFFER`	Moved to Block Extension Word Set
`FLUSH LOAD SAVE-BUFFERS UPDATE`	Moved to Block Extension Word Set
`VOCABULARY`	Moved to Vocabulary Extension Set
`2+ 2-`	Obsolete
`FORTH-83`	Obsolete
`COMPILE [COMPILE]`	Incompatible with native code
`CMOVE CMOVE>`	Inefficient
`PICK ROLL`	Clumsy and inefficient
`PAD`	Unsafe
`FORGET`	Moved to Reserved Word Set

NGS FORTH

A FAST FORTH, OPTIMIZED FOR THE IBM PERSONAL COMPUTER AND MS-DOS COMPATIBLES.

STANDARD FEATURES INCLUDE:

- •79 STANDARD
- •DIRECT I/O ACCESS
- •FULL ACCESS TO MS-DOS FILES AND FUNCTIONS
- •ENVIRONMENT SAVE & LOAD
- •MULTI-SEGMENTED FOR LARGE APPLICATIONS
- •EXTENDED ADDRESSING
- •MEMORY ALLOCATION CONFIGURABLE ON-LINE
- •AUTO LOAD SCREEN BOOT
- •LINE & SCREEN EDITORS
- •DECOMPILER AND DEBUGGING AIDS
- •8088 ASSEMBLER
- •GRAPHICS & SOUND
- •NGS ENHANCEMENTS
- •DETAILED MANUAL
- •INEXPENSIVE UPGRADES
- •NGS USER NEWSLETTER

A COMPLETE FORTH DEVELOPMENT SYSTEM.

PRICES START AT $70

NEW◆HP-150 & HP-110 VERSIONS AVAILABLE

NEXT GENERATION SYSTEMS
P.O.BOX 2987
SANTA CLARA, CA. 95055
(408) 241-5909

Forth-83

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 program 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 `GET1DIGIT#` and `GET2DIGIT#` 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, `GET1DIGIT#`, obtains just one digit of a number. But the intended progression from the `GET1DIGIT#` routine to the anticipated `GETnDIGIT#` routine never materialized. The routines did not dovetail as well as I had hoped. I had planned at least to use `GET1DIGIT#` to derive `GET2DIGIT#`. 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 outnumber 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. Get string, display it as entered.
2. Convert the input string to a numeric value.

The processing steps for converting numbers to character strings is handled by many difference routines, and includes the following steps:

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).

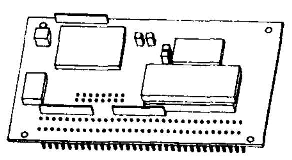
Figure Three shows the necessary source code for two versions of a number input routine (with shared primitives). The first version of DINPUT allows backspace editing of the default value, while the sec-

DASH, FIND
ASSOCIATES
Forth Recruiters

70 Elmwood Ave./Rochester, NY 14611/(716) 235-0168

TDS 9090 FORTH COMPUTER
Ideal for starter, teaching or target system

* **build into your product for rapid completion!**
* **program with IBM-PC**



- complete **Fig-Forth** system
- **30K RAM; 16K EPROM**
- over 3000 in use in Europe
- connect to keyboard, lcd display, RS 232
- **35 I/O lines; 10 bit A/D** option
- low power - down to **3 ma @ 6-16v**

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 lcd 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/novram 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. Put product application software into 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

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 numbers. Although it is common practice to accept numbers through a general input routine for strings, such an approach was inadequate for my needs.

The input routines I finally developed are better factored because they eliminate string processing in favor of simpler character processing (no string address and count is involved). In support of this, character processing is the only real requirement for number input. Obtaining a string can be viewed as the more circumspect route to number input, since it must be converted to a number later. Character input is necessary for either string or number input. So by using character input yet 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

```
SCR #13
: DIGIT-KEY
  BEGIN <KEY>
   CLR-EOL
   13 8   ANYOF2 0= >R
   DUP 48 < >R
   DUP 57 > R> OR
   R>
  AND ( IS OUTSIDE OF BOTH RANGES)
  WHILE DROP REPEAT ;


SCR #14
: <GET-DIGIT> ( N -- N? CRFLAG )
   DIGIT-KEY 13 OVER =
   IF 0> EXIT THEN
   SWAP DROP 48 - 0 ;

: GET1DIGIT#   ( DEFAULT -- N )
   <GET-DIGIT> DROP ;

EXIT
: GET1DIGIT#   ( DEFAULT -- N )
   DIGIT-KEY 13 OVER = IF DROP EXIT THEN
   SWAP DROP 48 - ;


SCR #15
: GET2DIGIT#   ( DEFAULT -- N )
   [ ' DIGIT-KEY CFA ] LITERAL 'KEY !
   PAD 2 0 FILL
   PAD 2 EXPECT
   [ ' <KEY> CFA ] LITERAL 'KEY !
   PAD C@ 0= IF EXIT THEN
   DROP PAD C@ 48 - ( 9 MIN 0 MAX)
   PAD 1+ C@ IF   ( 10'S -- )
     10 *
     PAD 1+ C@ 48 -
     ( 9 MIN 0 MAX )
     + THEN      ( N -- ) ;
```

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.

```
----
 31:
----
VARIABLE SIGNED. 0 SIGNED. !
VARIABLE MAX-DIGITS   5 MAX-DIGITS !
VARIABLE #DIGITS
VARIABLE PICTURE-STRING
24 ALLOT
" (99) 00.0"
DUP C@ 1+ PICTURE-STRING SWAP CMOVE
32 CONSTANT C@"PLUS"
: ANYOF2  ( datum test1 test2 -- datum flag )
  >R OVER =    ( datum flag -- )
  OVER R> =    ( datum flag flag -- )
  OR ;
: BACKSPACES
   ?DUP 0= IF EXIT THEN
   0 DO 8 EMIT LOOP ;
----
 32:
----
: CHAR>D+     (  d keycode -- d+ )
   C@" -" OVER = IF DROP DNEGATE EXIT THEN
   48 - DUP 0< IF  ( invalid key )
   7 EMIT DROP EXIT THEN
   DUP 16 > IF  ( letter=digit )
     7 - THEN
   DUP BASE @  ( d ones ones base -- )
   < 0=   #DIGITS @ MAX-DIGITS @ =   OR IF
     7 EMIT DROP EXIT THEN
   1 #DIGITS +!
   >R  BASE @ 1 M*/  R> 0 D+ ;
----
 33:
----
: ?D.R  ( d width -- )
   DUP BACKSPACES
   >R DDUP D0=   (  d flag -- )
   IF DDROP R> SPACES EXIT THEN
   R> D.R ;
EXIT
   PICTURE-STRING  R> OVER C@ - SPACES   D"." ;
   <# #S #> DUP >R TYPE R> R> SWAP - SPACES ;
```

(Continued on next page.)

Figure Three. Two versions of a number input routine.

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, and paths directed downward towards more minute 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

FORTH SOURCE™

WISC CPU/16

The stack-oriented "Writeable Instruction Set Computer" (WISC) is a new way of harmonizing the hardware and the application program with the opcode's semantic content. Vastly improved throughput is the result.

Assembled and tested WISC for IBM PC/AT/XT	$1500
Wirewrap Kit WISC for IBM PC/AT/XT	$ 500
WISC CPU/16 manual	$ 50

MVP-FORTH

Stable - Transportable - Public Domain - Tools
You need two primary features in a software development package... a stable operating system and the ability to move programs easily and quickly to a variety of computers. MVP-FORTH gives you both these features and many extras.

MVP Books - A Series

☐ **Vol. 1**, *All about FORTH*. Glossary	$28
☐ **Vol. 2**, *MVP-FORTH Source Code.*	$25
☐ **Vol. 3**, *Floating Point and Math*	$35
☐ **Vol. 4**, *Expert System*	$22
☐ **Vol. 5**, *File Management System*	$30
☐ **Vol. 6**, *Expert Tutorial*	$22
☐ **Vol. 7**, *FORTH GUIDE*	$25
☐ **Vol. 8**, *MVP-FORTH PADS*	$55
☐ **Vol. 9**, *Work/Kalc Manual*	$25

MVP-FORTH Software - A transportable FORTH

☐ **MVP-FORTH Programmer's Kit** including disk, documentation. Volumes 1, 2 & 7 of MVP Series, FORTH Applications, and Starting FORTH, IBM, Apple, Amiga, CP/M, MS-DOS, PDP-11 and others. Specify. $225

☐ **MVP-FORTH Enhancement Package** for IBM Programmer's Kit. Includes full screen editor & MS-DOS file interface. $110

☐ **MVP-FORTH Floating Point and Math** ☐ IBM, ☐ Apple, or ☐ CP/M, 8" $100

☐ **MVP-LIBFORTH** for IBM. Four disks of enhancements. $25

☐ **MVP-FORTH Screen editor** for IBM. $15

☐ **MVP-FORTH Graphics Extension** for ☐ IBM or ☐ Apple $100

☐ **MVP-FORTH PADS (Professional Application Development System)** An integrated system for customizing your FORTH programs and applications. PADS is a true professional development system. Specify Computer: ☐ IBM ☐ Apple $500

☐ **MVP-FORTH Floating Point Math** $100

☐ **MVP-FORTH Graphics Extension** $100

☐ **MVP-FORTH EXPERT-2 System** for learning and developing knowledge based programs. Specify ☐ Apple, ☐ IBM, or ☐ CP/M 8". $175

Order Numbers:
800-321-4103
(In California) **415-961-4103**

FREE CATALOG

MOUNTAIN VIEW PRESS
PO DRAWER X
Mountain View, CA 94040

(Figure Three, continued.)

```
----
34:
----
: DINPUT  ( d width -- d )
   0  #DIGITS !
   DUP >R SPACES BEGIN  ( d -- )
    DDUP R@ ?D.R KEY
    13 OVER - WHILE
      127 8 ANYOF2 IF  ( d char -- )
       DROP DDUP D0= IF 7 EMIT ELSE
         1 BASE @ M*/
        -1 #DIGITS +! THEN
       ELSE  ( not del or bkspc )
         CHAR>D+ THEN
   REPEAT R> 2DROP ;

----
35:
----
: DINPUT  ( d width -- d )
   0  #DIGITS !
   DUP >R SPACES  ( d -- )
   DDUP R@ ?D.R
   KEY   ( d keycode -- )
   13 OVER = IF R> 2DROP EXIT THEN
   >R DDROP  0.0 R>    ( d keycode -- )
  BEGIN    ( 0.0  keycode -- )
   127 8 ANYOF2 IF  ( d bkspc/del -- )
     DROP DDUP D0= IF 7 EMIT ELSE
       1 BASE @ M*/  -1 #DIGITS +! THEN
   ELSE  ( not del or bkspc )
       CHAR>D+  THEN
   DDUP R@ ?D.R
   KEY 13 OVER =  ( d keycode flag -- )
  UNTIL  R> 2DROP ;
----
36:
----
: D"." ( double cadr -- )
   0 MAX-DIGITS !
   >R  SWAP OVER DABS  <#
   R>
   DUP C@  ( d adr count -- )
   OVER + DO    ( d -- )
    I C@ C@" 0" C@" 9" ANYOF2 IF
      1 MAX-DIGITS +!  THEN   >R
    R@ C@" -" =  SIGNED. @ AND IF  ( INCLUDE POS/NEG SIGN)
      2 PICK 0< 0= IF R> DROP C@"PLUS" >R THEN THEN
    DDUP + R@ C@" 9" = AND IF R> DROP C@" 0" >R THEN
    DDUP D0= IF R@ C@" 9" = R@ C@" ," = OR IF
     R> DROP 32 >R THEN THEN
    R@ C@" 0" = IF # R> DROP ELSE R> HOLD THEN
   -1 +LOOP
   ROT  DROP  #> TYPE ;
```

Table One.

<u>Data</u>	<u>Input Resources</u>	<u>Output Resources</u>
numbers	`EXPECT/NUMBER`	`D.R <# # HOLD #S #> TYPE`
dates	`EXPECT`	`<# # HOLD #> TYPE`
strings	`EXPECT`	`COUNT TYPE`

Table Two.

<u>&lt;# # # ... #&gt;</u>	<u>&lt;# B# B# ... #B&gt;</u>
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.	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#`.

Formatted Numeric Input Via Picture Strings

The routines that have been offered for number input can easily be extended to support an input template or picture string, so that only a given number of digits can be input, with automatically included hyphens, decimal points, or the like.

The extra input functionality is achieved mostly as a result of calling a different output function. With minor changes in `?D.R`, the overall behavior of `DINPUT` changes dramatically, even though its definition remains unchanged (see Figure Three).

To implement a new user interface, rearrange the code in screen 32 of Figure Three so that the final line of `?D.R` is the desired one from the three available endings.

By using `D.R` in `?D.R`, `DINPUT` will compile as a number input routine for calculator-style input (old digits scroll left when new characters are entered at the rightmost end of the number).

By using `D.` in `?D.R`, `DINPUT` will compile as a left-to-right-digit style of number entry routine.

By using `D"."` in `?D.R`, `DINPUT` will compile as a calculator-style input routine which includes picture string support. The picture string defines an input template including commas, currency signs, or hyphens and any number of digits you deem appropriate (up to the limit of a double integer).

The picture string is read from the `PICTURE-STRING` variable. It must have a count byte in the first memory location, followed by a series of characters which are interpreted as follows:

0	Output a digit at this location.
9	Output a digit at this location, but only if it is significant (suppress leading zeros).
,	Output a comma at this location, but only if preceded by a significant digit.
-	Output a hyphen at this location if the value of `SIGND.` is nonzero; otherwise, output a sign indication. In the latter case, display a minus if the quantity is less than zero; otherwise, display the ASCII character corresponding to the constant `C@"PLUS"` when the quantity is non-negative.
<other>	Output the specified other character at this location.

To accept a telephone number, the picture string might look like:

```
(999) 999-9999
```

To accept a currency amount, you may specify a sign indicator in the picture string (remember to set `SIGND.` to a nonzero, or truth, value):

```
$ 99990.00 -
```

`D"."` can be considered a single replacement for all of the separately collected definitions for currency output, date output, telephone number output, and so on. You can also switch sign indications or widths of fields quite easily for different contexts and display needs, without compiling definitions tailored for specific kinds of input. Such a routine can be called a data-driven routine, since its actions are guided by an external data structure. The external data structure is much easier to change (without recompiling) compared to writing individual routines each time the need arises.

`D"."` also offers a more natural illustration of the desired output format, primarily because Forth's own toolset requires you to work in reverse. For example, a sign-preceded, two-digit integer could be displayed with the following sequence:

```
<# # # SIGN #> TYPE
```

Notice that the sign indication is specified after the calls to # because # yields one digit of the number, starting with the least-significant digit. The same effect can be achieved by entering:

```
" -00" PICTURE-STRING $! D"."
```

Note that the sign is placed exactly where one would expect when visualizing the number in the normal left-to-right sequence. `$!` is assumed to be a string storage operator that stores a string to the location that is atop the stack, from the source location given by the second address on the stack.

The usefulness of `D"."` as an extension of Forth's number output toolset has merit on its own. But this usefulness is dramatically increased when it is used in tandem with `DINPUT` so that number input can be just as "pretty" as number output.

Mike Elola 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.

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

1. Ham, Michael. "Structured Programming" column, *Software Tools*, July 1986.
2. Ham, Michael. "Structured Programming" column, *Software Tools*, April 1987.
3. Ham, Michael. "Making Numbers Pretty," *Forth Dimensions* VII/5, 1986.
4. Takara, Ken. "Number Editing Utility," *Forth Dimensions* VII/3, 1986.

FIG CHAPTERS

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. Grevillea Ave.
  Phillip Wasson
  (213) 649-1428

  North Bay Chapter
  2nd Sat., 10 a.m. Forth, AI
  12 Noon Tutorial, 1 p.m. Forth
  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

  Fairfield FIG Chapter
  4th Day, 8:15 p.m.
  Gurdy Leete (515) 472-7077

- **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 Weis Drive
  Ellisville, MO 63011

- **NEW JERSEY**
  New Jersey Chapter
  Rutgers Univ., Piscataway
  Nicholas Lordi
  (201) 338-9363

- **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.
  5115 Fisher Road
  Terry Webb
  (614) 878-7241

  Dayton Chapter
  2nd Tues. & 4th Wed., 6:30 p.m.
  CFC. 11 W. Monument Ave. #612
  Gary Ganger (513) 849-1483

- **OREGON**
  Willamette Valley Chapter
  4th Tues., 7 p.m.
  Linn-Benton Comm. College
  Pann McCuaig (503) 752-5113

- **PENNSYLVANIA**
  Villanova Univ. Chapter
  1st Mon., 7:30 p.m.
  Villanova University
  Dennis Clark
  (215) 860-0700

- **TENNESSEE**
  East Tennessee Chapter
  Oak Ridge
  3rd Wed., 7 p.m.
  Sci. Appl. Int'l. Corp., 8th Fl.
  800 Oak Ridge Turnpike
  Richard Secrist
  (615) 483-7242

- **TEXAS**
  Austin Chapter
  Matt Lawrence
  PO Box 180409
  Austin, TX 78718

  Dallas Chapter
  4th Thurs., 7:30 p.m.
  Texas Instruments
  13500 N. Central Expwy.
  Semiconductor Cafeteria
  Conference Room A
  Clif Penn (214) 995-2361

  Houston Chapter
  3rd Mon., 7:30 p.m.
  Houston Area League of PC Users
  1200 Post Oak Rd.
  (Galleria area)
  Russell Harris
  (713) 461-1618

- **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

  Potomac FIG
  D.C. & Northern Virginia
  1st Tues.
  Lee Recreation Center
  5722 Lee Hwy., Arlington
  Joseph Brown
  (703) 471-4409
  E. Coast Forth Board
  (703) 442-8695

  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 Tregeagle
  10 Binda Rd.
  Yowie Bay 2228
  02/524-7490
  Usenet
  tedr@usage.csd.unsw.oz

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

  Southern Belgium Chapter
  Jean-Marc Bertinchamps
  Rue N. Monnom, 2
  B-6290 Nalinnes
  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., 10a.m.-noon
  N. Alta. Inst. of Tech.
  Tony Van Muyden
  (403) 486-6666 (days)
  (403) 962-2203 (eves.)

  Southern Ontario Chapter
  Quarterly, 1st Sat., Mar., Jun., Sep., Dec., 2 p.m.
  Genl. Sci. Bldg., RM 212
  McMaster University
  Dr. N. Solntseff
  (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, Surry 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
  46/8-929631

- **SWITZERLAND**
  Swiss Chapter
  Max Hugelshofer
  Industrieberatung
  Ziberstrasse 6
  8152 Opfikon
  01 810 9289

SPECIAL GROUPS

- **NC4000 Users Group**
  John Carpenter
  1698 Villa St.
  Mountain View, CA 94041
  (415) 960-1256 (eves.)

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

Second Class
Postage Paid at
San Jose, CA