FORTH Dimensions

Volume 7, Number 3

September/October 1985

$2.50

Synonyms and Macros

Pseudo-Interrupts

Universal Text File Reader

Code Inspections

The ForthCard™

STAND ALONE OPERATION

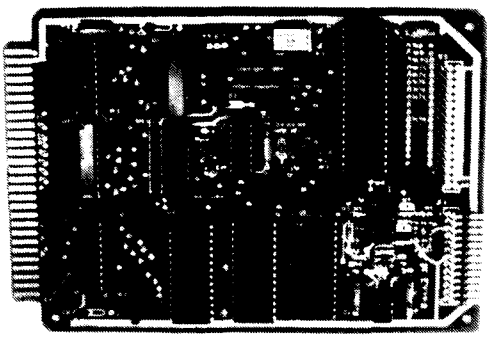
STD BUS INTERFACE

EPROM/EEPROM PROGRAMMER

RS-232 I/O

PARALLEL I/O

ROCKWELL FORTH CHIP



Evaluation Unit **$299**
Part #STD65F11-05 includes: ForthCard, Development ROM, 8Kbyte RAM, Manuals

OEM Version as low as **$199**
Part #STD65F11-00 does not include memory or manuals

The Forthcard provides OEMs and end users with the ability to develop Forth and assembly language programs on a single **STD bus compatible** card.

Just add a CRT terminal (or a computer with RS-232 port), connect 5 volts and you have a **self contained Forth computer.** The STD bus interface makes it easy to expand.

Download Forth source code using the serial port on your PC. Use the **onboard EPROM/EEPROM programming** capability to save debugged Forth and assembly language programs. Standard UV erasable EPROMs may also be programmed with an external Vpp supply.

NEW! Options and Application Notes

Electrically Eraseable PROMs (EEPROMs)

FREEZE the dictionary in EEPROM (save in non-volatile memory, to be restored on power up)

Download Software for your IBM PC or CP/M

Non-Volatile CMOS RAM with battery 2K, 8K, optional Clock/calendar

Fast 2MHz clock (4MHz crystal)

Disk Controller Card (5¼")

Self Test Diagnostics

Parallel printer interface

Ask about our ForthBox™
A complete STD bus oriented system including the ForthCard, Disk Controller, Disk Drive(s), STD Card Cage, Cabinet and power supply.

CALL TODAY FOR COMPLETE INFORMATION!

HiTech Equipment Corporation

9560 Black Mountain Road
San Diego, CA 92126
(619) 566-1892

MasterCard VISA

FORTH Dimensions

Published by the
Forth Interest Group

Volume VII, Number 3
September/October 1985

Editor
Marlin Ouverson

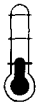
Production
Cynthia Lawson Berglund

Forth Dimensions solicits editorial material, comments and letters. No responsibility is assumed for accuracy of material published. Unless noted otherwise, material published by the Forth Interest Group (a non-profit organization) is in the public domain. Such material may be reproduced with credit given to the author and to the Forth Interest Group.

Subscription to *Forth Dimensions* is free with membership in the Forth Interest Group at $20 per year ($33 foreign air). For membership, change of address and to submit material for publication, write to: Forth Interest Group, P.O. Box 8231, San Jose, California 95155.

ISSN No. 0884-0822

Symbol Table



Simple; introductory tutorials and simple applications of Forth.

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

Advanced; requiring study and a thorough understanding of Forth.

Code and examples conform to Forth-83 standard.

Code and examples conform to Forth-79 standard.

Code and examples conform to fig-FORTH.

Deals with new proposals and modifications to standard Forth systems.

FORTH Dimensions

FEATURES

11 Synonyms and Macros, Part I **by Victor H. Yngve**

A ROT by any other name — is it the same? Investigation into this seemingly innocuous subject leads to productive new insights into how Forth works, and how to make it work for you.

14 Synonyms and Macros, Part II **by Victor H. Yngve**

This tool creates high-level macro definitions with the readability of colon definitions and some increase in execution speed. Installing macro tools is a great way to learn about the insides of your Forth implementation!

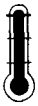
19 Forth Timer Macros **by Iram Weinstein**

We asked for readers' uses of Forth macros — what better than tools to improve the performance of the rest of your code? When the engine is installed and you're ready to fine tune, look for these in your desk-side toolbox.

28 Improved Forth-83 DO LOOP **by Dennis L. Feucht**

The parsimony of the two-item **DO LOOP** is possible in Forth-83 at the expense of slightly more complex compile-time procedures. This implementation requires minimal modification to the machine-dependent, run-time words of previous Forth-83 **DO LOOP**s. Metacompiling words are also provided.

30 A Forth I/O Technique: Pseudo-Interrupts **by Ed Schmauch**



Using interrupts allows a CPU to do background processing concurrently with I/O, but requires greater hardware know-how. Here is an intermediate technique which uses simple polling and allows background processing like interrupts.

34 An Approach to Reading Programs
by Kim Harris & Michael Ham

What increases productivity, reduces development time and contributes to programmer training? Formal "code inspections" facilitate the difficult task of truly comprehending a program, and help programmers learn from their colleagues' techniques and skills.

35 Volume VI Index **by Julie Anton**

Trying to remember where in *Forth Dimensions* that article or piece of code was printed, or if a particular subject was covered? This subject/author/title index was prepared as a reader service.

37 Number Editing Utility **by Ken Takara**

Crashproofing an application often means shielding the program from bad numeric input and providing in-progress editing features to the user. Vectored execution makes this technique powerful and general enough for many uses.

DEPARTMENTS

5 Letters
6 Editorial: "Macronautics"
7 Application Tutorial: "Universal Text File Reader"
by John James
41 Advertisers Index
41 Chapter News by Michael Ghormley
42 FIG Chapters

THE Journal OF Forth Application AND Research

The aim of the *Journal* is to provide a reliable source of state of the art techniques and applications of Forth to scientific and industrial problems. The *Journal's* editorial review board is drawn from the foremost workers in Forth in the U.S., Europe and Canada. Past issues had in-depth coverage of such topics as Robotics, Data Structures, Forth Computers, Real-Time Systems, and Extended Address Computing. The *Journal* is open to all new work in Forth as well as the entire range of threaded interpretive languages and Forth-like systems.

Volume 3

The *Journal* is beginning its third year of publication. Published quarterly, the *Journal* contains applications and techniques papers, technical notes, review papers, algorithms, book reviews, and conference abstracts.

Renew your subscription for Volume 3 of the *Journal of Forth Application and Research*. Here's what you'll find in Issue #1:

- —Forth-Based Software for Real-Time Control of a Mechanically-Scanned Ultrasonic Imaging System
- —Fast and Flexible Forth Programming in a Femtosecond Laser Lab
- —Stack Frames and Local Variables
- —Should VARIABLE Be an Immediate State-Sensitive Word?
- —Readable and Efficient Paramet Access via Argument Records
- —Run-Time Error Handling in Forth

Proceedings Rochester Forth Conference

1985 Rochester Forth Conference Proceedings
The Proceedings of the 1985 Rochester Forth Conference will appear as a Special Issue of Volume 3 of the *Journal*. The conference had a theme of software management and engineering and how to improve software productivity. Invited papers discuss a space shuttle experiment using Forth, the automation of an airport with Forth, and Forth in the development of MAGIC/L, and HFORTH: A business applications language. Proceedings available as a single issue, $20.

Subscription Rates, Vol. 3: Corporate/Institutional $100, Individual $40. Outside N. America add $20 for airmail charges. Prepaid subscriptions only, in US funds on a US bank, or by international money order, payable to Journal of Forth Application and Research, Inc.. Back issues available at $15 each, add $5 for foreign airmail. Send all orders to: Journal of Forth Application and Research, P.O. Box 27686, Rochester, NY 14627 USA.

Order Form

- ☐ Subscriptions Volume 3, 1985 Issues 1–4
  ☐ Institutions/Corporate $100 ☐ Individuals $40
- ☐ Subscriptions Volume 4, 1986 Issues 1–4
  ☐ Institutions/Corporate $100 ☐ Individuals $40
- ☐ Subscriptions outside N. America please add $20 *per volume* for airmail.
- ☐ Vol. 1 #1 **Robotics** $15
- ☐ Vol. 1 #2 **Data Structures** $15
- ☐ Vol. 2 #1 **Forth Machines** $15
- ☐ Vol. 2 #2 **Real-Time Systems** $15
- ☐ Vol. 2 #3 **Enhancing Forth** $15
- ☐ Vol. 2 #4 **Extended Addressing** $15
- ☐ Back issues outside N. America please add $5 per issue for airmail.
- ☐ Four or more back issues $12 each issue.
- ☐ Four or more issues outside N. America $16 each (includes airmail).
- ☐ Rochester Forth Conference Proceedings $20
- ☐ Please send further information on the Institute's publications and activities.

Name _____

Address _____

_____

_____

Amount Enclosed $_____

VISA/MC _____ Exp. _____

Signature _____

Letters to the Editor

Addressing the Marketplace

Dear Marlin,

"The biggest (and most common) mistake that can be made in a computer design is that of not providing enough address bits..."[1]

This deserves to go down as one of the classic statements of the century. Although originally referring to computer hardware, I believe that it applies just as well to the Forth virtual machine. Elsewhere in the same volume, I found the startling information that on the average, for a given price range, machines increase their (virtual) memory address by one bit per year, and their physical memory address by one bit per two years.

It is already the case that machines with a 64K memory are becoming restricted to the very low end of the marketplace. Increasing numbers of people are buying machines with much larger memories, and if they can't make effective use of this memory with Forth, they will use something else.

Of course, various thirty-two-bit Forths are available commercially, but no provision was made in the Forth-83 Standard for addresses to be other than sixteen bits. Although the next revision of the standard will undoubtedly do something about this, it could well be too late, and we will have a proliferation of incompatible solutions to the problem.

So I think it would be good to have some discussion about this, here in *Forth Dimensions* (which reaches the widest Forth audience), and maybe get some concensus. I am going to open the discussion with a proposal. I don't mind getting shot down in flames, so long as we are at least considering the question.

1. Existing (working) standard programs should still produce correct results, as far as possible.

2. A cell on the parameter or return stack will be large enough to hold an address. This will be at least sixteen bits, but the actual size will be implementation dependent. It will be available in a constant named **LSIZE** (with the L standing for "long").

3. @ and ! will still transmit sixteen bits. This way, sequences such as

```
DO ... I @ ... 2 +LOOP
```

will still work.

4. If more than sixteen bits are available in a stack cell, @ will sign-extend. A new operator **U@** will fetch sixteen bits but will supply high-order zeroes. This is consistent with the existing set of unsigned operators. On sixteen-bit Forths, **U@** will be an alias for @.

5. New operators **L@** and **L!** will transmit a full stack cell (i.e., **LSIZE** bits).

6. Existing double operators (e.g., **2SWAP** and **D+**) will still operate on two stack cells.

7. A compilation address will stay at sixteen bits.

This last point (at least!) is debatable. It would preclude allocating very large arrays within the dictionary, so that the dictionary can remain within 64K. An alternative would be to increase a compilation address to **LSIZE**. This, however, would expand the dictionary considerably in those implementations where **LSIZE** is greater than sixteen. A further possibility is to make the decision implementation dependent, in which case we would need yet another fetch-and-store operation (for compilation addresses) and another constant giving the length of a compilation address. I think my suggestion is the simplest.

Another headache concerns arithmetic operations on certain well-known processors, where the designers have inflicted us with a basically sixteen-bit design, but kludged to provide a longer address (the term "segment" sounds nice but doesn't change the facts). The most "efficient" solution would be for +, *, etc. to remain sixteen-bit operations even where **LSIZE** is longer. Thus, high-order bits of the "result" would be invalid. I think we should reject this approach, even at the expense of an execution-time penalty — the elegance of Forth would be compromised, and address arithmetic would be a nightmare! It's a pity, but that's probably the best we can do with awkward, underlying hardware.

Happy Forthing, even to those with 8086/8088s.

Michael Hore
Numbulwar, NT
Australia

Reference:

1. C.G. Bell and J.C. Mudge, "The Evolution of the PDP-11," in D.P. Siewiorek, C.G. Bell and A. Newell (eds.), *Computer Structures: Principles and Examples* (McGraw-Hill, 1982), p. 776.

More Applications, Please!

Dear Mr. Ouverson,

As a newcomer to Forth, I find it interesting that most of the discussion in *Forth Dimensions* is technical (e.g., the best **CASE** or **LEAVE** construct). Where are the articles on application programs in Forth? Possibly it is because hackers (myself included) enjoy working on small system utilities more than writing application programs.

Thankfully, you have published several good application programs. Three articles were especially interesting:

Editorial

Macronautics

We had a hunch our readers would benefit from macros back when we first published Jeffrey Soreff's article (*Forth Dimensions* V/5). *Mirabile dictu!* Macros reappear in this issue in a number of places: one letter from a reader shares two macros with us to aid program legibility; an article uses the technique to obtain benchmarks of Forth words; and from the University of Chicago, Professor Victor Yngve delivers a different method for creating macros and proposes an innovative use of them — but be sure to read his "Synonyms" first. Afterwards, dig out Soreff's original article and the follow-up by Don Taylor (VII/1), compare them, use the techniques, draw some conclusions and send your written results to us. And who will be the first to share his tookit of macros with the FIG community?

With this issue, a new coordinator will be reporting on the international network of FIG chapters. John Hall has stepped down from the position in order to spend time planning this year's FIG convention and to spend more time with his duties as a member of the FIG Board of Directors. Michael Ghormley will be charging ahead with the chapter work now, and is already working with the individual chapter coordinators. Mike brings abundant energy and creativity to his post, and we look forward to working with him.

FORML's annual conference at Asilomar, on California's Monterey peninsula, is fast approaching. For those of you haven't yet attended one, this is an intimate conference for advanced Forth practitioners. Its size permits close interaction between attendees, many of whom are very experienced Forth programmers with techniques to share and proposals for open consideration. Presentation of a paper is no longer required of all attendees, though most still participate fully. (In addition to a complete schedule of meetings, full participation includes walks through windswept trees overlooking the Pacific Ocean; conference grounds filled with deer, racoons and sand dunes; and tasty meals. A few non-programmer family members or spouses are usually present just to enjoy the locale.) For details about the November 29 - December 1 event, call the FIG Hotline at 408-277-0668.

We are experimentally providing an index to advertisers in this issue. The table of contents lists its location — we hope this makes it easier for you to find the particular vendor or product that was advertised in here *somewhere*... Finally, this issue is going to press just before the seventh annual Forth National Convention, so coverage of that event will be delayed until next issue. I look forward to renewing old friendships there but if we don't get the chance, then come to the FORML conference in November!

—Marlin Ouverson
Editor

(Letters continued from page 5)

"Simple Modem I/O," "Quicksort and Swords" and "A Forth Spreadsheet." I hope you can get John S. James to continue to write on the modem I/O words. Perhaps he could develop code words for an interrupt-driven input buffer, and then go on to develop a complete Forth communications program with file transfer, modem control, everything! This is a large project to put in the public domain. I hope someone is willing to write the necessary series of articles.

Thank Craig A. Lindley for his Forth spreadsheet, and please keep publishing the application programs. Thanks for your effort in publishing *Forth Dimensions*.

Sincerely,

Ramer W. Streed
North Mankato, Minnesota

It Would Have Been Grand...

Dear FIG,

I always enjoy Henry Laxen's column. After reading the YACS ["Yet Another Case Statement," VI/6, VII/1] tutorials a few times, I tried to implement the code on my Kaypro using LMI's Z80 FORTH, version 3. Though this version complies with the 83 Standard,] does not work as described in Mr. Laxen's column. I replaced line four with the following code segment:

```
BEGIN  >MARK  !CSP  1
STATE  !  INTERPRET
?DUP  WHILE  >RESOLVE  REPEAT
```

and it now works.

I was also glad to see macros addressed again (*Forth Dimensions* VII/1). The article by Jeffrey Soreff (V/5) is one of my favorite Forth articles. I have found the following macros to be helpful in making my code more readable by eliminating **0= IF** and **0= WHILE.**

```
: IF-NOT  COMPILE 0=
[COMPILE] IF  ; IMMEDIATE

: WHILE-NOT  COMPILE 0=
[COMPILE] WHILE  ; IMMEDIATE
```

I would like to thank the staff of the Forth Interest Group for all its great work and services.

Sincerely,

Ed Petsche
Greenport, New York

Editor's note: For more on the "Case of the Right Bracket," see the letters section of our last issue.

Application Tutorial

Universal Text File Reader

John S. James
Santa Cruz, California

File incompatibilities still cause serious problems. For example, files uploaded from different operating systems may have different ways of marking the end of a line, and the software available may not be able to list them with legible results. Even files produced on the same operating system may contain non-standard codes, internal to certain word processors, for such functions as tabs and underline. And you may occasionally want to read files not intended as text files; to see, for example, what operator and error messages an object program contains.

This tutorial shows you how to write a simple program to solve most of these problems — one to read almost any text file and do the best it can to give a reasonable result. And it meets the design goal of not asking the operator anything about what kind of file is being listed. (The operator may not know!)

We use F83 here to show how to get started with sequential file I/O and how to turn your programs into DOS commands which can be useful outside the Forth environment.

What the Program Does

Here are several common file-compatibility problems and ways the program handles them:

- If the parity bit is set on any character, the program clears and ignores it.
- The program cannot know what character, if any, indicates end of file. So it ignores them all and lists everything, until the file read operation detects end of file. Usually, the final block is filled out with end-of-file characters, or nulls, blanks, or something else fairly harmless.
- Different end-of-line indicators must be handled. Common ones are a carriage return followed by a linefeed, a carriage return alone, and a null alone. Word-processor "document" files sometimes have no end-of-line indication, except at the end of paragraph. We will take carriage return, null and linefeed as meaning end of line, and we will then supply a carriage return and linefeed. However, more than two carriage returns in a row will not be supplied — sacrificing triple spacing — to avoid problems such as multiple nulls in certain files. And if a carriage return is followed in the original file by a linefeed, that linefeed will be ignored, so that single-spaced text files will not come out as double spaced.

  We won't reformat word-processor document files without hard returns. Almost all terminals will scroll automatically when necessary. Words may be split at end of line, but they will still be readable. See note below about a better way to format text to various line lengths.
- This program ignores backspace, to avoid loss of information caused by unintended backspace in non-text data.
- We change formfeed to linefeed, to avoid waste of paper when printing object programs or other non-text files.
- We ignore other non-printable characters. However, you might want to print them as dots, or as some other character, to show where they were.

The Code

This particular implementation is for the well-known F83 system. It should run on any PC-compatible machine, and should not be difficult to convert to any Forth system on any computer, providing you know how to open a sequential file, read records and detect end of file. We chose not to go through Forth I/O — the word **BLOCK** — because many Forth systems don't provide any way to read an incomplete final block of a file not created for a Forth environment.

The first screen isolates most of the code which is specific to F83. If you aren't using F83, you will need to write whatever words are necessary for sequential I/O on your system. Or you could do it the easy way and use **BLOCK** if your system can read partial blocks which may be at the end of a file.

Notice that we did avoid the work of opening a file from scratch, by using an **OPEN** which this Forth system already provides for access to files of screens. Since that **OPEN** was intended only for random I/O, we had to initialize one more field in the file control block; that's the purpose of the word **FILE-RESET**.

Incidentally, the word **DOS-EMIT** in the first screen is almost identical to the Forth word **EMIT** supplied by the system. But **DOS-EMIT** uses the DOS call instead of the BIOS call to write a character to the screen. The practical difference is that with **DOS-EMIT** you can have the output printed by turning on the printer through DOS, by using whatever key or combination of keys your system provides for this purpose.

The word **?OUT** in the second screen filters the stream of characters and changes any which should be ignored to -1, which is not a possible ASCII value. Then it prints all except those it changed to -1. The first line of **?OUT** allows any key to toggle a pause, so users can stop the display and have time to read the information before it scrolls off the screen. Users can end the listing with control-C.

The variable **LINES** keeps count of multiple new lines in a row. After two, no more new lines are generated. We suppress additional new lines so that the program will behave reasonably even if there are strings of hundreds of carriage returns or nulls. **LINES** treats the linefeed character even more severely, suppressing it if even one new line has just occurred. The purpose is to prevent the normal carriage return and linefeed format from always generating double spacing.

Incidentally, most programmers prefer more white space in their code than is shown here. This example has

PRIME FEATURES

- Execute DOS level commands in HS/FORTH, or execute DOS and BIOS functions directly.
- Execute other programs under HS/FORTH supervision. (editors debuggers file managers etc)
- Use our editor or your own.
- Save environment any time as .COM or .EXE file.
- Eliminate headers, reclaim space without recompiling.
- Trace and decompile.
- Deferred definition, execution vectors, case, interrupt handlers.

HS/FORTH

- Full 8087 high level support. Full range transcendentals (tan sin cos arctan logs exponentials)
- Data type conversion and I/O parse/format to 18 digits plus exponent.
- Complete Assembler for 8088, 80186, and 8087.
- String functions - (LEFT RIGHT MID LOC COMP XCHG JOIN)
- Graphics & Music
- Includes Forth-79 and Forth-83
- File and/or Screen interfaces
- Segment Management
- Full megabyte - programs or data
- Fully Optimized & Tested for: IBM-PC XT AT and JR COMPAQ and TANDY 1000 & 2000 (Runs on all true MSDOS compatibles!)
- Compare

BYTE Sieve Benchmark jan 83
HS/FORTH 47 sec BASIC 2000 sec
with AUTO-OPT 9 sec Assembler 5 sec
other Forths (mostly 64k) 55-140 sec

FASTEST FORTH SYSTEM AVAILABLE.
TWICE AS FAST AS OTHER FULL MEGABYTE FORTHS!
(TEN TIMES FASTER WHEN USING AUTO-OPT!)

HS/FORTH, complete system only: $270.

VISA Visa Mastercard

HARVARD SOFTWORKS

P.O. BOX 69
SPRINGBORO, OH 45066
(513) 748-0390


```
Scr # 3          A:FD08.BLK
  0 \ Universal text-file reader                              18Aug85 JJ
  1 \ This screen has words specific to F83.
  2 \ Note that we are relying on the F83 word OPEN .
  3 \   After you open a file with  OPEN <name>  , the FCB address
  4 \   will be in the F83 variable FILE .
  5 : DOS-EMIT   ( c -- )  2 BDOS DROP ;  \ DOS function call #2
  6 128 CONSTANT RECORD-LENGTH
  7 CREATE RECORD  RECORD-LENGTH ALLOT  \ Buffer for DOS to use
  8 : SET-RECORD   \ --   ;P Tell DOS to use RECORD for buffer
  9    RECORD  26 BDOS  DROP ;
 10 : FILE-RESET   \ --  ;P Set sequential-record number in FCB
 11    0  FILE @ 32 +  C! ;
 12 : READ-RECORD   ( -- ?   ;P Read 128 bytes; 0 = success )
 13    FILE @  20 BDOS  1 <> ;  \ '1' means no more data at all
 14 : INITIALIZE-DOS   \ --   ;P Prepare DOS for sequential I/O
 15    SET-RECORD   FILE-RESET ;

Scr # 4          A:FD08.BLK
  0 \ Transform input character, and maybe write it           18Aug85 JJ
  1 VARIABLE LINES  \ Counter to prevent multiple lines
  2 : ?KILL   \ c1 -- c2   ;P Kill unwanted control characters
  3    DUP 0 <>  OVER 10 <>  AND  OVER 12 <>  AND  OVER 13 <>  AND
  4    IF ( None of these )  DROP -1  THEN ;
  5 : ?NEW-LINE   \ Start a new line, if not more than 2 in a row
  6    LINES @ 2 <  IF  13 DOS-EMIT 10 DOS-EMIT  1 LINES +!  THEN ;
  7 : ?OUT   \ c --   ;P Maybe output the character
  8    KEY? ( F83 ) IF  KEY DROP KEY DROP  THEN \ Toggle pause
  9    127 AND   \ Clear parity bit if set
 10    DUP 32 <  IF  ?KILL  THEN \ Change unwanted char. to -1
 11    DUP 12 =  IF  DROP 10  THEN \ Change formfeed to linefeed
 12    DUP 10 = IF  DROP -1  LINES @ 0= IF  ?NEW-LINE  THEN  THEN
 13    DUP 13 =  OVER 0 = OR  IF  DROP -1  ?NEW-LINE
 14      ELSE  0 LINES !  THEN \ New line if CR, or Null
 15    DUP -1 <>  IF  DOS-EMIT  ELSE  DROP  THEN ;

Scr # 5          A:FD08.BLK
  0 \ Main loop to read the file                              18Aug85 JJ
  1 : READ-FILE   \ --  ;P List the OPEN file.
  2    INITIALIZE-DOS  0 LINES !
  3    BEGIN  READ-RECORD  WHILE  \ There is more data
  4      RECORD RECORD-LENGTH  OVER + SWAP
  5      DO  I C@ ?OUT  LOOP  REPEAT ;
  6
  7
  8 \ Optional F83 code for saving program to be used from DOS.
  9 \ To use the program, from DOS say  LIST <filename>  .
 10 : BOOTUP   ( -- )   DEFAULT ( to open <filename> for input )
 11    READ-FILE   0 0 BDOS ( Return to DOS ) ;
 12 ' BOOTUP IS BOOT
 13 : NEW-ERR   ( n n f -- )   IF  0 0 BDOS  ELSE  2DROP  THEN ;
 14 ' NEW-ERR IS ?ERROR
 15 SAVE-SYSTEM LIST.COM
```

been packed more tightly to save space for publication.

Turnkey Operation

The second part of the last screen has optional code to make this program usable as a DOS command. The F83 word **BOOT** controls action of the system on startup; here it is set to open a filename given in the DOS command line as a default input file, then to execute **READ-FILE** to run the program, then to return to DOS. The F83 word **IS** provides a convenient way to change the meaning of certain other words in F83 (such as **BOOT**) which have been set up to allow this kind of re-vectoring.

The word **?ERROR** should also be re-vectored, since otherwise an attempt to list a non-existent file would leave the user in Forth, not in DOS. The system call **0 0 BDOS** returns control to DOS. The word **NEW-ERR** has a confusing stack effect; when there is no error it must dispose of three arguments, since other words in this system expect **?ERROR** to use three. But there is no need to drop these extra arguments in the case that **0 0 BDOS** is executed (when there is an error), since then the program is finished anyway.

Notice that this program uses the old DOS calls (before hierarchical directories). Therefore, it will work with old versions of DOS, such as 1.1, as well as with all later versions. If you are using a later version of DOS, such as 2.1, note that the **LIST** command which this program creates cannot accept a pathname. It can list files in a hierarchical directory, either by having a copy of LIST.COM in the same directory as the file to be listed, or by using the DOS command PATH. And it can redirect its output into a file — which can prove useful for cleaning files before uploading if they have parity bits set or contain other non-text characters.

Note on Formatting Text

This tutorial derives from my computer-conferencing program "CommuniTree Second Edition," which needs to read files transmitted by phone from different operating systems, with or without hard returns, and displays them on terminals of different sizes. The program cannot ask what kinds of files they are; the same procedure must work in all cases.

The biggest problem, which we ignored in this tutorial example, was determining when carriage returns are "hard." Tables, poetry, legal text with numbered lines, character graphics and other such material must preserve the existing line structure; all end-of-line indicators in the file must cause a new line in the output, so carriage returns are considered to be hard. But for listing straight text, which may have been sent with one line length and now may be formatted to another, the program should recompose the line endings until the end of paragraph.

The solution chosen was to let the user set the terminal line length, not only to a usual range of values (20–132), but also to zero. Zero means to respect the existing line structure: make a new line at every end of line, and not otherwise (as the program example presented in this article does). Any non-zero line-length value will cause text to be reformatted, unless a paragraph break is indicated (by an end of line followed by either a blank or another end of line).

For poetry, tables, etc., we advise users to start every line with at least one blank, since lines beginning with a blank will not be reformatted onto previous lines. But since users won't always do so, the CommuniTree system comes up with the line length defaulted to zero. Since most users these days have the same terminal line length (eighty characters), and because those who don't can specify their line length, the only remaining problem concerns internal word-processor document files which have no end of line until the end of a paragraph. But most terminals scroll automatically if more than eighty characters are sent in a line, so the data isn't lost and the only problem is that words will be divided in the middle (i.e., wrapped incorrectly). Users can set their terminal line length to its actual value (not zero) if this lack of proper word wrap bothers them. But since most upload programs provide end-of-line characters, the problem seldom arises.

One final adjustment. On many terminals, if you format text to the full line length, there will be extra blank lines inserted in the printout after those lines which happen to go to their full length. Writing into the last position on the line causes an automatic scroll, as it should, but the terminal software isn't smart enough to suppress a single linefeed character which arrives immediately after the terminal has supplied its own new line. So we set the line length to one less than the user requests; eighty-column terminals use only seventy-nine. Users who notice or care can ask for eighty-one.

References

1. Microsoft Corporation, *Disk Operating System Technical Reference*, publ. by IBM Corporation. This manual explains DOS calls. You can also find this information in various books about MSDOS or programming the IBM and compatibles.

2. The F83 source code is an excellent source of information for the intermediate or advanced student. All of the source is supplied with this public-domain system. The command **VIEW** <**name**> will show the screen where a word is defined; and then you can use the words **A L** to get to the "shadow" documentation screen. If you are running F83 but don't have the source files handy, the command **SEE** <**name**> provides a convenient decompilation.

FORTH IS NOW VERY.FAST!

.Sieve 1.3s/pass
.Compile 300 screens/minute
.Drop 1.82 us
.Concurrent I/O @ 250K baud

DEVELOP YOUR APPLICATIONS IN A TOTAL FORTH ENVIRONMENT.

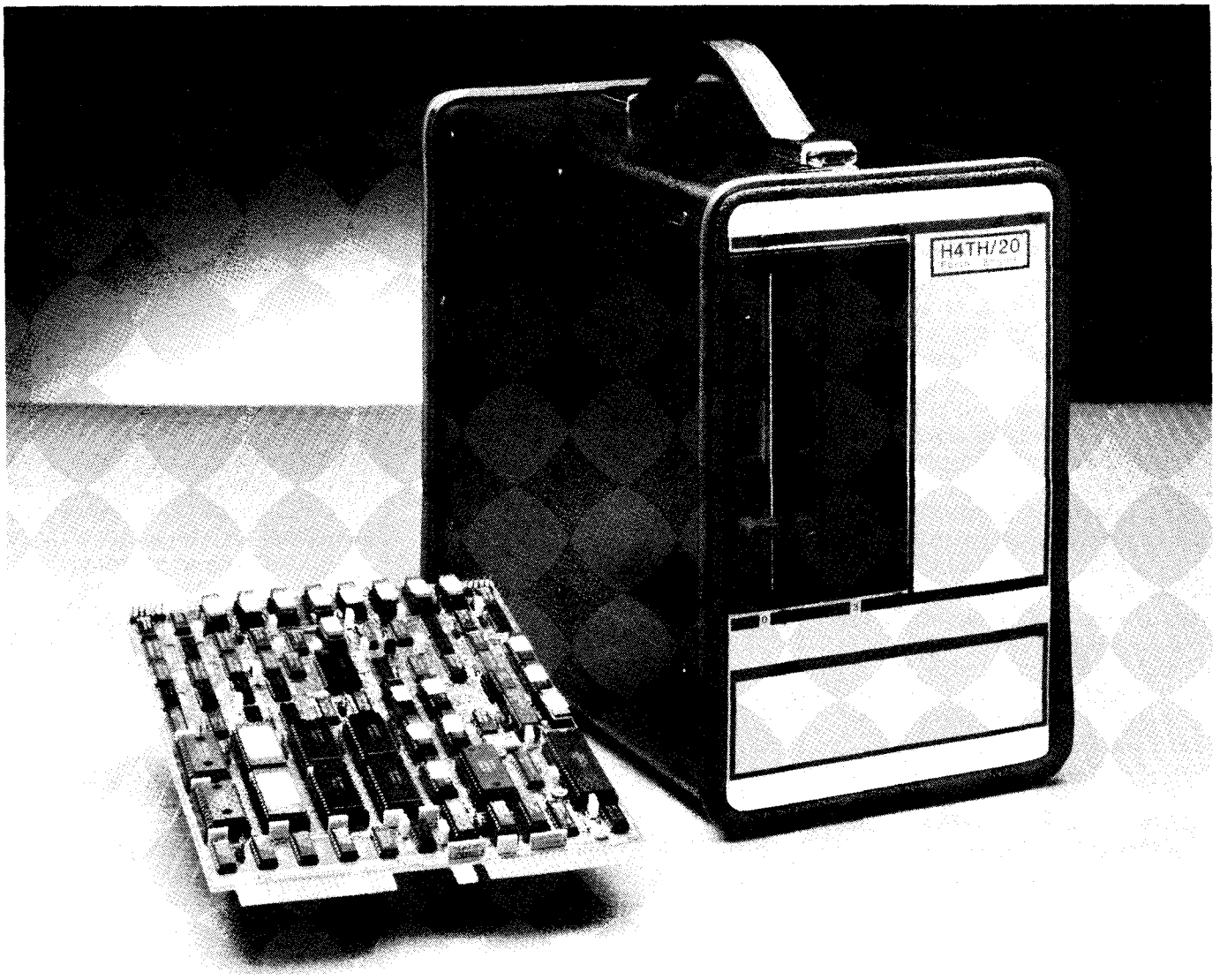
MICROPROGRAMMED BIT SLICE FORTH ENGINE
.Microcoded forth kernel
.Microcoded forth primitives
.Multi-level task switching architecture for real time applications
.Optional writable control store

H.FORTH OPERATING SYSTEM
.Hierarchical file system
.Monitor level for program debug
.Multi-user multi-tasking
.Target compiler
.I/O management
.Forth 83 Compatible

H4TH/01 OEM SINGLE BOARD
.Floppy disk controller
.2 channel SIO to 38.2K baud
.Calendar clock–4HR backup
.44K Byte ram 200NS
.32K Byte EPROM operating system
.1K X 32 microprogram memory 70ns

H4TH/10 DESKTOP
.Dual 0.8m Byte floppys
.H4TH/01 processor
.Three user slots
.Two expansion slots
.Power & cooling

H4TH/20 DESKTOP
.10 m Byte Winchester
.0.8 m Byte floppy
.H4TH/01 processor
.300K byte RAM expandable 2m byte
.Three user slots
.One expansion slot
.Power & cooling



A forth-engine consisting of a state-of-the-art integrated hardware/software system giving unsurpassed performance for professionals and their applications from a company that is totally dedicated to the forth concept and its implementation.

HARTRONIX, Inc. 1201 North Stadem Drive Tempe, Arizona 85281 602.966.7215

Synonyms and Macros

Synonyms

Victor H. Yngve
Chicago, Illinois

A startling thing beginners at Forth learn quite early is that a function can easily be renamed for convenience. For example,

```
: .S .STACK ;
: edit EDIT ;
```

Or even, to make a point with other beginners:

```
: PRINT . ;
: FETCH @ ;
```

This is fine as far as it goes, and it is widely used. But the technique will not always work. Consider this attempt at defining a fig-FORTH **R** as **R@** in a Forth-83 system, so as to make it easier to convert an old program to run on a new system:

```
: R R@ ;
```

This will not work, because the execution of **R** will place a return on the return stack, covering up the item wanted.

Then how about

```
: R COMPILE R@ ; IMMEDIATE
```

This will work fine here, but the technique will only work while compiling; it cannot be used for the **.S** and edit examples, which are to be executed from the console.

Consider the case of

```
: -DUP COMPILE ?DUP ; IMMEDIATE
```

This will work when compiling, but one would want it to work from the console, too, so try

```
: -DUP ?DUP ;
```

But this is less than optimal for a word that might be compiled into an inner loop, because it will run slower, on account of the extra nesting and unnesting of the colon definition. In cases like this, neither method is completely adequate, since the choice of the best method of definition will depend on how the word is to be used.

Note, too, that neither technique can be used for immediate words. For these one needs

```
: ENDIF [COMPILE] THEN ; IMMEDIATE
```

And if you don't happen to remember whether the word you want to make a synonym of is immediate or not, you will have to look it up. Furthermore, for any words that are both immediate and operate on the return stack, it may be that none of these methods will work, and it is not at all clear how to define synonyms for them.

One must conclude that the situation is less than satisfactory. For one and the same function, namely, simply defining a synonym or alias for a word, there are three different methods and they are all hedged by ifs and buts. It makes the language harder to learn and harder to use.

The word **SYNONYM** presented here provides a single simple way for defining synonyms. The usage of the word is

SYNONYM < new-name> <old-name>

A Forth-83-style glossary entry is provided on screen 47.

A word defined by **SYNONYM** is functionally identical to the word it is a synonym of, and it runs just as fast. The only differences are that the compilation address obtained by tick (') will be different, and if this different compilation address is executed, it will run slower. And the new word will be marked immediate, even though it will have the same action — immediate or nonimmediate — as the original word. By means of **SYNONYM** one can make a synonym for any word in the dictionary. One can even make synonyms of synonyms, with no run-time penalty. It is actually possible to switch the names of two words (proving that the technique is completely general): suppose two words, A and B, have actions X and Y. Then, using an intermediate name C, one can define

```
SYNONYM C A
SYNONYM A B
SYNONYM B C
```

Now A has the function Y, and B has the function X.

When should this method of defining synonyms be used? One can make a strong case for including this word in a system for its general utility and in order to rationalize and clean up one corner of Forth. Having the word in the system, it should always be used

```
Screen # 46
  0 ( Usage: SYNONYM <new-name> <old-name>   Forth-83 11/28/84 vhy)
  1
  2 : SYNONYM     ( -- )
  3     CREATE                          ( make header for new word )
  4     32 WORD FIND DUP                ( old word found? )
  5     IF SWAP ,                       ( yes, compile its cfa )
  6        IMMEDIATE                    ( make new word immediate )
  7        1+                           ( was old word immediate? )
  8        IF   DOES> @ EXECUTE         ( yes, set new to execute )
  9        ELSE DOES> STATE @           ( no, set new to check state )
 10             IF   @ ,                ( and compile if compiling )
 11             ELSE @ EXECUTE          ( or execute if executing )
 12             THEN
 13        THEN
 14     ELSE 1 ABORT" not found"        ( old word not found )
 15     THEN  ;

Screen # 47
  0 ( SYNONYM  Glossary entry )
  1
  2
  3 SYNONYM          --              I
  4   A defining word used in the form:
  5         SYNONYM <new-name> <old-name>
  6   Create a dictionary entry for <new-name> so that when
  7   <new-name> is later used, it will have substantially the same
  8   action that <old-name> would have had.  If <old-name> was
  9   immediate, the action will be immediate, otherwise not.
 10   During compilation, the action is to compile the same thing
 11   that <old-name> would have compiled.
 12
 13
```

DASH, FIND & ASSOCIATES

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

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

Committed to Excellence

instead of the other methods for defining synonyms. It produces synonyms that run faster than colon definitions, give no problems with the return stack, have the same immediate or nonimmediate action of the original word, can always be executed from the console or compiled, just as could the old word, and take up less space in memory because their parameter fields contain just one address rather than two or three.

A synonym facility has obvious utility for providing abbreviations of words for use at the console, as

```
SYNONYM SYN SYNONYM
SYN P PRINTER
SYN D DUMP
```

But the use of abbreviations on screens should generally be avoided in the interest of readability; they could make a program rather cryptic.

The proper use of synonyms on screens can enhance the readability of a program. Normally, the names of the definitions that are part of an application program are carefully chosen to reflect their function relative to the application. But mixed in with these application-relative words are Forth-relative words like **DUP** and **2DUP**, or @ and **C@**. Understanding the meaning of these relative to the application requires not only understanding the logic of the application program but also the particular way in which the application functions are mapped onto Forth. Thus, readability and understandability suffer. With the judicious use of synonyms, the logic of the application program can be written mainly in application-specific terminology, with the translation into Forth-relative words being given in a preamble containing colon definitions, synonyms and macros. Macros will be the topic of the second of these articles.

I wonder, since there is no run-time penalty for using synonyms, if it would be possible or desirable to develop multi-purpose routines? For example, a routine might operate in single precision or in double precision depending on which preamble is used, or on which vocabulary it is loaded under. Or a routine might sort bytes, words, double words, fields or pointers to strings.

A synonym facility would also make possible the easy preparation of bilingual Forth systems for other languages, such as Portuguese, Turkish, French, Dutch or Finnish[1]. The synonyms listed on the translation screens would resemble the entries in a traveler's pocket dictionary:

```
SYNONYM SI IF
```

This might lead to new national standards of Forth nomenclature in language areas where the political, linguistical and educational realities favor or even dictate the use of the local language. Remember that readability, like beauty, is in the eye of the beholder. This alone, independent of the other obvious merits of Forth as a teaching language, might give Forth an edge in the local schools over other languages, where changing the names of commands is not as easy. The translation kit screens could be preloaded in the system. Or, for the occasional application that needs the extra memory, synonyms not used by the application could simply be commented out before loading the conversion screens.

The definition of **SYNONYM** on screen 46 is programmed entirely in Forth-83 and should be completely portable across Forth-83 systems. The routine provides a good exercise in the use of the defining-word technique in Forth. The programming is fairly straightforward, except that the embedding of **DOES>** twice in an **IF ... ELSE ... THEN** construct after a single **CREATE** may raise some eyebrows. It is perfectly legal, however, and works beautifully. It allows the single defining word **SYNONYM**, with a single **CREATE**, to make two different families of words: a family with the behavior of immediate words for synonyms of immediate words, and a family with nonimmediate behavior for synonyms of nonimmediate words. This is necessary because the usage chosen of

SYNONYM <new-name> <old-name>

puts the new name first, as is done with colon (:), **VARIABLE**, **CONSTANT** and **CREATE**, so it is not known which class the new word should belong to until after the new name field has been

created and the old word has been found in the dictionary.

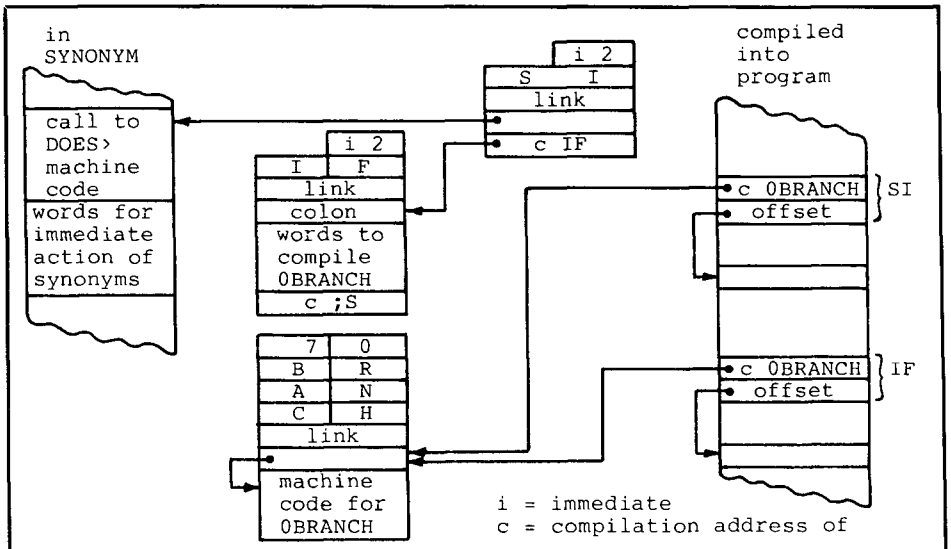
The routine first creates a name field and link field for the new word. Then, in the body (parameter field), it compiles the compilation address (CFA) of the old word. Then in the code field it compiles the address of one of the two **DOES>** expressions in the defining word. Which one is compiled depends on whether or not the old word was immediate. There is an error exit in case the old word could not be found in the dictionary.

The result is shown in the third column of Figure One for a synonym of the nonimmediate word **R@**, and in the third column of Figure Two for a synonym of the immediate word **IF**.

The new word is made immediate so that the proper **DOES>** part of the defining word will always be executed when the new word is interpreted from the console or from a screen. If the old word was immediate, one of the **DOES>** expressions (the first one) will fetch the compilation address of old word from the body of the the new word and execute it. If the old word was not immediate, the other **DOES>** expression (the second one) will either fetch and compile the compilation address or it will fetch and execute the compilation address, depending on whether or not the interpreter is in the compile state.

Thus, when the new word is interpreted from the console or from a screen, the appropriate **DOES>** part will either compile or execute the old word, just as if the old word had been interpreted. And since what is compiled is the same as what would have been compiled had the old word been used, as shown in the figures, the run-time behavior will be identical to that of the old word.

If the compilation address of the new word is obtained by tick and then either executed, or compiled and executed later, one or the other of the **DOES>** expressions will fetch the compilation address of the old word from the body of the new word and execute it, just as if the compilation address of the old word had instead been obtained directly by tick; but because of the execution of the **DOES>** expression, it will run slower.

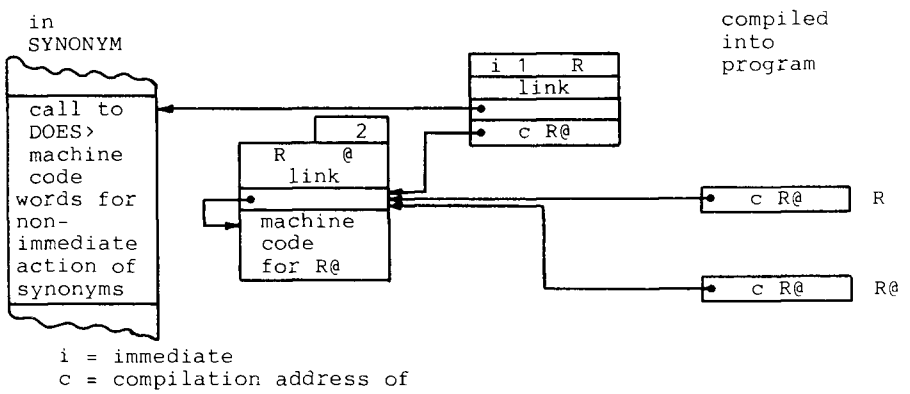


Operation of a synonym of a nonimmediate word. The expression

SYNONYM R R@

compiles into the dictionary the nonimmediate synonym definition of R in the third column. When R is encountered during compiling, the second DOES► expression in SYNONYM (first column) is executed, which obtains the compilation address of R@ from the body of R and compiles it into the program (fourth column). This is the same result as when R@ is encountered while compiling. When R is encountered in the noncompiling state, the DOES► expression obtains the compilation address of R@ from the body of R and executes it, just as if R@ had been encountered.

Figure One



Operation of a synonym of an immediate word. The expression

SYNONYM SI IF

compiles into the dictionary the immediate synonym definition of SI in the third column. When SI is encountered during compilation, the first DOES► expression in SYNONYM (first column) is executed, which obtains the compilation address of IF from the body of SI and executes it. This provides the immediate action of executing IF. The execution of IF compiles the compilation address of 0BRANCH into the program (fourth column) and arranges for compiling a branch offset after it, the same as when IF is encountered while compiling.

Figure Two

References

1. Huang, Timothy. "First Chinese Forth: A Double-Headed Approach." *Dr. Dobb's Journal* Vol. 9, No. 6 (June 1984).

Synonyms and Macros

Macros

Victor H. Yngve
Chicago, Illinois

One of the many advantages of Forth is the ease with which new features can be added to the language. Described here is a facility that provides Forth with high-level macro definitions[1] in addition to the colon definitions and code definitions already available. This offers some additional possibilities and tradeoffs.

A high-level macro definition has the readability of a colon definition but, like a code definition, it provides some increase in execution speed (though not as much). It does not place a return on the return stack like a colon definition does, and this can sometimes be an advantage.

Except for this, a macro definition can be used like a colon definition. It is written in the same way, except the word **MACRO** is substituted for the colon, and **END-MACRO** is substituted for the semicolon.

The expression

MACRO <name> ... END-MACRO

compiles a macro definition into the dictionary. When the word so defined is later interpreted from the console or from a screen, it will either be executed like a colon definition or, when compiling, the compiled words of the macro definition will be moved in-line into the body of the word being compiled.

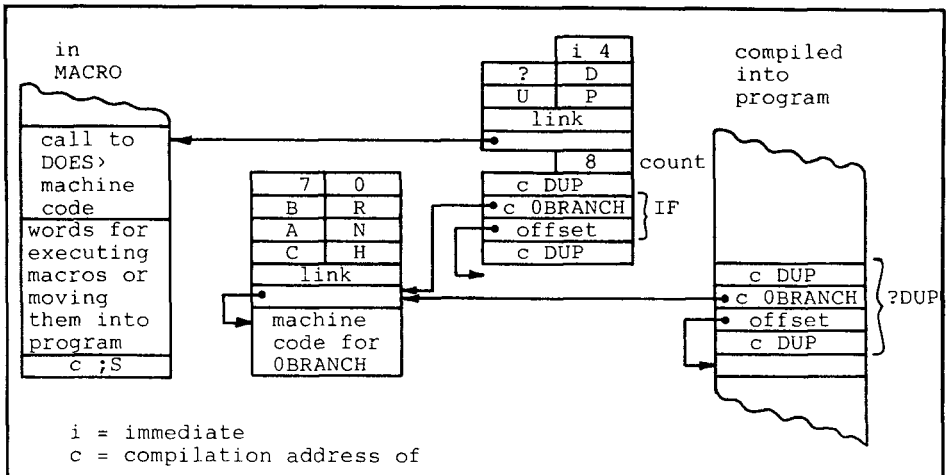
To take an example, some Forths include as a code definition primitive the word **-ROT**, which is the opposite of **ROT** in that when it rotates the top three elements on the stack, it leaves the top one the deepest. For Forths that do not have this word, the textbook way of defining it uses a colon definition:

```
: -ROT  ROT ROT  ;
```

An alternative way would be to define it as a macro:

```
MACRO -ROT  ROT ROT  END-MACRO
```

When executed from the console, the three definitions of **-ROT** all have the



Operation of a macro definition. The expression

MACRO ?DUP DUP IF DUP THEN END-MACRO

compiles into the dictionary the macro definition for ?DUP in the third column. When ?DUP is encountered during compilation, the words in the DOES► expression in MACRO (first column) move the compiled code for

DUP IF DUP THEN

from the body of the macro definition into the program (column four), which is the same result as if

DUP IF DUP THEN

had been encountered during compilation instead of ?DUP.

Figure One

```
Screen # 51
  0 ( MACRO 1  MACRO          LMI Z-80 FORTH 3.01     2/23/85     vhy )
  1 ( Usage: MACRO <name> ... END-MACRO )  ( Like  : <name> ... ; )
  2 : MACRO                  ( start compiling a macro  )
  3     ?EXEC                ( -error if not executing )
  4   ( !CSP )               ( *moved to line 13 after HERE )
  5     CURRENT @            ( -current vocabulary )
  6     CONTEXT @            ( -                   )
  7     ?CLR_HASH            ( -                   )
  8     CONTEXT !            ( -becomes context    )
  9   BUILD                  ( -create heading for macro )
 10     SMUDGE               ( -make macro unavailable )
 11     1 ALLOT              ( space for byte count in macro )
 12     HERE    ( -- addr )  ( for use by END-MACRO )
 13     !CSP                 ( -save stack position for check )
 14     COMPILER             ( -start compiling words into macro )
 15   ( ;CODE ... )  -->     ( *replaced by does> ... as follows )

Screen # 52
  0 ( MACRO 2  MACRO cont. )
  1   DOES>     ( pfa -- )   ( move words of macro into place )
  2     STATE @              ( is macro being used while )
  3     IF                             ( compiling? )
  4       COUNT HERE         (   -- pfa+1 n here )
  5       OVER ALLOT         ( space for macro words in program )
  6       SWAP CMOVE         ( move macro words into program )
  7     ELSE                           ( interpreting? )
  8       ['] CR @  HERE !   ( set colon definition cfa at HERE )
  9       COUNT SWAP OVER    (   -- n pfa+1 n )
 10       HERE 2+            ( address for macro words )
 11       SWAP CMOVE         ( move macro words into definition )
 12       HERE 2+ +          ( address for ;S in definition )
 13       ['] ;S  SWAP !     ( set ;S into definition )
 14       HERE EXECUTE       ( now execute this colon definition )
 15     THEN  ;     -->
```

```
Screen # 53
  0 ( MACRO 3  END-MACRO )
  1
  2 : END-MACRO ( addr -- )  ( stop compiling a macro )
  3     ?COMP                ( -error if not compiling )
  4     ?CSP                 ( -error if stack position not same )
  5   ( COMPILE ;S )         ( *replaced by the following )
  6     HERE                 (   -- start-addr  end-addr )
  7     OVER -               ( calculate byte count )
  8     SWAP 1-  C!          ( install count in new definition )
  9     IMMEDIATE            ( make new definition immediate )
 10     UNSMUDGE             ( -make new definition available )
 11     R> DROP              ( -prepare to exit from COMPILER )
 12     STATE OFF            ( -stop compiling macro words )
 13   ;                      ( -exit from COMPILER )
 14     IMMEDIATE            ( -make END-MACRO immediate )
 15
```

same effect on the stack. The differences come when they are compiled into a program to be run later.

In the case of the code definition for **-ROT**, the compilation address of **-ROT** is compiled into the program. (We assume here the usual fig-FORTH type of implementation.) Then, when the program is run, the address interpreter transfers control to the machine code of the definition, which is run. This is the fastest.

In the case of the colon definition for **-ROT**, the compilation address of the word is again compiled into the program. But this time when it is run, the address interpreter must first nest down by placing a return on the return stack, corresponding to the colon. Then it must execute the body of the colon definition, which means executing the primitive **ROT** twice. Then it must retrieve the return off the return stack, nesting back up, corresponding to the semicolon. This is the slowest.

In the case of the macro definition of **-ROT**, the compilation address of **-ROT** is not compiled into the program. Instead, the compilation addresses of **ROT ROT** from the body of the macro are compiled. This takes up more room in the program, but when it is run, the nesting down and up are eliminated. The speed of this is intermediate between the code definition and the colon definition.

The macro facility has few restrictions. Anything that can be programmed as a colon definition that results in position-independent code in the body of the definition can be programmed as a macro. That means that even such constructs as

IF ... ELSE ... THEN
and
DO ... LOOP

can be included in a macro if the Forth implementation uses relative branch addresses (offsets). Not all Forths do. To find out whether your implementation compiles position-independent code, compile

```
: TRIAL   IF DROP THEN  ;
: TRIAL   IF DROP THEN  ;
```

If the code is position independent, a memory dump will show that the two compiled definitions are identical except for their link fields. In particular, the branch addresses will be offsets, and will be the same, probably 0004.

Macros can be embedded in macros to any level desired. For example, one Forth has primitives for **DNEGATE**, **D+**, **OR** and **0=**, while **D-**, **D0=** and **D=** are defined as colon definitions. These latter could be defined instead as macros:

```
MACRO D-   DNEGATE D+   END-MACRO
MACRO D0=   OR   0=   END-MACRO
MACRO D=   D-   D0=   END-MACRO
```

When **D=** is used in a program, it would compile not the compilation address of **D=**, but the compilation addresses of

DNEGATE D+ OR 0=

So we see that macros compile inline code, even when nested, and with that comes a speed advantage. But, like colon definitions, they also bring the same advantages of structured pro-

FOR TRS-80 MODELS 1, 3, 4, 4P
IBM PC/XT, AT&T 6300, ETC.

WHICH ONE?

Which microcomputer word processor lets you create and edit without retyping, but *won't* slow down your creative process? Knows when to capitalize the first letter while replacing one phrase with another? Can outdent as well as indent? Will do typesetting at your command, even with proportional characters, right justification and tabbed columns? Lets you use the same (extra-capacity) data disks on IBM PC *and* TRS-80? And eases your learning with common-sense keystrokes, Help menus, good examples and a professionally authored manual?

Hint: it can integrate to communicate from home to office, and will interface with a database for form letters, data tables, and more!

It's the professional's word processor for your IBM PC, Compaq, or TRS-80 Model 1, 3 or 4:

FORTHWRITE
in
MMSFORTH

With an unusually powerful set of tools and an unusually easy way of helping you to use them.

The total software environment for IBM PC/XT, TRS-80 Model 1, 3, 4 and close friends.

- Personal License (required):
  MMSFORTH V2.4 System Disk **$179.95**
  (TRS-80 Model 1 requires lowercase, DDEN, 1 40-track drive.)
- Personal License (additional modules):
  FORTHCOM communications module . . . . **$ 49.95**
  UTILITIES **49.95**
  GAMES **39.95**
  EXPERT-2 expert system **69.95**
  DATAHANDLER **59.95**
  DATAHANDLER-PLUS (PC only, 128K req.) **99.95**
  FORTHWRITE word processor **99.95**
- Corporate Site License Extensions . . . . . **from $1,000**
- Bulk Distribution . . . **from $500/50 units.**
- Some recommended Forth books:
  STARTING FORTH (programming) **19.95**
  THINKING FORTH (technique) **15.95**
  BEGINNING FORTH (re MMSFORTH) . . . **16.95**

Shipping/handling & tax extra. No returns on software.

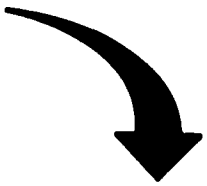
Ask your dealer to show you the world of MMSFORTH, or request our free brochure.

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

BRYTE FORTH

for the

INTEL 8031 MICRO-CONTROLLER



FEATURES

—FORTH-79 Standard Sub-Set
—Access to 8031 features
—Supports FORTH and machine code interrupt handlers
—System timekeeping maintains time and date with leap year correction
—Supports ROM-based self-starting applications

COST

130 page manual —**$ 30.00**
8K EPROM with manual—**$100.00**

Postage paid in North America.
Inquire for license or quantity pricing.

Bryte Computers, Inc.
P.O. Box 46, Augusta, ME 04330
(207) 547-3218

gramming in the source code of the program. This brings benefits in ease of programming, in readability and in ease of program maintenance.

When should a macro be used?

A macro definition should be used to provide the structured programming advantages of a colon definition where a colon definition cannot be used for some reason. For example, a colon definition inside a loop cannot use **I** and **J** to access the loop indices, because the return placed on the return stack by the colon definition would interfere. In such cases, the only recourse has been to program the words of the colon definition explicitly in the interior of the loop. This may introduce considerable clutter into the loop that the colon definition would have removed if it could have been used. With a macro, one can have the in-line code that is needed, while at the same time retaining the advantages of structured programming.

As discussed in the previous article on synonyms, macros can also be used to help increase the readability, understandability and maintainability of a program, for it becomes possible to write the logic of an application in application-relative words, with most Forth-relative words removed to a preamble by means of colon definitions, macros and synonyms.

A macro definition should always be used instead of a colon definition for routines of two or more words when execution speed considerations outweigh the extra bytes of the in-line code produced. To get an initial idea of the speed tradeoff, some timing tests were carried out. In the implementation used, **0**, **=** and **0=** were all defined as code definition primitives. The speed of the code definition **0=** was then compared with an equivalent macro definition

```
MACRO  M0=  0  =  END-MACRO
```

and with an equivalent colon definition

```
: :0=  0  =  ;
```

The execution times came out to be almost exactly in the ratio of 1 : 2 : 4, with the macro definition saving about a minute of execution time per million iterations over the colon definition (using a 4 MHz Z80 system). This was the time spent nesting down and coming back up. The timings and the ratios might be quite different for different processors and different Forth implementations, especially for processors better adapted to coding the Forth inner interpreter than is the Z80.

This means that the macro facility has a potential for speeding up programs by speeding up their frequently executed inner loops that contain colon definitions. Simply recode the colon definitions as macros. It would also be worthwhile to examine the other words used in the loop to see whether they have been implemented as primitives or as colon definitions. For example, in one Forth, **0<>** is a primitive, while in another it is a colon definition. Redefine as macros all that are colon definitions.

Of course, the user could achieve the same speed increase by simply recoding his loop by hand as a long string of primitives. But he would lose the readability and ease of programming and maintenance provided by the structured nature of Forth.

A macro facility gives Forth implementors some extra flexibility by providing another possibility besides colon and code definitions for sequences like

```
=  NOT
<  NOT
AND  NOT
SWAP  DROP
ROT  ROT
OVER  OVER
```

and so on. One of the advantages of using macros over code definitions in an implementation of Forth is that they are high level. This means that it would be easier to move the implementation to another processor.

The use of macros might be particularly significant in a Forth implementation built around a hardware Forth machine that has only a few dozen words coded in microcode. It would provide another option besides colon definitions for the rest of the words.

A macro facility can play a part in a kit to allow programs written in one dialect of Forth to run in a system implementing another dialect. Here, speed considerations and noninterference with the return stack would provide important advantages. In one program of this sort, an overall speed penalty of roughly 50% has been reported[2]. A recoding using macros and synonyms should reduce the penalty substantially.

A macro definition, like a colon definition, can contain one or more words, or even none. That means a one-word macro could be used in place of a synonym. That is not normally advisable, however. As a rule of thumb, always use a synonym to replace a colon definition with one word in it, and a macro to replace a colon definition with two or more words in it.

It is instructive to compare the difference between macros and synonyms. The action of a synonym with regard to the immediate/nonimmediate distinction is deferred, where the action of a macro is not. Suppose we define **SI** as a synonym of **IF**:

```
SYNONYM SI  IF
```

As we saw in Figure Two of the previous article, the compilation address of **IF** is compiled into the parameter field of **SI** even though **IF** is an immediate word. Then, when **SI** is used in a program, it has the same effect that using **IF** would have had. Namely, it executes as an immediate word and compiles a conditional branch into the program and arranges for a branch offset. This conditional branch will be executed at a still later time, when the program is run.

But when **IF** is compiled into a macro for **?DUP**

```
MACRO ?DUP  DUP IF DUP THEN  END-MACRO
```

the **IF** will have its (immediate) action now, and a conditional branch with its branch address will be compiled into the macro definition in exactly the same way that it would be compiled into a colon definition. (See Figure One of this article.) Then, when the macro **?DUP** is used in a program, the body of the macro definition compiled for **?DUP**, including the conditional branch and its branch offset, is moved (copied) bodily into the program, so that the words of the macro definition can then be executed in line at a still later time when the program is run. Since the code is copied in this way to a different location to be run, it must not contain any absolute branch addresses, but only relative branch addresses (offsets) if the implementation provides them, or else no branches at all.

Although synonyms may be defined for immediate words as well as for nonimmediate words, there would be no advantage in using a macro definition to replace a colon definition that is marked immediate, and this possibility is not provided for.

The definitions shown in screens 51-53 are fairly straightforward. First, **MACRO** compiles a name and link field for the heading of the macro definition. It then allots space in the body of the definition (parameter field) for a count byte. Then the compiler is called, which starts compilation, and the words of the definition are compiled into the macro in the same way in which they would be compiled into a colon definition.

Compilation is stopped by **END-MACRO**, which calculates the number of bytes of dictionary space used for compiling the words of the macro, and places a length byte in the space previously allotted for it. The length byte and the words of the definition then constitute a string that can be moved out of the definition as needed.

The macro definition is then made immediate, so that the **DOES>** part of **MACRO** will be executed when the macro is later interpreted from the console or a screen, and the words of the definition will then be moved out appropriately. **END-MACRO** then drops its own return off the return stack so that control will return to **MACRO** when **END-MACRO** is finished. Then compilation is stopped. When control returns from **END-MACRO**, the word **DOES>** alters the code field of the macro being defined, so that the **DOES>** part of **MACRO** will be executed later when the macro definition is used.

FORTH

FREEDOM OF CHOICE

SOTA Computing Systems Limited lets you choose between either the versatile figFORTH model or the popular 79 Standard. Each version is available for a number of popular computer systems including the IBM PC, XT and AT (or compatibles); the TRS-80 Model I, III and 4/4P, or any computer system running CP/M (version 2.x) or CP/M Plus (version 3.x). What's more, SOTA doesn't require you to enter into any awkward or expensive royalty or licensing arrangements. As long as your applications programs do not offer the end user access to the basic FORTH system, you are free to make as many copies of the compiled FORTH system as you please and distribute them as you wish. FORTH from SOTA is the FORTH of choice for both the novice and experienced programmer. Make it your choice now! Order your copy today.

79 STANDARD

fig MODEL

IBM® TRS-80 CP/M

When you order from SOTA, both the fig model and 79 standard come complete with the following extra features at no additional charge:

• full featured string handling • assembler • screen editor • floating point • double word extension set • relocating loader • beginner's tutorial • comprehensive programmer's guide • exhaustive reference manual • unparalleled • technical support • source listings • • unbeatable price •

ORDER FORM

GENTLEMEN: Rush me my order!
☐ Enclosed is my ☐ check ☐ money-order
☐ Please bill my ☐ VISA ☐ MasterCard
for $89.95
TOTAL US funds

Please send me: ☐ 79 Standard FORTH ☐ figFORTH model for the
☐ IBM PC ☐ XT ☐ AT (and compatibles)
☐ TRS-80 Model I ☐ Model III ☐ Model 4 ☐ Model 4P
☐ CP/M Version 2.x ☐ CP/M Plus (Version 3.x)
For CP/M versions please note 5 1/4" formats only and please specify computer type:

NAME: _____
STREET: _____
CITY/TOWN: _____
STATE: _____ ZIP: _____
CARD TYPE: _____ EXPIRY: _____
CARD NO: _____
SIGNATURE: _____

ORDER TODAY 213-1080 Broughton Street Vancouver, British Columbia Canada • V6G 2A8

MasterCard • Order by Mail or Phone • (604) 688-5009 • VISA

State of the Art since 1981

SOTA Computing Systems Limited

IBM, TRS-80 and CP/M are registered trademarks of International Business Machine Corporation, Radio Shack and Digital Research respectively

ATTENTION: ENGINEERS PROGRAMMERS

PolyFORTH® II

the operating system and programming language for real-time applications involving ROBOTICS, INSTRUMENTATION, PROCESS CONTROL, GRAPHICS and more, is now available for...

IBM PC*

PolyFORTH II offers IBM PC users:

- **Unlimited control tasks**
- **Multi-user capability**
- **8087 mathematics co-processor support**
- **Reduced application development time**
- **High speed interrupt handling**

Now included at no extra cost: Extensive interactive **GRAPHICS SOFTWARE PACKAGE!** Reputed to be the fastest graphic package and the only one to run in a true multi-tasking environment , it offers point and line plotting, graphics shape primitives and interactive cursor control.

PolyFORTH II is fully supported by FORTH, Inc.'s:

- **Extensive on-line documentation**
- **Complete set of manuals**
- **Programming courses**
- **The FORTH, Inc. hot line**
- **Expert contract programming and consulting services**

From FORTH, Inc., the inventors of FORTH, serving professional programmers for over a decade.

Also available for other popular mini and micro computers.

For more information contact:

FORTH, Inc.

2309 Pacific Coast Hwy.
Hermosa Beach,
CA 90254
213/372-8493
RCA TELEX: 275182

Eastern Sales Office
1300 N. 17th St.
Arlington, VA 22209
703/525-7778

*IBM PC is a registered trademark of International Business Machines Corp.

Nothing is compiled at the end of the macro definition that would correspond to the **;S** or other word that is compiled at the end of a colon definition by the semicolon.

When the macro definition is later interpreted from the console or screen in compile state, the **DOES>** part of **MACRO** moves the words of the macro definition as a block into the program being compiled, to be later executed in line rather than being called by nesting to a lower level as with a colon definition.

In execute state, it moves the words of the macro definition into a temporary colon definition constructed at **HERE** and executes it. To this end, the address of the run-time code for colon (:) is obtained by **['] CR** and set in place at **HERE**. Then the compiled macro words are moved into the following locations. Finally, the address of **;S** is set in place at the end. Then this temporary colon definition is executed.

These definitions of **MACRO** and **END-MACRO** have been checked out under Laboratory Microsystems Z-80 Forth 3.01, a Forth-83 implementation. These words are, of necessity, implementation specific, since they make assumptions about the return stack, and they address into the parameter field of the word being compiled. The assumptions they make, however, are not at all unusual.

It was initially thought that **MACRO** and **END-MACRO** should be patterned after an array definition with **CREATE** using **]** and **[** to start and stop compilation. But it soon developed that one would want more of the features of colon and semicolon in the definitions, so it seemed better to pattern them after the definitions of colon and semicolon found in the implementation, making the appropriate modifications.

The lines on the screens with comments beginning with a single hyphen are copied (with LMI's permission) intact from the definitions of colon and semicolon. The lines with comments beginning with an asterisk show modifications. The rest of the lines have been added to implement the special features of **MACRO** and **END-MACRO**.

Thus, to get the definitions to work on a different system, obtain a listing of the definitions of colon and semicolon in your implementation, or at least obtain the compiled code using a decompiler or memory dump. Then identify the various functional parts of these definitions as implemented in your system, referring to the fig-FORTH model[3,4], or other sources if needed for help in understanding them. Then make the modifications indicated. This process should not be difficult, and will be a learning experience if you don't yet know much about the Forth compiler.

On sixteen-bit machines where it is advantageous or required to have addresses start at sixteen-bit word boundaries, change the **1 ALLOT** on screen 51 to **2 ALLOT** and add **1+** after the **DOES>** on screen 52. (It is an interesting aspect of the Forth-83 Standard that **SYNONYM** could be written entirely within the standard, but **MACRO** and **END-MACRO** could not.)

References

1. Soreff, Jeffrey. "Macro Expansion in Forth," *Forth Dimensions* V/5.

2. Berkey, Robert. "Forth-83 Program to Run Forth-79 Code," *Forth Dimensions* VI/4.

3. Ragsdale, William F. *fig-FORTH Installation Manual: Glossary, Model, Editor*. San Carlos, CA: Forth Interest Group, 1980.

4. Derick, Mitch and Linda Baker. *FORTH Encyclopedia: The Complete Forth Programmer's Manual*, 2nd ed. Mountain View, CA: Mountain View Press, 1982.

Forth Timer Macros

Iram Weinstein
McLean, Virginia

There comes a time in the development of many applications when everything works and attention turns to speeding up the run time. The first step in this phase of development is to determine which words contribute most to the running time of the application. A typical scheme for this might involve temporarily changing the definition of each word to be tested, as in Figure One, where **NOW** is a word that puts the current clock time on the stack.

After recompiling and running the application with this augmented definition of **WORD**, **T** will contain the accumulated time used by **WORD**. Each word in the application can be tested in turn. The inconvenience of having to insert temporary modifications into colon definitions, combined with the desire to count the number of uses of each word in the application, led to screens 52-55 shown here. These screens define a set of Forth macros (*Forth Dimensions* V/5) that redefine the defining words : and ; to automatically insert the timing actions into any words subsequently compiled. The technique is similar to that used in the article "Tracer for Colon Definitions" (*Forth Dimensions* V/2).

Screen 52 contains some words used to measure time. These will be system dependent. The words shown here are for the Commodore-64, which has an interrupt-driven, three-byte, "jiffy" counter at locations 160-162. Location 162 is incremented once each interrupt cycle, approximately sixty per second. Only the lower two bytes of this counter are used here, giving a timing range of over 1000 seconds.

FREEZE and **UNFREEZE** prevent the overhead time consumed by the timing macros themselves from accumulating into the measured time. **FREEZE** moves the current time value into the variable **VNOW**, while **UNFREEZE** resets the jiffy clock back to its value when **FREEZE** was executed. In some systems, these words might be written to actually stop

```
SCR # 52
 0  ( WORD TIMER MACROS-1)
 1
 2  161 CONSTANT JIFFY
 3  VARIABLE VNOW
 4  : INITNOW 0 JIFFY ! ;
 5  : FREEZE JIFFY @ VNOW ! ;
 6  : UNFREEZE VNOW @ JIFFY ! ;
 7  : NOW VNOW DUP C@ 256 * SWAP 1+ C@ + ;
 8
 9  : ARRAY CREATE 2* ALLOT DOES> SWAP 2* + ;
10  20 ARRAY #USES
11  20 ARRAY TCUM
12  VARIABLE WORD#
13  : INIT 0 WORD# ! 0 #USES 40 ERASE 0 TCUM 40 ERASE ;
14  53 LOAD 54 LOAD 55 LOAD INIT INITNOW
15


SCR # 53
 0  ( WORD TIMER MACROS-2)
 1
 2  1000 60 2CONSTANT CONV ( CONVERT TO MILLI-SECS)
 3
 4  : TTOT ( N---D) TCUM @ 0 CONV U*/ ;
 5  : TAVG ( N---D) DUP TTOT ROT #USES @ DUP 0= +1000 SWAP U*/ ;
 6
 7  : .HEADER ." WORD" 13 SPACES ." #USES   AVG. MSECS" ;
 8
 9  : FMT <# # # # 46 HOLD #S #> ;
10
11  : TYPE.R ( ADDR NCHAR NWIDTH ---) OVER - SPACES TYPE ;
12
13  : .DATA ( N---) DUP #USES @ 7 .R TAVG FMT 13 TYPE.R CR ;
14
15


SCR # 54
 0  ( WORD TIMER MACROS-3)
 1
 2  FIND : @ CONSTANT DOCOLON
 3  : CHECK ( NFA---NFA)
 4  ( EXAMINES WORD FOR DOCOLON IN CFA. IF FOUND, RETURNS NFA)
 5  ( ELSE, SCANS LINKAGE UNTIL DOCOLON IS FOUND)
 6   BEGIN PFA DUP CFA @ DOCOLON = NOT
 7     IF LFA @ 0 ELSE NFA 1 ENDIF UNTIL ;
 8  : SUMMARY ( --- ) CR CR .HEADER CR
 9    [COMPILE] FORTH CONTEXT @ @ 0 WORD# @ DO
10        CHECK DUP ID.
11        14 OVER C@ 127 AND - SPACES
12        I 1- .DATA
13        PFA LFA @
14    -1 +LOOP DROP
15    INITNOW ;
```

(Continued on page 25)

Write it once!

MasterFORTH

Portable programming environment

Whether you program on the **Macintosh,** the **IBM PC,** an **Apple II** series, a **CP/M** system, or the **Commodore 64,** your program will run unchanged on all the rest. If you write for yourself, MasterFORTH will protect your investment. If you write for others, it will expand your marketplace.

MasterFORTH is a state-of-the art implementation of the Forth computer language. Forth is interactive – you have immediate feedback as you program, every step of the way. Forth is fast, too, and you can use its built-in macro assembler to make it even faster. MasterFORTH's relocatable utilities, transient definitions, and headerless code let you pack a lot more program into your memory. The resident debugger lets you decompile, breakpoint, and trace your way through most programming problems. A string package, file interface, and full screen editor are all standard features.

MasterFORTH exactly matches the Forth-83 Standard dialect described in *Mastering Forth* by Anderson and Tracy (Brady, 1984). The standard package includes the book and over 100 pages of supplementary documentation.

MasterFORTH standard package	
Macintosh	$125
IBM PC and PC Jr. (MS DOS 2.1)	125
Apple II, II+, IIe, IIc (DOS 3.3)	125
CP/M 2. x (IBM 3740 8")	125
Commodore 64	100
Extensions	
Floating Point (1984 FVG standard)	$60
Graphics (Apple II series)	60
Module relocator (with utility sources)	60
Printed source listing (each)	35
Publications	
Mastering Forth (additional copies)	$18
Thinking Forth by Leo Brodie	16
Forth-83 International Standard	15
Rochester Bibliography, 2nd ed.	15
1984 Rochester Conference	25
1984 FORML Conference	25

MICROMOTION

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

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

Mulituser/Multitasking for 8080, Z80, 8086

Industrial Strength FORTH

TaskFORTH™

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

LOADS OF TIME SAVING PROFESSIONAL FEATURES:

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

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

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

Available on 8" or 5¼" disk in various formats under CP/M 2.2 or greater and 5¼" MS-DOS

FULLY WARRANTIED, DOCUMENTED AND SUPPORTED

VISA DEALER INQUIRIES INVITED MasterCard

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

FORTH INTEREST GROUP MAIL ORDER FORM

P.O. Box 8231 San Jose, CA 95155 (408) 277-0668

MEMBERSHIP

IN THE FORTH INTEREST GROUP

107 - MEMBERSHIP in the FORTH INTEREST GROUP & Volume 7 of FORTH DIMENSIONS. No sales tax, handling fee or discount on membership. See the back page of this order form.

The Forth Interest Group is a worldwide non-profit member-supported organization with over 5,000 members and 80 chapters. FIG membership includes a subscription to the bi-monthly publication, FORTH Dimensions. FIG also offers its members publication discounts, group health and life insurance, an on-line data base, a job registry, a large selection of Forth literature, and many other services. Cost is $20.00 per year for USA, Canada & Mexico; all other countries may select surface ($27.00) or air ($33.00) delivery.

The annual membership dues are based on the membership year, which runs from May 1 to April 30.

When you join, you will receive issues that have already been circulated for the current volume of Forth Dimensions and subsequent issues will be mailed to you as they are published.

You will also receive a membership card and number which entitles you to a 10% discount on publications from FIG. Your member number will be required to receive the discount, so keep it handy.

HOW TO USE THIS FORM

1. Each item you wish to order lists three different Price categories:

   Column 1 - USA, Canada, Mexico
   Column 2 - Foreign Surface Mail
   Column 3 - Foreign Air Mail

2. Select the item and note your price in the space provided.

3. After completing your selections enter your order on the fourth page of this form.

4. Detach the form and return it with your payment to **The Forth Interest Group**.

FORTH DIMENSIONS BACK VOLUMES

The six issues of the volume year (May - April)

101 - Volume 1 FORTH Dimensions (1979/80) $15/16/18 _____
102 - Volume 2 FORTH Dimensions (1980/81) $15/16/18 _____
103 - Volume 3 FORTH Dimensions (1981/82) $15/16/18 _____
104 - Volume 4 FORTH Dimensions (1982/83) $15/16/18 _____
105 - Volume 5 FORTH Dimensions (1983/84) $15/16/18 _____
106 - Volume 6 FORTH Dimensions (1984/85) $15/16/18 _____

ASSEMBLY LANGUAGE SOURCE CODE LISTINGS

Assembly Language Source Listings of fig-Forth for specific CPUs and machines with compiler security and variable length names.

513 - 1802/MARCH 81 ..... $15/16/18 _____
514 - 6502/SEPT 80 ..... $15/16/18 _____
515 - 6800/MAY 79 ..... $15/16/18 _____
516 - 6809/JUNE 80 ..... $15/16/18 _____
517 - 8080/SEPT 79 ..... $15/16/18 _____
518 - 8086/88/MARCH 81 ..... $15/16/18 _____
519 - 9900/MARCH 81 ..... $15/16/18 _____
520 - ALPHA MICRO/SEPT 80 ..... $15/16/18 _____
521 - APPLE II/AUG 81 ..... $15/16/18 _____
522 - ECLIPSE/OCT 82 ..... $15/16/18 _____
523 - IBM-PC/MARCH 84 ..... $15/16/18 _____

524 - NOVA/MAY 81 ..... $15/16/18 _____

525 - PACE/MAY 79 ..... $15/16/18 _____

526 - PDP-11/JAN 80 ..... $15/16/18 _____

527 - VAX/OCT 82 ..... $15/16/18 _____

528 - Z80/SEPT 82 ..... $15/16/18 _____

BOOKS ABOUT FORTH

200 - ALL ABOUT FORTH ..... $25/26/35 _____
Glen B. Haydon
An annotated glossary for MVP Forth; a 79-Standard Forth.

205 - BEGINNING FORTH ..... $17/18/21 _____
Paul Chirlian
Introductory text for 79-Standard.

215 - COMPLETE FORTH ..... $16/17/20 _____
Alan Winfield
A comprehensive introduction including problems with answers. (Forth 79)

220 - FORTH ENCYCLOPEDIA ..... $25/26/35 _____
Mitch Derick & Linda Baker
A detailed look at each FIG-Forth instruction.

225 - FORTH FUNDAMENTALS, V. 1 ..... $16/17/20 _____
Kevin McCabe
A textbook approach to 79 Standard Forth.

230 - FORTH FUNDAMENTALS, V. 2 ..... $13/14/16 _____
Kevin McCabe
A glossary.

233 - FORTH TOOLS ..... $19/21/23 _____
Gary Feierbach & Paul Thomas
The standard tools required to create and debug Forth-based applications.

237 - LEARNING FORTH ..... $17/18/21 _____
Margaret A. Armstrong
Interactive text, introduction to the basic concepts of Forth. Includes section on how to teach children Forth.

240 - MASTERING FORTH ..... $18/19/22 _____
Anita Anderson & Martin Tracy (MicroMotion)
A step-by-step tutorial including each of the commands of the Forth-83 International Standard; with utilities, extensions and numerous examples.

245 - STARTING FORTH (soft cover) ..... $20/21/22 _____
Leo Brodie (FORTH, Inc.)
A lively and highly readable introduction with exercises.

246 - STARTING FORTH (hard cover) ..... $24/25/29 _____
Leo Brodie (FORTH, Inc.)

255 - THINKING FORTH (soft cover) ..... $16/17/20 _____
Leo Brodie
The sequel to "Starting Forth". An intermediate text on style and form.

265 - THREADED INTERPRETIVE LANGUAGES $23/25/28 _____
R.G. Loeliger
Step-by-step development of a non-standard Z-80 Forth.

270 - UNDERSTANDING FORTH ..... $3.50/5/6 _____
Joseph Reymann
A brief introduction to Forth and overview of its structure.

FORML CONFERENCE PROCEEDINGS

FORML PROCEEDINGS - FORML (the Forth Modification Laboratory) is an informal forum for sharing and discussing new or unproven proposals intended to benefit Forth. Proceedings are a compilation of papers and abstracts presented at the annual conference. FORML is part of the Forth Interest Group

310 - FORML PROCEEDINGS 1980 ..... $25/28/35 _____
Technical papers on the Forth language and extensions.

311 - FORML PROCEEDINGS 1981 (2V) ..... $40/43/45 _____
Nucleus layer, interactive layer, extensible layer, metacompilation, system development, file systems, other languages, other operating systems, applications and abstracts without papers.

312 - FORML PROCEEDINGS 1982 ..... $25/28/35 _____
Forth machine topics, implementation topics, vectored execution, system development, file systems and languages, applications.

313 - FORML PROCEEDINGS 1983 ..... $25/28/35 _____
Forth in hardware, Forth implementations, future strategy, programming techniques, arithmetic & floating point, file systems, coding conventions, functional programming, applications.

314 - FORML PROCEEDINGS 1984 ..... $25/28/35 _____
Expert systems in Forth, using Forth, philosophy, implementing Forth systems, new directions for Forth, interfacing Forth to operating systems, Forth systems techniques, adding local variables to Forth.

ROCHESTER PROCEEDINGS

The Institute for Applied Forth Research, Inc. is a non-profit organization which supports and promotes the application of Forth. It sponsors the annual Rochester Forth Conference.

321 - ROCHESTER 1981 (Standards Conference) $25/28/35 _____
79-Standard, implementing Forth, data structures, vocabularies, applications and working group reports.

322 - ROCHESTER 1982
(Data bases & Process Control) ..... $25/28/35 _____
Machine independence, project management, data structures, mathematics and working group reports.

323 - ROCHESTER 1983 (Forth Applications) . $25/28/35 _____
Forth in robotics, graphics, high-speed data acquisition, real-time problems, file management, Forth-like languages, new techniques for implementing Forth and working group reports.

324 - ROCHESTER 1984 (Forth Applications) . $25/28/35 _____
Forth in image analysis, operating systems, Forth chips, functional programming, real-time applications, cross-compilation, multi-tasking, new techniques and working group reports.

THE JOURNAL OF FORTH APPLICATION & RESEARCH

A refereed technical journal published by the Institute for Applied Forth Research, Inc.

401 - JOURNAL OF FORTH RESEARCH V.1 #1 $15/16/18 _____
Robotics.

402 - JOURNAL OF FORTH RESEARCH V.1 #2 $15/16/18 _____
Data Structures.

403 - JOURNAL OF FORTH RESEARCH V.2 #1 $15/16/18 _____
Forth Machines.

404 - JOURNAL OF FORTH RESEARCH V.2 #2 $15/16/18 _____
Real-Time Systems.

405 - JOURNAL OF FORTH RESEARCH V.2 #3 $15/16/18 _____
Enhancing Forth.

406 - JOURNAL OF FORTH RESEARCH V.2 #4 $15/16/18 _____
Extended Addressing.

REPRINTS

420 - BYTE REPRINTS ..... $5/6/7 _____
Eleven Forth articles and letters to the editor that have appeared in *Byte* magazine.

421 - POPULAR COMPUTING 9/83 ..... $5/6/7 _____
Special issue on various computer languages, with an in-depth article on Forth's history and evolution.

DR. DOBB'S

This magazine produces an annual special Forth issue which includes source-code listings for various Forth applications.

422 - DR. DOBB'S 9/82 ..... $5/6/7 _____

423 - DR. DOBB'S 9/83 ..... $5/6/7 _____

424 - DR. DOBB'S 9/84 ..... $5/6/7 _____

HISTORICAL DOCUMENTS

501 - KITT PEAK PRIMER ..... $25/27/35 _____
One of the first institutional books on Forth. Of historical interest.

502 - FIG-FORTH INSTALLATION MANUAL .. $15/16/18 _____
Glossary model editor - We recommend you purchase this manual when purchasing the source-code listings.

REFERENCE

305 - FORTH 83 STANDARD ..... $15/16/18 _____
The authoritative description of 83-Standard Forth. For reference, not instruction.

300 - FORTH 79 STANDARD ..... $15/16/18 _____
The authoritative description of 79-Standard Forth. Of historical interest.

316 - BIBLIOGRAPHY OF FORTH REFERENCES
2nd edition, Sept. 1984 ..... $15/16/18 _____
An excellent source of references to articles about Forth throughout microcomputer literature. Over 1300 references.

MISCELLANEOUS

NEW FIG T-SHIRT!

601 - T-SHIRT SIZE _____
Small, Medium, Large and Extra-Large. White design on a dark blue shirt.
$10/11/12 _____

602 - POSTER (BYTE Cover) ..... $15/16/18 _____

616 - HANDY REFERENCE CARD ..... FREE _____

683 - FORTH-83 HANDY REFERENCE CARD ..... FREE _____

FALL SPECIAL

FORTH Dimensions

BACK VOLUMES 1-6

containing the six issues of each volume year (May - April) from 1979/80 through 1985/85.

$50/59/90 _____

Available until November 29, 1985

PUBLICATIONS SURVEY

If you would like to suggest any other publication for review by the FIG Publications committee for inclusion in the Forth Interest Group Order Form, please complete the information below and return to FIG.

Title: _____

Author: _____

Publisher: _____

Comments: _____

_____

_____

Your comments on any of the publications we currently carry are most welcome, please complete information below.

Title: _____

Comments: _____

_____

_____

FORTH INTEREST GROUP

P.O. BOX 8231 SAN JOSE, CALIFORNIA 95155 408/277-0668

Name _____

Company _____

Address _____

City _____

State/Prov. _____ **ZIP** _____

Country _____

Phone _____

OFFICE USE ONLY

By _____ Date _____ Type _____

Shipped By _____ Date _____

UPS Wt. _____ Amt. _____

USPS Wt. _____ Amt. _____

BO Date _____ Wt. _____ Amt. _____

By _____

ITEM #	TITLE	AUTHOR	QTY	UNIT PRICE	TOTAL
107	MEMBERSHIP				SEE BELOW

☐ Check enclosed (payable to: **FORTH INTEREST GROUP**)

☐ VISA ☐ MASTERCARD

Card # _____

Expiration Date _____

Signature _____

SUBTOTAL	
10% MEMBER DISCOUNT MEMBER # _____	
CA. RESIDENTS SALES TAX	
HANDLING FEE	**$2.00**
MEMBERSHIP FEE $20/27/33 ☐ NEW ☐ RENEWAL	
TOTAL	

PAYMENT MUST ACCOMPANY ALL ORDERS

MAIL ORDERS
Send to:
Forth Interest Group
P.O. Box 8231
San Jose, CA 95155

PHONE ORDERS
Call 408/277-0668 to place credit card orders or for customer service. Hours: Monday-Friday, 9am-5pm PST.

PRICES
All orders must be prepaid. Prices are subject to change without notice. Credit card orders will be sent and billed at current prices. $15 minimum on charge orders. Checks must be in US$, drawn on a US Bank. A $10 charge will be added for returned checks.

POSTAGE & HANDLING
Prices include shipping. A $2.00 handling fee is required with all orders.

SHIPPING TIME
Books in stock are shipped within five days of receipt of the order. Please allow 4-6 weeks for out-of-stock books (delivery in most cases will be much sooner).

SALES TAX
California deliveries add 6%. San Francisco Bay Area add 7%.

(Continued from page 19)

and restart a real-time clock. **NOW** simply places the frozen time value on the stack. **INITNOW** resets the jiffy timer.

Screen 52 also contains the nonstandard word **ARRAY**. Other familiar but nonstandard words are **D/S** and **U*/** (*Forth Dimensions* V/1). All other words are in Forth-79. Screen 52 finishes by setting up two arrays to store the number of uses and accumulated time for up to twenty words, and defining the variable **WORD#** used to count the number of words being timed. All these are initialized to zero by **INIT**.

Screens 53-54 define a display of the measured timings. We will come back to these screens later.

Screen 55 does all the work. The object is to modify the definition of each new word compiled, as shown in Figure Two. Since **NEXTNO** is an **IMMEDIATE** word, it executes at the compile time of **WORDn** with the result that the current value of **WORD#** is compiled by **LITERAL** as an in-line literal. During later execution of **WORDn** this value is placed on the stack as input to

```
VARIABLE T

: WORD   NOW NEGATE T +!

  <...original body of word...>

  NOW T +!  ;
```

Figure One

```
: WORDn [NEXTNO]

  LITERAL MARKSTART >R

  <...original body of word...>

  R> MARKEND  ;
```

Figure Two

Word	#Uses	Avg. Msecs
T2*	1	5416.000
TWO*	1000	1.516
DUP+	1000	0.733
2*	1000	0.650
TASK	0	0.000

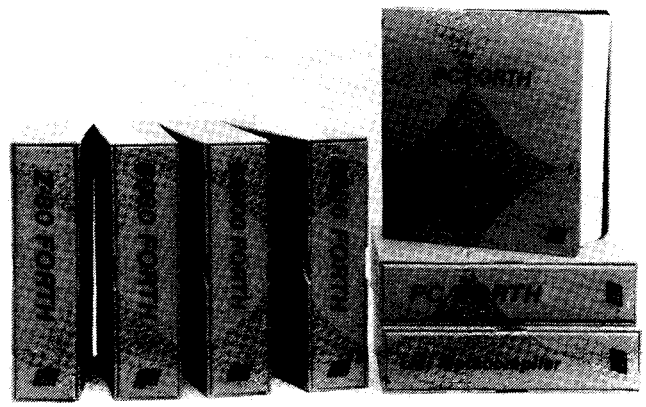
Summary

Figure Three

```
SCR # 55
 0  ( WORD TIMER MACROS-4)
 1
 2  : INC  ( ADDR ---  )  1 SWAP +! ;
 3  : NEXTNO  ( ---  N )  WORD# DUP @ SWAP INC ;  IMMEDIATE
 4
 5  : MARKSTART  ( N ---N)  FREEZE DUP DUP #USES INC
 6    NOW NEGATE SWAP TCUM +! UNFREEZE ;
 7
 8  : MARKEND  ( N ---)  FREEZE NOW SWAP TCUM +! UNFREEZE ;
 9
10  : :: [COMPILE] : ; IMMEDIATE
11
12  : : [COMPILE] : [COMPILE] NEXTNO [COMPILE] LITERAL
13      COMPILE MARKSTART COMPILE >R ;              IMMEDIATE
14
15  :: ; COMPILE R> COMPILE MARKEND [COMPILE] ; ; IMMEDIATE


SCR # 56
 0  ( TIMER MACRO TESTING )
 1
 2  : TASK ;
 3  : 2* 2* ;
 4  : DUP+ DUP + ;
 5  : TWO* 2 * ;
 6
 7  : T2* 234 1000 0 DO
 8         DUP 2* DROP
 9         DUP DUP+ DROP
10         DUP TWO* DROP
11    LOOP DROP ;
12
13
14  T2* SUMMARY
15
```

TOTAL CONTROL
with *LMI FORTH*™



For Programming Professionals:

an expanding family of compatible, high-performance, Forth-83 Standard compilers for microcomputers

For Development:
Interactive Forth-83 Interpreter/Compilers
- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 400 page manual written in plain English
- Options include software floating point, arithmetic coprocessor support, symbolic debugger, native code compilers, and graphics support

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, 8086, 68000, and 6502
- No license fee or royalty for compiled applications

Support Services for registered users:
- Technical Assistance Hotline
- Periodic newsletters and low-cost updates
- Bulletin Board System

Call or write for detailed product information and prices. Consulting and Educational Services available by special arrangement.

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone credit card orders to: ***(213) 306-7412***

Overseas Distributors.
Germany: Forth-Systeme Angelika Flesch, D-7820 Titisee-Neustadt
UK: System Science Ltd., London EC1A 9JX
France: Micro-Sigma S.A.R.L., 75008 Paris
Japan: Southern Pacific Ltd., Yokohama 220
Australia: Wave-onic Associates, 6107 Wilson, W.A.

MARKSTART, which increments the appropriate element of **#USES** and subtracts the current time from the total time accumulated for **WORDn**. The word number is pushed onto the return stack and we are ready for execution of the main body of **WORDn**. Afterwards, the word number is recovered from the return stack and **MARKEND** adds the current time to the accumulated time.

All that remains is to put together these timer words into macros that replace : and ;. Before redefining : the macro :: is defined. This is just a synonym for : and is needed so that the definition of ; will not include all the : timer words.

Creating the macros is simple. Just follow the rules developed in *Forth Dimensions* V/5: precede each **IMMEDIATE** word by **[COMPILE]**, precede all other words by **COMPILE** and declare the macro **IMMEDIATE**.

After loading these screens, load and run the application you want to time, and then execute **SUMMARY** to produce a display of all the colon-defined words in your application, with the number of uses and average time per use for each word. Screens 53–54 contain the definitions of **SUMMARY** and supporting words. Everything here is straightforward except **CHECK**, perhaps, which scans the dictionary linkage, skipping over constants, variables and other word types until a colon-defined word is found. **CHECK** returns the NFA of this word.

Screen 56 contains an example application of these macros. Three words that multiply by two are defined. The first, **2***, makes use of a built-in **CODE** definition. The other two are self-explanatory. The word **T2*** exercises each of the "multiply-by-two" words 1000 times on the same problem. **SUMMARY** prints the results, as shown in Figure Three.

When using the timer macros, you may want to define additional application words. These will be automatically assigned word numbers as they are compiled, and will be added to the summary. However, if you go back and redefine one of your words, it is necessary to execute **INIT** and then recompile your entire application.

Now You Can Add

New!

ARTIFICIAL INTELLIGENCE

To Your Programs Using a Powerful Combination

"SUPERFORTH 64" +AI

By Elliot Schneider & Jack Park

Heres Your Chance to Profit by being on the Forefront, Write 5th Generation Software

Learn How To:

- Create Intelligent Programs
- Build Expert Systems
- Write Stand Alone License Free Programs
- Construct Rule Bases
- Do Knowledge Engineering
- Use Inference Engines

Write Intelligent Programs For:

- Home Use
- Robotics
- Medical Diagnosis
- Education
- Intelligent CAI
- Scientific Analysis
- Data Acquisition
- Data Analysis
- Business
- Real Time Process Control
- Fast Games
- Graphics
- Financial Decisions

Extended Math Functions

- Fast ML Floating Point & Integer Math
- Double Precision 2E+38 with Auto. Sci Not.
- $n^x e^x$ Logx Loge Sin Cos Tan SQR 1/X...
- Matrix and Multidimensional Lattice Math
- Algebraic Expression Evaluator

Easy Graphics & Sound Words

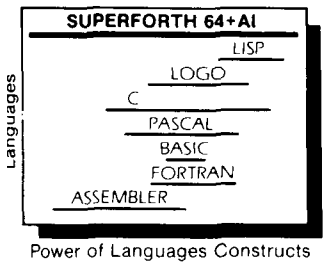
- Hires Plotting
- Windows
- Split Screen
- Printer/Plotter Ctrl
- Sprite & Animation Editor
- Turtle Graphics
- Koala Pad Graphics Integrator
- Hires Circle, Line, Arc
- Music Editor
- Sound Control

Easy Control of all I/O...

- RS232 Functions
- Access all C-64 Peripherals
- Interrupt Routines

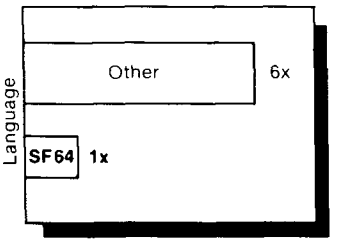
Utilities

- Interactive Interpreter
- Forth Virtual Memory
- Full Cursor Screen Editor
- Full String Handling
- Trace & Decompiler
- Conditional Macro Assembler
- Interactive Compiler
- Romable Code Generator
- 40K User Memory
- All Commodore File Types
- Conversational User Defined Commands



Power of Languages Constructs

SuperForth 64 is more powerful than most other computer languages



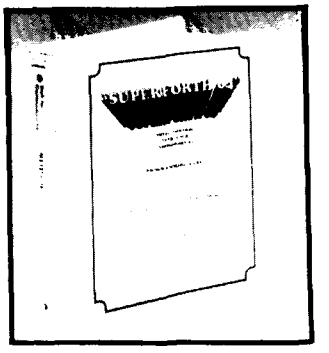
Programming Time

SuperForth 64 Saves You Time and Money

Great Documentation

- Easy to Read 350 pg. Manual with Tutorials
- Source Screen Provided
- Meets all MVP Forth-79 Industrial Standards
- Personal User Support

A Total Integrated Package for the Commodore 64



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

® Parsec Research
Commodore 64 TM of Commodore

SPECIAL INTRODUCTORY OFFER

only **$99.00**

$203.00 Value
Limited Time Offer

Call:
(415) 961-4103
MOUNTAIN VIEW PRESS INC
P.O. Box 4656
Mt. View, CA 94040

Dealer for
PARSEC RESEARCH
Drawer 1776, Fremont, CA 94538

Improved Forth-83 DO LOOP

Dennis L. Feucht
Beaverton, Oregon

When the Forth-83 Standard was released, it was accompanied by a well-written, public-domain model which could run on CP/M systems. Another model was provided by Laxen, Perry, Tracy, et al. for the Apple II and other 650X-based computers. Both implementations of Forth handle **DO LOOP** constructs in the same way. The Forth-83 definition for **LEAVE** differs from that of Forth-79 or fig-FORTH, necessitating a different approach to **DO LOOP** implementation.

Structured vs. Unstructured Constructs

In pre-83 Forths, **LEAVE** would cause a **DO LOOP** to be exited when the program flow reached **LOOP**. In Forth-83, the loop is left immediately. Although the Forth-83 **LEAVE** is more versatile than the previous one, it also has the effect of unstructuring the **DO LOOP** construct. Other Forth control constructs, such as **IF ELSE THEN** or **BEGIN UNTIL**, are *structured* in that they have one entry and one exit. No branches into or out of a control construct are otherwise allowed. When compiling structured constructs, all forward referencing can be handled on the stack, since multiple constructs within a word will be nested rather than overlapping. For example, when compiling, a **BEGIN** will invoke **HERE**, leaving the address of the next word to be compiled. Later, **UNTIL** compiles a **?BRANCH** followed by the address left on the stack by **BEGIN**; it is where **?BRANCH** will (conditionally) branch to. If another **BEGIN** is encountered before **UNTIL**, it too will leave the address of its forward reference on the stack. Since the forward-reference address on the top of the stack goes with the second **BEGIN**, the first **UNTIL** encountered must terminate the second **BEGIN UNTIL** construct, causing it to be nested inside the first one. Similar reasoning applies to the other Forth control constructs — except **LEAVE**.

Since multiple **LEAVE**s can occur within a **DO LOOP** along with other constructs, and since **DO LOOP**s can also be nested, the simple structured scheme

```
VARIABLE LOOP-LINK

: DO-EXIT TRUE LOOP-LINK @ HERE LOOP-LINK ! 0 , ;

: DO COMPILE (DO) DO-EXIT ; IMMEDIATE

: ?DO COMPILE (?DO) DO-EXIT ; IMMEDIATE
```

Figure One

LEAVE now compiles (LEAVE) and links in the following address:

```
: LEAVE COMPILE (LEAVE) HERE LOOP-LINK @ ,
        LOOP-LINK !
; IMMEDIATE
```

Figure Two

```
: LOOP-EXIT SWAP ?CONDITION LOOP-LINK @
      BEGIN DUP @ >R HERE 2+ OVER ! R@ ?DUP
         IF NIP THEN R> 0=
      UNTIL 2+ , LOOP-LINK !
;
: LOOP COMPILE (LOOP) LOOP-EXIT ; IMMEDIATE
: +LOOP COMPILE (+LOOP) LOOP-EXIT ; IMMEDIATE
```

Figure Three

```
: (DO)  R> 2+ -ROT SWAP DUP >R - >R >R ;
: (?DO) 2DUP =
      IF 2DROP R> @ >R
      ELSE R> 2+ -ROT SWAP DUP >R - >R >R
      THEN
;
```

Figure Four

```
VARIABLE LOOP-LINK-T


: DO-EXIT LOOP-LINK-T @ HERE LOOP-LINK-T ! 0 , ;
: LOOP-EXIT LOOP-LINK-T @
      BEGIN DUP @ >R HERE THERE 2+ OVER ! R@ ?DUP
         IF NIP THEN R> 0=
      UNTIL THERE 2+ , LOOP-LINK-T !
;
```

Figure Five

```
T: LOOP COMPILE [TARGET] (LOOP) LOOP-EXIT T;

T: +LOOP COMPILE [TARGET] (+LOOP) LOOP-EXIT T;

T:  LEAVE  COMPILE  [TARGET]  (LEAVE)  HERE  LOOP-LINK-T  @  ,
        LOOP-LINK-T !
T;

T: DO COMPILE [TARGET] (DO) DO-EXIT T;

T: ?DO COMPILE [TARGET] (?DO) DO-EXIT T;
```

Figure Six

no longer applies. For example, consider this word:

```
: WORD  DO ... BEGIN ... LEAVE
... UNTIL ... LOOP  ;
```

To implement **LEAVE**, have it compile **(LEAVE)** which, at run time, pops the **DO LOOP** items off the return stack and branches to the forward address just after **LOOP**. This address is contained in the memory location following **(LEAVE)**. An obvious (but unsuccessful) way to then resolve the forward reference created after **(LEAVE)** is to put the address after **(LEAVE)** on the stack so that **LOOP** can put the address after **(LOOP)** into the address after **(LEAVE)** (that is, resolve the forward reference). But **UNTIL** would use the address left by **LEAVE** to resolve the forward reference of **BEGIN** instead! This control flow is unstructured due to **LEAVE**.

DO LOOP Implementations

To solve this compile-time problem, the Forth-83 implementations have **(DO)** push an extra item on the return stack, which it gets from the location following **(DO)** in the threaded code. This item is the address used by **(LEAVE)**. Thus, the forward reference for **(LEAVE)** is compiled by **LOOP**, which puts it in the address following **(DO)**. Any **LEAVE**s within the **DO LOOP** compile **(LEAVE)**, but no forward reference following **(LEAVE)**. It is at *run time* that **(LEAVE)** gets its forward reference from the return stack and branches to it. At compile time, **LEAVE**s do not leave forward-reference addresses on the stack, and the **DO LOOP** remains structured.

The implementation of **DO LOOP** given here avoids having to place the forward reference for **(LEAVE)** on the return stack, resulting in a two-item use of this stack, as did pre-83 Forths. The tradeoff is a somewhat more complex compile-time activity. This implementation should be easy to adapt to the current **DO LOOP** constructs, since the run-time words **(LOOP)**, **(+LOOP)**, **(LEAVE)**, **I** and **J** (which are **CODE** words and, thus, machine dependent) require very little modification. The loop index word **I** remains unchanged. For **J**, reduce the indexing into the stack by 2 (since the **LEAVE** address is no longer there). For **(LOOP)** and **(+LOOP)**, the change is simple. At the end of each of these words, the items used by the **DO LOOP** are pulled off the return stack. For 8080 implementations, the **6 D LXI** in **LOOP-EXIT** should be changed to **4 D LXI**. For the 650X, **LOOP-EXIT** contains six **PLA**s. Remove two of them. Similarly, for **(LEAVE)**, two fewer return stack pulls are needed.

The problem of unstructured forward references left on the parameter stack by **LEAVE** is eliminated by building a linked list of these references. When **LOOP** (or **+LOOP**) is encountered, it follows the pointers of this list backward, resolving references. The first reference is always created by **DO** or **?DO**. **?DO** requires a forward reference, but **DO** does not. However, to keep **LOOP** simple, a spurious reference is compiled for **(DO)** also, though it never uses it. To handle nested **DO LOOP**s, multiple lists of forward references are created. The variable **LOOP-LINK** is used to point to the head of the current list. When **DO** is encountered, **LOOP-LINK** is pushed on the parameter stack and is initialized to **HERE**, the address following **(DO)**. **LOOP** uses the pointer in **LOOP-LINK** to resolve addresses, then pops the stack into **LOOP-LINK**. **LOOP-LINK** then points to the last unresolved forward reference location of the next **DO LOOP** out. **DO** and **?DO** compile a zero after **(DO)** (a zero terminates the linked list). Their definitions are shown in Figure One.

LEAVE now compiles **(LEAVE)** and links in the address shown in Figure Two. As described, **LOOP** or **+LOOP** compiles its run-time word, resolves the list of forward references (with the **BEGIN UNTIL** loop in **LOOP-EXIT**), then restores **LOOP-LINK** from the stack (see Figure Three). **DO** and **?DO** place a *true* flag on the stack for error checking, which is absorbed by **?CONDITIONAL**.

Finally, the run-time words for **DO** and **?DO** are shown in Figure Four. Notice that the **ELSE** part of **(?DO)** is just **(DO)**. It might be tempting to substitute **(DO)** instead, but the top of the return stack would then contain the return pointer into the threaded code in **(?DO)**. Since the arguments used by **(DO)** are on the *return* stack, it would not be properly set up.

Metacompiler Considerations

To metacompile this **DO LOOP** construct, transition words similar to **DO**, **?DO**, **LEAVE**, **LOOP** and **+LOOP** are needed. Error checking has been omitted from the implementation given, and the variable **LOOP-LINK-T** has been added. The words given in Figure Five go into the vocabulary of the metacompiler. The word **THERE** takes a compile-time, target-code address and offsets it to the corresponding run-time address. The following words go into the **TRANSITION** vocabulary, which contains defining and compiling words found in the target source code. The **T:** and **T;** words indicate this (see Figure Six). All **T:** definitions are made immediate by **T;** and **[TARGET]** does a **[COMPILE]** from the **TARGET** vocabulary, which acts as the symbol table for target words.

A Forth I/O Technique

Pseudo-Interrupts

Ed Schmauch
Ponca City, Oklahoma

Traditionally, there are two methods of computer input/output (I/O): polling and interrupts. Polling is easier to implement but has the disadvantage of completely tying up the CPU during I/O. The use of interrupts allows the CPU to do background processing concurrently with the I/O but requires a more detailed understanding of the hardware. Using Forth, I have developed an intermediate technique which requires no more understanding of the hardware than polling, but allows background processing like interrupts. I have dubbed the technique "pseudo-interrupts."

The technique takes advantage of Forth's threaded nature. The inner interpreter routine **NEXT** is patched to execute a code word with every execution of **NEXT**. Because Forth code is threaded, execution is constantly passing through **NEXT**. The code word that is patched in will therefore execute with great frequency. This pseudo-interrupt service routine can poll I/O port registers a single time and take any necessary action. This gives a form of polling which is neatly interleaved with Forth execution. This technique does not match the rapid response and CPU efficiency of hardware interrupts, but offers an improvement over traditional polling techniques by allowing high-speed I/O with background processing occurring simultaneously.

Screens 21-23 show the basic pseudo-interrupt words. I am using the Forth-79 Standard (MVP-FORTH version 1.01.03) on a Kaypro II portable computer. **PATCH** (screen 21) expects the parameter field address (PFA) of the pseudo-interrupt service routine on the top of the stack and patches a jump to this location over the three one-byte instructions at the beginning of **NEXT**. For example, if the pseudo-interrupt service routine is named **PISR** one should enter **' PISR PATCH** to enable the pseudo-interrupt. Make sure the pseudo-interrupt service routine preserves the values in any registers which are used by the Forth inner interpreter. With MVP-FORTH on a Z-80 machine, I must preserve the value in the BC register pair, since this is the Forth instruction pointer. The pseudo-interrupt service routine must end with the macro **BACK** (screen 22), which assembles the three patched-over, one-byte instructions and a jump to location **NEXT+3**. **UNPATCH** (screen 21) returns **NEXT** to its normal form, effectively disabling the pseudo-interrupt. (*Note that this technique applies only to those Forth systems which use a call to a central* **NEXT**, not to those which compile that code with each word.)

In my CP/M version of Forth, **KEY-I/O** (screen 23) can be necessary, since **<KEY>** uses CP/M to get a character from the console. If no character is present, CP/M waits until one is input. During this time, **NEXT** is not being executed, rendering the pseudo-interrupt inactive. **KEY-I/O** waits in Forth until a character is present and then executes **<KEY>**, ensuring that execution is constantly passing through **NEXT**. **KEY** is vectored through the variable **'KEY**, allowing easy definition of **KEY-PATCH** and **KEY-UNPATCH** to vector execution to **KEY-I/O** and back to **<KEY>**, respectively.

Both to demonstrate the use of pseudo-interrupts and to evaluate their performance, I have written code to run the prime number sieve benchmark (*BYTE*, January 1983) concurrently with pseudo-interrupt-driven serial output. This code is shown in screens 24-27. The message "I AM RUNNING THE PRIME NUMBER BENCHMARK," followed by a carriage return and line feed, is placed in an output buffer by screen 24. The pseudo-interrupt service routine (screen 25) continuously outputs this message. When the last character of the message (the line feed) is transmitted, the buffer pointer is reset to zero so the message will be restarted from the beginning. **PISR** also counts the number of characters output, so character-output efficiency can be evaluated.

Screen 26 has the code to initialize and set the baud rate on the serial port. This code and parts of **PISR** are hardware dependent. Be sure to modify them to suit your hardware before you attempt to implement them. **INIT_SERIAL** only needs to be executed

Baud Rate	Time (Seconds)	Percent Full-Speed	Characters Output	Output Efficiency (%)
No output	132.9	100.0	-	-
1,200	183.5	72.4	21,792	99.0
2,400	185.0	71.8	43,852	98.8
4,800	187.5	70.9	88,860	98.7
9,600	192.6	69.0	182,632	98.8
19,200	204.5	65.0	400,252	101.9

Summary of Benchmark Performance
Table One

once after the computer is booted. **BAUD_SET** is used whenever you want to change the baud rate. Screen 27 contains the *BYTE* benchmark. Line 4 zeroes the buffer pointer, zeroes the character counter and enables the pseudo-interrupt. Line 15 disables the pseudo-interrupt and prints the number of characters transmitted. The rest of the code is from *BYTE*, except that the sieve is executed ten times to add to the precision of the timings. Besides enabling and disabling the pseudo-interrupt, the benchmark code makes no reference to the serial output process. As with true interrupt-driven I/O, pseudo-interrupts operate without requiring the attention of the main line code. Note that **KEY-PATCH** was not necessary, since the benchmark never gets input from the keyboard. To run the benchmark without pseudo-interrupts, "comment out" lines 4 and 15, except the semi-colon.

Table One shows pseudo-interrupt performance for serial output baud rates ranging from 1,200 to 19,200. Also shown is the time for the benchmark with no concurrent pseudo-interrupt-driven output. The benchmark was run twice at each baud rate to lessen the possibility of an error. The serial output was observed on a Lear Siegler ADM 5 terminal. The percent of full speed is calculated by dividing the time for the benchmark without pseudo-interrupt-driven output by the time for a particular run and multiplying by 100. (One might expect to divide the time for a run by the time without pseudo-interrupt-driven output, but we are interested in speed, which is the reciprocal of time.) The character output efficiency is the number of characters output divided by the maximum number of characters that could be output at the given baud rate in the time required for the benchmark to run, multiplied by 100. As can be seen from the table, the benchmark runs at approximately two-thirds of full speed, with the speed gradually decreasing with increasing baud rate. Worded another way, the pseudo-interrupt-driven serial output slows the benchmark by only one third. The character output efficiency for baud

```
SCR #21
  0 ( PSEUDO-INTERRUPT -   PATCH, UNPATCH    - EHS 29AUG83 FORTH-79 )
  1 ( CRC=10792 )
  2 BASE @ HEX
  3
  4 CODE PATCH    ( PFA --- )
  5   C3 ( JMP ) A MVI      NEXT STA      H POP      L A MOV
  6   NEXT 1+ STA      H A MOV      NEXT 2+ STA      NEXT JMP
  7 END-CODE
  8
  9 CODE UNPATCH    ( --- )
 10   NEXT C@ A MVI     NEXT STA     NEXT 1+ C@ A MVI     NEXT 1+ STA
 11   NEXT 2+ C@ A MVI     NEXT 2+ STA     NEXT JMP
 12 END-CODE
 13
 14 BASE !
 15 -->


SCR #22
  0 ( PSEUDO-INTERRUPT -      MACRO BACK      - EHS 29AUG83 FORTH-79 )
  1 ( CRC= 7942 )
  2 ( BACK ASSEMBLES THE THREE ONE-BYTE INSTRUCTIONS THAT ARE )
  3 ( PATCHED OVER IN NEXT AND A JUMP TO NEXT+3 )
  4
  5 BASE @ CONTEXT @ CURRENT @ DECIMAL ASSEMBLER DEFINITIONS
  6
  7 : BACK    ( --- )
  8   [ NEXT     C@ ] LITERAL C,
  9   [ NEXT 1+ C@ ] LITERAL C,
 10   [ NEXT 2+ C@ ] LITERAL C,
 11   NEXT 3 + JMP ;
 12
 13 CURRENT ! CONTEXT ! BASE !
 14
 15 -->


SCR #23
  0 ( PSEUDO-INTERRUPT - KEY-I/O, VECTORING - EHS 29AUG83 FORTH-79 )
  1 ( CRC=40007 )
  2 ( DON'T WAIT FOR CHAR IN CP/M WHERE NEXT WON'T BE EXECUTING )
  3
  4 : KEY-I/O    ( --- )
  5   BEGIN
  6     ?TERMINAL
  7   UNTIL
  8   <KEY> ;
  9
 10 : KEY-PATCH    ( --- )
 11   ' KEY-I/O CFA 'KEY ! ;
 12
 13 : KEY-UNPATCH    ( --- )
 14   ' <KEY> CFA 'KEY ! ;
 15 -->

SCR #24
  0 ( PSEUDO-INTERRUPT -     BUFFER, ETC.      - EHS 26MAY84 FORTH-79 )
  1 ( CRC=55411 )
  2
  3 : MAKE_BUFFER    ( --- )
  4   CREATE    34 WORD    C@ 1+ ALLOT    13 C, 10 C, ;
  5
  6 MAKE_BUFFER PISR_BUFFER I AM RUNNING THE PRIME NUMBER BENCHMARK"
  7               ( COUNT BYTE WILL BE USED AS BUFFER POINTER )
  8
  9  0 CONSTANT  BAUD
 10  4 CONSTANT  DATA
 11  6 CONSTANT  CSR
 12  4 CONSTANT  READY?
 13 10 CONSTANT  LF
 14    DVARIABLE CHARS
 15 -->
```

rates 1,000 to 9,600 is around ninety-nine percent. I am puzzled by the character output efficiency over 100 percent for 19,200 baud. If the efficiency is really 100 percent, the time would have to be in error by almost four seconds. I have confidence that the precision of my timings was greater than this. The only other possibility I can think of is that when the Kaypro serial port is set for 19,200 baud, it actually outputs characters slightly faster than that. I admit this explanation is not very satisfying and invite readers to offer other suggestions.

In conclusion, the pseudo-interrupt-driven serial output slows the benchmark by only one third while delivering an output efficiency of ninety-nine percent. This demonstrates that the technique is a viable alternative to traditional I/O methods.

Two final notes will help those planning to implement this in MVP-FORTH. I had to increase the memory available to Forth with the following:

```
LIMIT 16384 + ' LIMIT ! CHANGE
```

Also, my version of MVP-FORTH did not contain --> in the kernel. It is easily defined as follows:

```
: --> ?LOADING 0 >IN ! 1 BLK +! ;
IMMEDIATE
```

```
SCR #25
  0 ( PSEUDO-INTERRUPT - SERVICE ROUTINE  - EHS 26MAY84 FORTH-79 )
  1 ( CRC=34702 )
  2
  3 CODE PISR ( --- )
  4   CSR IN    READY? ANI   0# IF   ( SERIAL PORT READY? )
  5     PISR_BUFFER H LXI   M E MOV   E INR   E M MOV   ( INC PTR )
  6     0 D MVI   D DAD   M A MOV   DATA OUT   ( OUTPUT CHAR )
  7     LF CPI   0= IF   ( END OF LINE? )
  8       A SUB   PISR_BUFFER STA   THEN   ( ZERO BUFFER POINTER )
  9     CHARS 2+ LHLD   H INX   CHARS 2+ SHLD   ( INCREMENT CHARS )
 10     L A MOV   H ORA   0= IF   ( CARRY TO HIGH WORD? )
 11       CHARS LHLD   H INX   CHARS SHLD   THEN
 12   THEN
 13   BACK
 14 END-CODE
 15 -->


SCR #26
  0 ( PSEUDO-INTERRUPT -   SERIAL PORT    - EHS 05SEP83 FORTH-79 )
  1 ( CRC=29294 )
  2 BASE @ HEX
  3
  4 CREATE INIT_ARRAY  ( OUTPUT TO CSR TO INITIALIZE Z80 SIO )
  5   18 C, 04 C, 44 C, 01 C, 00 C, 03 C, C1 C, 05 C, EA C,
  6 HERE INIT_ARRAY - CONSTANT INIT_ARRAY_LENGTH
  7
  8 : INIT_SERIAL   ( --- )
  9   INIT_ARRAY_LENGTH 0 DO
 10     INIT_ARRAY I + C@   CSR P!                    (  N   BAUD )
 11   LOOP ;                                          (  7   1200 )
 12                                                   ( 10   2400 )
 13 : BAUD_SET     ( N --- )   BAUD P! ;              ( 12   4800 )
 14 BASE !                                           ( 14   9600 )
 15 -->                                              ( 15  19200 )


SCR #27
  0 ( PSEUDO-INTERRUPT -   BYTE BENCHMARK   - EHS 26MAY84 FORTH-79 )
  1 ( CRC=42100 )  ( TO RUN WITHOUT PSEUDO-INT, COMMENT OUT 4 & 15 )
  2 8190 CONSTANT SIZE   VARIABLE FLAGS SIZE 2- ALLOT
  3 : DO-PRIME   ( --- )
  4   0 PISR_BUFFER C!   0 0 CHARS D!   ' PISR PATCH
  5   10 0 DO
  6     FLAGS SIZE 1 FILL ( SET ARRAY )   0 ( ZERO PRIMES COUNT )
  7     SIZE 0 DO   FLAGS I + C@
  8       IF I DUP + 3 + DUP I +
  9          BEGIN DUP SIZE <
 10          WHILE 0 OVER FLAGS + C! OVER + REPEAT
 11          DROP DROP 1+
 12       THEN
 13     LOOP   CR . ." PRIMES "
 14   LOOP
 15   UNPATCH   4 SPACES CHARS D@ D. ." CHARS OUTPUT " ;
```

Attend the Seventh Annual

FORML CONFERENCE

The original technical conference for professional Forth programmers, managers, vendors, and users.

November 29 – December 1, 1985

Asilomar Conference Center

Monterey Peninsula overlooking the Pacific Ocean
Pacific Grove, California

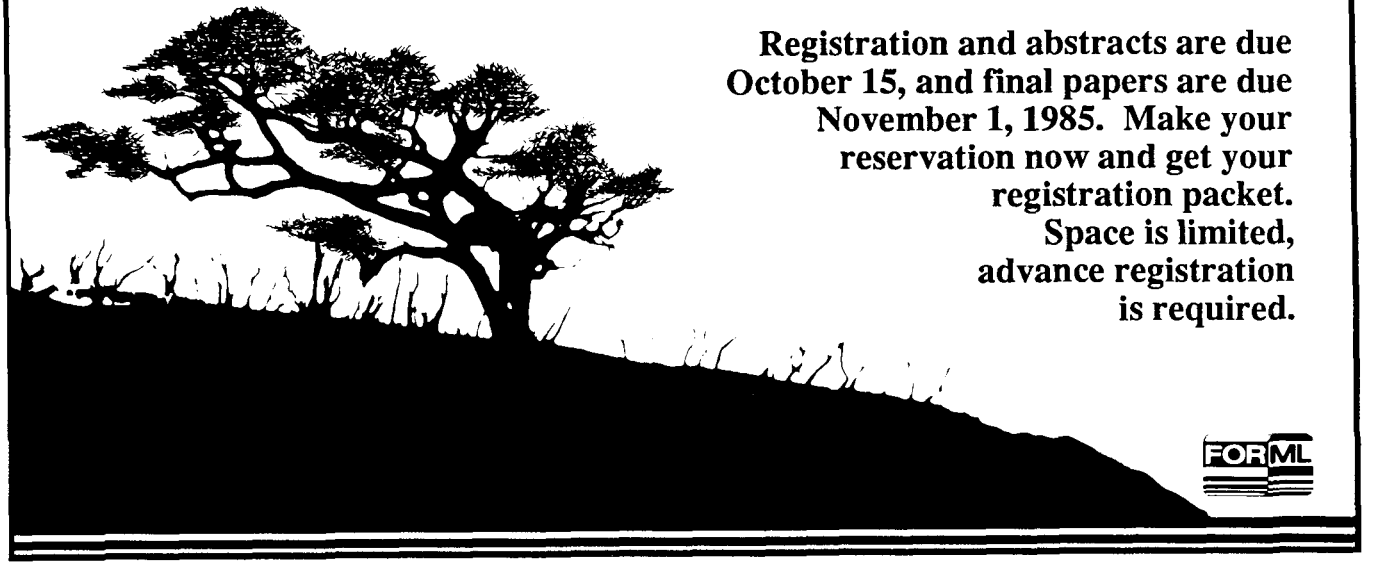
This year's theme: SOFTWARE TOOLS, a Forth natural.

Present your favorite Forth tool and publish it in the conference proceedings, or present a paper on another Forth topic. You will meet other Forth professionals and learn about the latest in Forth applications, techniques, and directions.

To get your registration packet call the FIG Hot Line (408) 277-0668 or write to: FORML Registration, Forth Interest Group, P. O. Box 8231, San Jose, CA 95155.

Registration:	$265	Double Room
	$315	Single Room (Limited availability)
	$200	Non-conference guest (Share a double room)

Registration includes room, meals, conference materials, and social events.



Registration and abstracts are due October 15, and final papers are due November 1, 1985. Make your reservation now and get your registration packet. Space is limited, advance registration is required.

FORML

An Approach to Reading Programs

Kim Harris
Palo Alto, California
Michael Ham
Santa Cruz, California

Programmers frequently need to understand in detail a program that was written by someone else. Understanding other people's programs is essential in program maintenance and revision, and it also plays a valuable role in programmer education by allowing programmers to learn from the techniques and skills embodied in their colleagues' creations, as well as from their mistakes and oversights.

Formal "code inspections" efficiently facilitate the difficult task of truly comprehending a program. This technique has the added advantage that the use of an inspection *team* means several people learn the program at one time. The team approach also ensures that the program is inspected from various viewpoints; thus, inefficiencies and errors are more likely to be detected.

Code inspection provides an excellent way for members of a FIG chapter to work together through the examples of Forth code published in *Forth Dimensions* and other journals. This technique helps the chapter members learn from the code and from each other.

A code inspection team consists of four to six members; more or fewer hinder the process. Larger FIG chapters can form several inspection teams. In a company environment, it is important that supervisory or management personnel *not* be on the team, since their presence has a chilling effect on the frankness of the criticism and on the open give-and-take of the process.

Specific roles are assigned in advance to individual team members:

Author One team member takes the role of author of the code. Ideally, of course, the actual author will play this role, but in the absence of the true author, a stand-in will suffice. The purpose of the author is to be the expert who answers questions about the code: why a given approach was taken, what considerations led to the development of a particular word. The author's ultimate responsibility is to make the appropriate changes to the code.

Moderator The moderator's role is to keep things moving and to keep the inspection session on track. The idea of the code inspection is to understand the code and to point out ways in which it falls short: errors, inefficiencies and the like. This meeting, though, is *not* the place to write new code or to develop alternate solutions. The moderator is a facilitator who keeps the meeting focused on its purpose. Also, every suggestion and correction must be recorded during the meeting so that it will not be forgotten. The moderator is responsible for the minutes of the meeting.

Reader One team member reads the code aloud, paraphrasing it in terms of its purpose, rather than merely echoing the actual definitions. Instead of, "Next is SUM-COUNTS. ONE RECORD-COUNT STORE BEGIN FETCH-RECORD...," the reader says something like, "The next word begins with the first record; for each record, it reads the total and increments the count box." By paraphrasing the code, the reader focuses the group's attention on the *meaning* of the code; by paraphrasing it aloud, the group stays together and actually examines each word. Without such a reading, code inspections quickly degenerate into, "Next is screen 10; any comments? No? Screen 11? Screen 12?" The process goes faster, but the code is not really examined or understood.

Inspectors The remaining team members are "inspectors." They follow the code as the reader reads it, they ask questions of the "author" and point out what they question or don't understand. Some inspectors may be asked to pay special attention to program functions in their area of expertise (e.g., drivers, data-base design, quality assurance, integration, testing).

All members of the team check the code for style: Are the names well chosen? Are stack diagrams present and accurate? Are comments present and helpful?

To prepare for a code inspection, each person should spend two hours carefully reviewing the code, to get familiar with it and to get some grasp of how it does what it does. The reader, of course, may have to prepare in greater detail, since to paraphrase the code requires a good understanding of its intentions. Preparation is vitally important to the success of the code inspection.

The meeting should be limited to two hours; people cannot focus with any intensity for a longer period of time. Experience shows that a line of Forth code requires (on the average) about thirty seconds in this kind of review; this means a screen can be covered in about seven minutes. The result is that a review of Forth code cannot normally cover more than about seventeen screens in any session; twenty is probably the maximum.

After the code inspection, the author revises the code to take into account the comments and criticisms offered by the inspection team. Compared to uninspected code, the code resulting from this process of review and revision shows a number of improvements: The style and factoring are better and more understandable, which makes subsequent maintenance and revision much easier. Reviewing the code with other programmers catches problems that would normally show up only later, when the various separate modules would be integrated — problems like the use of the same name for different functions, redundancy with other modules (as when a programmer redevelops a routine that another programmer has already written and tested), inconsistency in data values, and mismatches of context or side effects. Experience has shown that formal code inspections increase productivity, reduce development time and cost, and contribute to programmer training and education.

The literature on code "walk throughs" and team programming is extensive. A good starting place for the interested reader is The Psychology of Computer Programming *by Gerald Weinberg.*

Volume VI Index

This reference guide to Volume VI was prepared as a service to our readers and to all members of the Forth Interest Group. Items are referenced by issue number and page number.

79-Standard
  An Augmented TRACE 5/18
  Automatic Capitalization in Forth 1/20
  Forth-83 Program to Run Forth-79 1/20
  Forth List Handling 1/36
  Forth Semaphores 4/23
  In-Word Parameter Words 6/9
  Mathquiz 6/13
  Re-Defining a Colon Word 3/20
  Think Like a User, Write Like a Fox 3/23
  Upgrading Programs to Forth-83 3/26
83-Standard
  Enhanced DO LOOP 6/18
  Forth-83 Program to Run Forth-79 Code 4/28
  Forth Control Structures 2/20
  Long Divisors and Short Fractions 3/10
  Pollard's Monte Carlo Factorizer 6/25
  Quicksort and Swords 5/25
  A Simple Data Transfer Protocol 2/32
  Simple Modem I/O Words 5/13
  Upgrading Forth-79 Programs to Forth-83 3/26

ANDIF and ANDWHILE 4/33
Anonymous Variables 1/33
Anway, A. 1/22
Apple IIe, Screens for 1/22
Applications
  Mathquiz 6/13
  Tutorial: Simple Modem I/O Words 5/13
Ask the Doctor
  Astronomical Problems 3/30
  Forth and the AIM-65 2/37
  How to Learn Forth 5/9
  Learning Forth 6/7
  Moving to ROM 1/10
An Augmented TRACE 5/18
Automatic Capitalization in Forth 1/20

Baden, W. 5/25
Berkey, R. 3/26, 4/28

Capitalization 1/20
CASE Statement, "YACS" 6/38
Chen, S. S. 6/9
China Tour 1984, FORML 5/38, 6/33
CODE, Mixing With High-Level Forth 4/37
Colon Word 3/20
Control Structures 2/20

Data Transfer 2/32
Debugging
  An Augmented TRACE 5/18
  Techniques 2/38
DO LOOP, Enhanced 6/18

Enhanced DO LOOP 6/18
Ericson, K. 2/32

The Far Right Stuff 6/21
Feucht, D. 2/32
fig-FORTH
  fig-FORTH Interpreters 1/12
  Forth P-Code Interpreter 4/9
  Local Definitions 6/16
  More Screens for the Apple 1/22
  Quicksort and Swords 5/25
  TI 99/4A Screen Dump 6/11
Forchheimer, R. 5/32
FORML
  1984 Asilomar Conference 5/34
  China Tour 1984 5/38, 6/33
Forth-79 Standard *see* 79-Standard
Forth-83 Program to Run Forth-79 Code 4/28
Forth-83 Standard *see* 83-Standard
Forth Control Structures 2/20
Forth in Rehabilitation Applications 2/28
Forth List Handling 1/36
Forth P-Code Interpreter 4/9
Forth Semaphores 4/23

Gates, W. C. 1/24, 4/33
Goppold, A. 5/18
Grossman, N. 3/10, 6/25

Hall, J. D. 1/39, 3/35, 4/40, 5/41, 6/40
Ham, M. 3/23, 4/19
Harralson, D. W. 2/20
High-Level Packet Communication 5/32
Hore, M. 6/18

In-Word Parameter Words 6/9
Interactive Editing 1/24
Interpreters
  fig-FORTH 1/12
  Forth P-Code 4/9

Jaffe, D. L. 2/28
James, J. S. 5/13

Laxen, H. 2/38, 3/32, 4/37, 6/38
List Handling 1/36
Local Definitions 6/16
Long Divisors and Short Fractions 3/10
Lotspiech, J. B. 1/20
Luoto, K. W. 1/26, 2/10

Mathematics
  Long Divisors and Short Fractions 3/10
  Pollard's Monte Carlo Factorizer 6/25
Mathquiz 6/13
Monroe, A. J. 4/9
More Screens for the Apple 1/20
Morgenstern, L. 1/33
Morton, L. D. 6/13

Olofsson, B. 1/36

P-Code Interpreter 4/9
Parameters, In-Word 6/9
Parnas' it...ti Structures 1/26
Pascal 4/9
Perkel, M. 2/18
Perry, M. 6/21
Pollard's Monte Carlo Factorizer 6/25
Procedural Arguments 2/10
Pruitt, C. 6/16

Quicksort and Swords 5/25

Ragsdale, W. F. 1/8, 1/10, 2/37, 3/30, 5/9, 6/7
Recursion 4/19
Re-Defining a Colon Word 3/20
Reiling, R. R. 2/9, 3/9
Rogers, H. H. 6/11
Ruehl, T. M. 1/20

Schmauch, E. 3/20
Screen Dump, TI 99/4A 6/11
Semaphores 4/23
SOFTNET: High-Level Packet Communication 5/32
Sorts, Quicksort and Swords 5/25
Standards, Forth
  see also 79-Standard, 83-Standard, fig-FORTH
  Forth-83 Program to Run Forth 79 Code 4/28
  Upgrading Forth-79 Programs to Forth-83

Tan, L. 6/9
Techniques Tutorials
  Debugging Techniques 2/38, 3/32
  Mixing CODE With High-Level Forth 4/37
  "YACS" 6/38
Telecommunications
  Simple Modem I/O Words 5/13
  SOFTNET: High-Level Packet Communication 5/32
Tevet, A. 5/30
Texas Instruments *see* TI 99/4A
Think Like a User, Write Like a Fox 3/23
TI 99/4A Screen Dump 6/11
Ting, C. H. 1/12
TRACE, Augmented 5/9

Upgrading Forth-79 Programs to Forth-83 3/26

Why Forth Isn't Slow 5/30

Zander, J. 4/23, 5/32

THE FORTH SOURCE™

MVP-FORTH

Stable - Transportable - Public Domain - Tools

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

MVP Books - A Series

- ☐ **Vol. 1,** *All about FORTH* by Haydon. MVP-FORTH glossary with cross references to fig-FORTH, *Starting FORTH* and FORTH-79 Standard. 2nd Ed. $25
- ☐ **Vol. 2,** *MVP-FORTH Assembly Source Code*. Includes IBM-PC®, CP/M® and APPLE® listing for kernel $20
- ☐ NEW **Vol. 3,** *Floating Point* with source code by Koopman $25
- ☐ **Vol. 4,** *Expert System* with source code by Park $15
- ☐ **Vol. 5,** *File Management System* with interrupt security by Moreton $25
- ☐ **Vol. 6,** *Expert Tutorial for Volume 4* by M & L Derick $15
- ☐ NEW **Vol. 7,** *FORTH GUIDE to MVP-FORTH* by Haydon $20

MVP-FORTH Software - A transportable FORTH

- ☐ **MVP-FORTH Programmer's Kit** including disk, documentation. Volumes 1, 2 & 7 of MVP Series, and Starting FORTH. ☐CP/M, ☐CP/M 86, ☐Z100, ☐Apple, ☐STM PC, ☐IBM PC, XT/AT & compatibles, ☐PC/MS-DOS, ☐Osborne, ☐Kaypro, ☐MicroDecisions, ☐DEC Rainbow, ☐NEC 8201, ☐TRS-80/100, NEW ☐HP150, ☐HP110, ☐Macintosh, ☐Atari 600/800/1200, ☐ADAM $175
  - ☐ **MVP-FORTH Enhancement Package** for IBM-PC/XT/AT Programmer's Kit. Includes full screen editor, MS-DOS file interface, disk, display and assembler operators. $110
  - ☐ **MVP-FORTH Floating Point and Matrix Math** for IBM PC/XT/AT with 8087 or Apple with Applesoft $85
  - ☐ **MVP-FORTH Graphics Extension** for IBM PC/XT/AT or Apple $65
  - ☐ **MVP-FORTH Programming Aids** for CP/M, IBM or APPLE Programmer's Kit. Extremely useful tool for decompiling, callfinding, translating, and debugging. $200
  - ☐ **MVP-FORTH Cross Compiler** for CP/M Programmer's Kit. Generates headerless code for ROM or target CPU $300
  - ☐ **MVP-FORTH Meta Compiler** for CP/M Programmer's kit. Use for applications on CP/M based computer. Includes public domain source. $150
- ☐ **MVP-FORTH PADS (Professional Application Development System)** for IBM PC/XT/AT or PCjr or Apple II, IIB or IIe. An integrated system for customizing your FORTH programs and applications. The editor includes a bi-directional string search and is a word processor specially designed for fast development. PADS has almost triple the compile speed of most FORTH's and provides fast debugging techniques. Minimum size target systems are easy with or without heads. Virtual overlays can be compiled in object code. PADS is a true professional development system. Specify Computer. $500
  - ☐ **MVP-FORTH MS-DOS** file interface for IBM PC PADS $80
  - ☐ **MVP-FORTH Floating Point & Matrix Math** see above $85
  - ☐ **MVP-FORTH Graphics Extension** see above $65
- ☐ **MVP-FORTH EXPERT-2 System** for learning and developing knowledge based programs. Both IF-THEN procedures and analytical subroutines are available. Source code is provided. Specify ☐Apple, ☐IBM, or ☐CP/M. Includes MVP Books, Vol. 4 & 6. $100
- ☐ NEW **Word/Rule** A Word Processor for the IBM PC/XT/AT with 256K. MVP-FORTH compatible kernel with Files. Edit and Print systems. Includes Disk and Calculator systems and ability to compile additional FORTH words. $150

FORTH DISKS

- ☐ **APPLE** by MM $125
- ☐ **APPLE** by MM.F. & G. $245
- ☐ **ATARI**® valFORTH $60
- ☐ **ATARI** by PNS. F.G. & X. $90
- ☐ **C64** by HES Commodore 64 cartridge $40
- NEW ☐ **C64** with EXPERT-2 by PS MVP.G.F. & X. $99
- ☐ **CP/M** by MM $125
- ☐ **CP/M** by MM. F. $185
- ☐ **HP-75** by Cassady $150
- ☐ **HP-85** by Lange $90
- ☐ **IBM-PC** by LM $100
- NEW ☐ **IBM-PC** by MM $125
- NEW ☐ **MacIntosh** by MM $125
- ☐ **Timex** by HW. cassette
  - ☐ T/S 1000/ZX-81 $25
  - ☐ 2068 $30
- ☐ **Z80** by LM $100
- ☐ **8086/88** by LM $100
- ☐ **68000** by LM $250
- ☐ **VIC FORTH** by HES. VIC20 Cartridge $20
- ☐ **Extensions** for LM. Specify IBM Z80 or 8086
  - ☐ Software Floating Point $100
  - ☐ 8087 Support (IBM-PC or 8086) $100
  - ☐ 9511 Support (Z80 or 8086) $100
  - ☐ Color Graphics (Z80 or 8086) $100
  - ☐ Data Base Management $200

Key to Vendors:

HW	Hawg Wild Software
LM	Laboratory Microsystems
MM	MicroMotion
PNS	Pink Noise Studio
PS	Par Sec

Codes:

F - Floating Point
G - Graphics
T - Tutorial
X - Other Extras

FORTH MANUALS, GUIDES & DOCUMENTS

- NEW ☐ **Thinking FORTH** by Leo Brodie, author of best selling "Starting FORTH" $16
- ☐ **ALL ABOUT FORTH** by Haydon, MVP Glossary $25
- ☐ **FORTH Encyclopedia** by Derick & Baker $25
- NEW ☐ **FYS FORTH** from the Netherlands
  - ☐ User Manual $25
  - ☐ Source Listing $25
- NEW ☐ **FORTH Tools and Applic.** by Feierbach $19
- ☐ **The Complete FORTH** by Winfield $16
- NEW ☐ **Learning FORTH** by Armstrong $17
- ☐ **Understanding FORTH** by Reymann $3
- ☐ **FORTH, An Applications Approach** by Toppen $20
- ☐ **FORTH Applications** by Roberts $13
- NEW ☐ **Mastering FORTH** by Anderson & Tracy $18
- ☐ **Beginning FORTH** by Chirlian $17
- ☐ **FORTH Encycl. Pocket Guide** $7
- ☐ **And So FORTH** by Huang A college level text $25
- ☐ **FORTH Programming** by Scanlon $17
- ☐ **STARTING FORTH** by Brodie. Best instructional manual available (soft cover) $20
- ☐ **68000** fig-Forth with assembler $25
- ☐ **FORML Proceedings**
  - ☐ 1980 ☐ 1981 Vol 1
  - ☐ 1981 Vol 2 ☐ 1982
  - NEW ☐ 1983 ☐ 1984 each $25
- ☐ **1981 Rochester Proceedings**
  - ☐ 1981 ☐ 1982 ☐ 1983
  - ☐ 1984 each $25
- ☐ **Bibliography of FORTH** $17
- ☐ **The Journal of FORTH Application & Research**
  - ☐ Vol 1/1 ☐ Vol 1/2
  - ☐ Vol 2/1 ☐ Vol 2/2
  - NEW ☐ Vol 2/3 each $15
- ☐ **METAFORTH** by Cassady $30
- ☐ **Threaded Interpretive Languages** $25
- ☐ **Systems Guide to fig-FORTH** by Ting $25
- NEW ☐ **Inside F83** Manual by Ting $25
- ☐ **FORTH Notebook** by Ting $25
- ☐ **Invitation to FORTH** $20
- ☐ **PDP-11 User Man.** $20
- ☐ **6502 User's Manual** by Rockwell Intl. $10
- ☐ **FORTH-83 Standard** $15
- ☐ **FORTH-79 Standard** $15

- ☐ **Installation Manual for fig-FORTH** $15
- ☐ **Source Listings of fig-FORTH.** Specify CPU or Computer $15

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

MOUNTAIN VIEW PRESS, INC.

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

Number Editing Utility

Ken Takara
San Jose, California

When writing interactive applications, one often needs numeric data to be entered by a user at the keyboard. Unfortunately, most built-in input routines do not provide the kind of data checking and display formatting usually wanted.

In BASIC, for example, you might code

```
10  INPUT X
```

The user then types "HI, I'M KEN" and the program blows up. In Forth, you would say something like

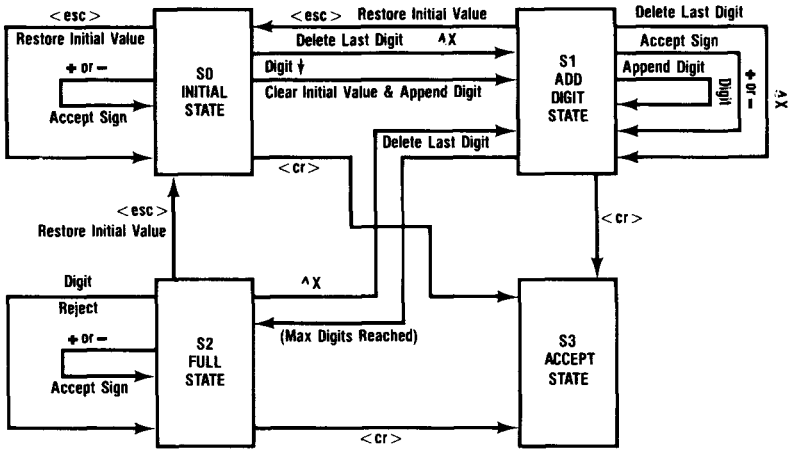
```
PAD 80 EXPECT PAD NUMBER
```

which has the sole advantage of not crashing when the user enters some other, arbitrary sequence of key strokes.

In this little article, I describe a simple data-entry utility written just to handle certain numeric entry problems. Naturally, it is written in Fortran. I call the utility **NUMED**.

How To Use It

You call **NUMED**, passing it the row and column position on the display at which it should appear, and a double-length initial value. **NUMED** displays the initial value in inverse video. You can then edit it, replace it or accept it as it stands.



NUMED State Diagram
Figure One

```
: NUMEDIT-5.2 ( row col -- : Display format 5.2)
  CH CV                    ( Position the cursor)
  0. NUMED.EDLINE 1-       ( Point to the editing buffer)
  CONVERT DROP             ( Convert to a binary value)
  <# # # "." HOLD # # # NUMED.SIGN @ SIGN #> TYPE ;

:  NUMBER-EDITOR-5.2   (  row col dval1 -- dval2 : Editor call,
format 5.2)
  ' NUMED-5.2 CFA NUMED*DISPLAY !     ( Set display vector)
  5 NUMED.MAXDIGITS !                 ( Set the digit count)
  NUMBER-EDITOR ;                     ( Call the editor)
```

Create a display word for the 5.2 format: Now create a word to set the display vector, set the digit count and call the number editor
Figure Two

```
================================ 1001 ================================
 0.  ( Data entry: INSTRING)
 1.
 2.  : INSTRING ( string char -- position or -1)
 3.    0 4 ROLL 4 ROLL          ( Duplicate string)
 4.    OVER + SWAP              ( Start/stop address in string)
 5.    DO                       ( For each entry in string...)
 6.      OVER I C@ = IF         ( Found it...)
 7.        SWAP DROP 0 LEAVE    ( Return its offset and quit flag)
 8.      ELSE                   ( Not found yet...)
 9.        1+                   ( Increment offset pointer)
10.      THEN
11.    LOOP                     ( Keep looking)
12.    IF DROP -1 THEN          ( Failure flag, return failure)
13.  ;     -->
14.
15.
================================ 1002 ================================
 0.  ( Data entry: SELECT)
 1.
 2.  : SELECT ( string -- offset : Return position in string of char)
 3.    BEGIN
 4.      OVER OVER KEY          ( Dup string; get key)
 5.      INSTRING               ( See if it's in the string)
 6.      DUP -1 =               ( Not found yet...)
 7.    WHILE
 8.      DROP                   ( Drop key and keep trying)
 9.    REPEAT
10.    >R DROP DROP R>          ( Keep only position offset)
11.  ;
12.
================================ 1003 ================================
```

Valid keys are defined as follows:

Keys 0 - 9	Numeric digit
<cr>	Accept current value
<esc>	Get initial value
X	Delete last digit
+	Change sign to positive
-	Change sign to negative

The actions of the keys are best described by referring to the state diagrams (Figure One). Note that if the first key pressed is a digit, the initial value is cleared and replaced by whatever the user enters.

Notes on the Display

The default display word is set to show a thirteen-character field: ten digits and a decimal point, a dollar sign and either a plus or a minus. Obviously, this was developed for business applications! The format is 10.2 (ten digits, with two to the right of the decimal point).

The display is vectored, so you aren't stuck with only a 10.2 dollar format. You can write your own display word based on the default word, then tell **NUMED** to use it instead. I'll talk about that in the "Configuration" section.

Implementation Dependencies

This utility was developed under MicroMotion Forth-79 on the Apple II+. Certain implementation-specific words were used, which should be mentioned.

Cursor positioning and video attributes are used by **NUMED**. Specifically, the four words **CV**, **CH**, **INVERSE** and **NORMAL** are present. Most versions of Forth have some sort of equivalent. Glossary entries are:

CV (row ---)	Put cursor on given row.
CM (col ---)	Put cursor on given column.
INVERSE (---)	Use inverse video.
NORMAL (---)	Use normal video.

```
============================== 1003 ==============================
 0. ( NUMBER EDITOR--start: Primitives)
 1.
 2. CREATE NUMED.SAVESTR     16 ALLOT  ( Initial string)
 3. CREATE NUMED.EDLINE      16 ALLOT  ( Edit string)
 4. VARIABLE NUMED.MAXDIGITS 12 NUMED.MAXDIGITS !
 5. VARIABLE NUMED.SAVESIGN  ( Initial sign)
 6. VARIABLE NUMED.SIGN      ( Current sign)
 7. VARIABLE NUMED.ADDPT     ( Add pointer)
 8. VARIABLE NUMED*DISPLAY   ( Display word vector)
 9. 46 CONSTANT "."          ( Decimal point ascii)
10. 16 STRING SIGNED.SLCT " xyz+-0123456789" SIGNED.SLCT S!
11.      27 SIGNED.SLCT DROP C!          ( <esc> = char0)
12.      24 SIGNED.SLCT DROP 1+ C!       ( ^X    = char1)
13.      13 SIGNED.SLCT DROP 2+ C!       ( <cr>  = char2)
14.      -->
15.
================================================================
============================== 1004 ==============================
 0. ( NUMBER EDITOR: NUMED-CLRBUF NUMED-RESET )
 1.
 2. : NUMED-CLRBUF ( -- : Clear editing buffers)
 3.   NUMED.EDLINE  16 32 FILL     ( Blank out edit line)
 4.   0 NUMED.ADDPT ! ;            ( Add point at begining)
 5.
 6. : NUMED-RESET ( -- : Reset to initial value)
 7.   NUMED-CLRBUF                 ( Clear edit buffer)
 8.   16 NUMED.ADDPT !             ( Pointer to end of string)
 9.   NUMED.SAVESTR NUMED.EDLINE 16 CMOVE ( Restore initial str)
10.   NUMED.SAVESIGN @ NUMED.SIGN ! ;      ( Restore initial sign)
11.
12.      -->
13.
14.
15.
================================================================
============================== 1005 ==============================
 0. ( NUMBER EDITOR: NUMED-INITIAL NUMED-ACCEPT )
 1.
 2. : NUMED-INITIAL ( dval -- : Set initial value)
 3.   NUMED.SAVESTR 16 32 FILL   ( Clear holding buffer)
 4.   NUMED-CLRBUF               ( Clear edit buffer)
 5.   2DUP 0. D< IF -1 ELSE 0 THEN NUMED.SAVESIGN !
 6.   <# #S #> DUP >R            ( dval -> string)
 7.   NUMED.SAVESTR SWAP CMOVE   ( Save initial value string)
 8.   NUMED-RESET R> NUMED.ADDPT ! ; ( Initial add point)
 9.
10. : NUMED-ACCEPT ( -- dval : Accept the edited value)
11.   0. NUMED.EDLINE 1- CONVERT DROP ( string->dval)
12.   NUMED.SIGN @ IF DNEGATE THEN ;  ( Get sign)
13.
14.      -->
15.
================================================================
============================== 1006 ==============================
 0. ( NUMBER EDITOR: ?NUMED-FULL NUMED+ )
 1.
 2. : ?NUMED-FULL ( -- flag : True if buffer is full)
 3.   NUMED.ADDPT @ NUMED.MAXDIGITS @ < NOT ;
 4.
 5. : NUMED+ ( char -- : Add digit to string)
 6.   NUMED.EDLINE          ( Edit line...)
 7.   NUMED.ADDPT @ + C!    ( Append to add point)
 8.   1 NUMED.ADDPT +! ;    ( Increment add point)
 9.
10.      -->
11.
12.
13.
14.
15.
================================================================
```

MicroMotion Forth also has a set of string-handling words. I've used **S!** and **STRING** to create a single-dimension array of characters for my own word, **SELECT**.

STRING is a defining word that makes space for a character string. For example,

```
10 STRING MUSH
```

creates a dictionary entry called **MUSH** with space for ten characters. When you type **MUSH** it leaves the address of the first character and the length of the string on the stack. You could approximate it with

```
CREATE MUSH   10 C,   10 ALLOT
```

and then, when calling **MUSH**, type

```
MUSH COUNT
```

to get the string pointer and length. The word **S!** moves a character string into the array part of the string variable. Some versions of Forth have the word ," to compile a string into the dictionary.

You will note that I have to patch some characters into the selection string. This is because it is difficult to include things like <esc>, <cr> and ^X as character literals in Forth.

SELECT

SELECT is a small utility with uses outside the **NUMED** package. Given a string of characters, **SELECT** accepts only those keystrokes whose characters are included in the string. It returns the index of the character within the string. This is permits the routine using **SELECT** to **CASE** on the keystroke, frequently done in menu-driven programs.

Implementation Notes

NUMED has been implemented as a state machine where each state consists of an action part (within the **CASE**) and a transition part. Some action words within a state require the keystroke index as input, while some do not. Digits, for example, must be passed to

```
============================== 1007 ==============================
 0. ( NUMBER EDITOR: NUMED-XDIGIT NUMED-SIGN ?NUMED-CMD )
 1.
 2. : NUMED-XDIGIT ( -- : Delete last digit)
 3.   NUMED.ADDPT @ 0> IF -1 NUMED.ADDPT +! THEN ( Delete last char)
 4.   32 NUMED.EDLINE NUMED.ADDPT @ + C! ;       ( Set to blank)
 5.
 6. : NUMED-SIGN ( sign -- : Set sign)
 7.   NUMED.SIGN ! ;
 8.
 9. : ?NUMED-CMD ( -- <commandlist> : Get a command)
10. (  0=<esc> : 1=^X : 2=<cr> : sign 3=setsign : char 4=add-digit)
11.   SIGNED.SLCT SELECT      ( Get valid keystroke)
12.   DUP 3 < IF EXIT THEN    ( Return a control key)
13.   DUP 4 > IF SIGNED.SLCT DROP + C@ 4 EXIT THEN ( Return a digit)
14.   3 = IF 0 ELSE -1 THEN 3 ;                    ( Return a sign)
15.      -->
================================================================

============================== 1008 ==============================
 0. ( NUMBER EDITOR: NUMED-1STDIGIT )
 1.
 2. : NUMED-1STDIGIT ( char -- : Add 1st digit)
 3.   NUMED-CLRBUF       ( Clear edit buffer)
 4.   NUMED+ ;           ( Add the digit)
 5.
 6. : NUMED-REJECT ( char -- : Reject; full buffer)
 7.   DROP BELL ;        ( Say 'all full')
 8.
 9. : NUMED-NOTHING ( -- : Do nothing) ;
10.
11.      -->
12.
13.
14.
15.
================================================================

============================== 1009 ==============================
 0. ( NUMBER EDITOR: NUMED-S0 )
 1. : NUMED-S0 ( -- nxtstate : State 0 control)
 2.   ?NUMED-CMD DUP >R      ( Save copy of cmd code)
 3.   CASE                   ( Legal actions)
 4.     0 OF NUMED-RESET    ENDOF
 5.     1 OF NUMED-XDIGIT   ENDOF
 6.     2 OF NUMED-NOTHING  ENDOF
 7.     3 OF NUMED-SIGN     ENDOF
 8.     4 OF NUMED-1STDIGIT ENDOF
 9.   ENDCASE
10.   R>
11.   DUP 2 = IF DROP 3 EXIT THEN       ( Accept value)
12.   ?NUMED-FULL IF DROP 2 EXIT THEN   ( Maxdigits reached)
13.   DUP 0= OVER 3 = OR IF DROP 0 EXIT THEN     ( esc/sign)
14.   DUP 1 = OVER 4 = OR IF DROP 1 EXIT THEN ; ( ^X/digit)
15.      -->
================================================================

============================== 1010 ==============================
 0. ( NUMBER EDITOR: NUMED-S1 )
 1. : NUMED-S1 ( -- nxtstate : State 1 control)
 2.   ?NUMED-CMD DUP >R
 3.   CASE
 4.     0 OF NUMED-RESET   ENDOF
 5.     1 OF NUMED-XDIGIT  ENDOF
 6.     2 OF NUMED-NOTHING ENDOF
 7.     3 OF NUMED-SIGN    ENDOF
 8.     4 OF NUMED+        ENDOF
 9.   ENDCASE
10.   R>
11.   ?NUMED-FULL IF DROP 2 EXIT THEN      ( Maxdigits reached)
12.   DUP 0= IF DROP 0 EXIT THEN           ( esc=Undo)
13.   DUP 1 = OVER 2 > OR IF DROP 1 EXIT THEN ( ^X/digit)
14.   DROP 3 ;                ( CR=Accept)
15.      -->
================================================================
```

the append-digit word, while other action words take no parameters at all. The resulting code is not as elegant as it might be, but I saw no overriding reason to change it.

Configuring NUMED

NUMED is normally configured to accept a ten-digit value, displayed in the $10.2 format mentioned earlier. This may not always be desirable, of course. You may wish to reconfigure the editor before editing a field. To do this, you need to create a new display word, then change the display vector to point to it, and change the acceptable number of digits. **NUMED-10.2** is the default display word, and **NUMED* DISPLAY** is the vector. **NUMED. MAXDIGITS** is a variable containing the maximum number of digits accepted for a value.

For example, to edit a field in the form xxx.xx with all five digits displayed, see Figure Two.

Glossary

SELECT (s-addr len --- index)

Given a string, accept a keystroke represented within the string and return its index within the string.

NUMBER-EDITOR (row col dval1 --- dval1)

Call the number editor. The field will be displayed on the video screen at *row,col* in inverse video. Digits will enter the display in calculator format. On exit, the double-value result is returned.

```
============================== 1011 ==============================
 0. ( NUMBER EDITOR: NUMED-S2 )
 1. : NUMED-S2 ( -- nxtstate : State 2 control)
 2.   ?NUMED-CMD DUP >R
 3.   CASE
 4.     0 OF NUMED-RESET    ENDOF
 5.     1 OF NUMED-XDIGIT   ENDOF
 6.     2 OF NUMED-NOTHING ENDOF
 7.     3 OF NUMED-SIGN     ENDOF
 8.     4 OF NUMED-REJECT   ENDOF
 9.   ENDCASE
10.   R>
11.   DUP 0= IF DROP 0 EXIT THEN    ( esc=Undo)
12.   DUP 1 = IF DROP 1 EXIT THEN   ( ^X=Delete)
13.   DUP 2 = IF DROP 3 EXIT THEN   ( CR=Accept)
14.   DROP 2 ;                      ( Sign/digit)
15.     -->
================================================================
============================== 1012 ==============================
 0. ( NUMBER EDITOR: NUMED-10.2 )
 1.
 2. : NUMED-10.2 ( row col -- : Display value)
 3.   CH CV
 4.   0. NUMED.EDLINE 1-
 5.   CONVERT DROP DABS
 6.   <# # # "." HOLD #S 36 HOLD NUMED.SIGN @ SIGN #>
 7.   13 OVER - SPACES TYPE ;
 8.
 9.   NUMED-10.2 CFA NUMED*DISPLAY !
10.
11.     -->
12.
13.
14.
15.
================================================================
============================== 1013 ==============================
 0. ( NUMBER EDITOR: NUMED-STATES )
 1.
 2. : NUMED-STATES ( state -- nxtstate : Full state control)
 3.   CASE
 4.     0 OF NUMED-S0 ENDOF
 5.     1 OF NUMED-S1 ENDOF
 6.     2 OF NUMED-S2 ENDOF
 7.   ENDCASE ;
 8.
 9.
10.     -->
11.
12.
13.
14.
15.
================================================================
============================== 1014 ==============================
 0. ( NUMBER EDITOR: NUMBER-EDITOR )
 1.
 2. : NUMBER-EDITOR ( row col dval1 -- dval2 : Full edit)
 3.   NUMED-INITIAL            ( Prepare to edit)
 4.   OVER OVER INVERSE NUMED*DISPLAY @ EXECUTE ( Display dval1)
 5.   0 BEGIN                  ( Initial state is S0)
 6.     NUMED-STATES           ( Execute state)
 7.     3 PICK 3 PICK NUMED*DISPLAY @ EXECUTE   ( Display result)
 8.   DUP 3 = UNTIL            ( Go until state 3 reached)
 9.   DROP                     ( Drop extra state value)
10.   NORMAL NUMED*DISPLAY @ EXECUTE ( Display final dval2)
11.   NUMED-ACCEPT ;           ( Accept the result)
12.
13.
14.
15.
================================================================
```

Chapter News

Michael Ghormley
San Jose, California

With this issue, we say goodbye to John Hall as the International FIG Chapter Coordinator. John has served in this capacity for more than two years, and has done a sterling job. For the entire FIG membership, I wish to thank you, John, for your enormous contribution of time and energy. I hope that in the future I can do as well.

We wish to welcome two new FIG chapters:

Holland FIG Chapter, Breda, Holland

FIG des Alpes, Annely, France

Atlanta FIG Chapter

Ron Skelton reports that *Rick Nixon* has developed a Forth-83 for the Commodore 64 which follows *Mastering Forth*, except for an improved editor. The price is $20, which includes ongoing support. Rick's home phone is 404-377-3509. *Brian Walsh* provided a comparison of fig-FORTH, Forth-79, Forth-83 and HP-71B Forth.

East Tennessee FIG Chapter

Norman E. Smith writes that *Richard Spille* talked on the operation of the Forth inner interpreter. *Paul Satterlee* presented a high-level floating-point package, and *Dick Tracey* demonstrated Forth on a TI 99/4A.

Boston FIG Chapter

The report from *Bob Demrow* is that their group is developing a controller project for the Plymouth-Carver Planetarium as a public-service project. It is being coordinated by *Russell Blake*. Also, *Al Grant* spoke on the Rockwell R65F11 chip set.

Richmond FIG Chapter

Donald Full reports that *John C. Lundin, Jr.* is giving a tutorial on data structures. *Donald Full* presented new words called "conditional comments" which may be passed over or compiled, depending on a value on the stack. *Phil Smith* brought a Rockwell R65F11-based, single-board computer for inspection.

Advertiser's Index

Bryte Computers	16
Dash, Find & Associates	12
Forth Interest Group	21-24
Forth Dimensions	44
FORML Conference	33
FORTH, Inc.	18
Forth Institute	4
Hartronix	10
Harvard Softworks	8
HiTech Equipment	2
Laboratory Microsystems	26
MCA	41
MicroMotion	20
Miller Microcomputer Services	15
Mountain View Press	36
Next Generation Systems	20
Parsec Research	27
Shaw Laboratories	20
Sota	17

MCA

FIG-Forth for the Compaq, IBM-PC, and compatibles. $35

Operates under DOS 2.0 or later, uses standard DOS files.

Full-screen editor uses 16 x 64 format. Editor Help screen can be called up using a single keystroke.

Source included for the editor and other utilities.

Save capability allows storing Forth with all currently defined words onto disk as a .COM file.

Definitions are provided to allow beginners to use *Starting Forth* as an introductory text.

Source code is available as an option

A Metacompiler on a host PC, produces a PROM for a target 6303/6803 Includes source for 6303 FIG-Forth. Application code can be Metacompiled with Forth to produce a target application PROM. $280

FIG-Forth in a 2764 PROM for the 6303 as produced by the above Metacompiler. Includes a 6 screen RAM-Disk for stand-alone operation. $45

An all CMOS processor board utilizing the 6303. Size: 3.93 x 6.75 inches. Uses 11-25 volts at 12ma, plus current required for options. $240 - $360

Up to 24kb memory: 2kb to 16kb RAM, 8k PROM contains Forth. Battery backup of RAM with off board battery.

Serial port and up to 40 pins of parallel I/O.

Processor buss available at optional header to allow expanded capability via user provided interface board.

Micro Computer Applications Ltd

8 Newfield Lane
Newtown, CT 06470
203-426-6164

Foreign orders add $5 shipping and handling. Connecticut residents add sales tax.

FIG Chapters

• ALABAMA

Huntsville FIG Chapter
Call Tom Konantz
205/881-6483

• ALASKA

Kodiak Area Chapter
Call Horace Simmons
907/486-5049

• ARIZONA

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

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

• ARKANSAS

Central Arkansas Chapter
Twice Monthly: 2nd Sat., 2 p.m. &
4th Wed., 7 p.m.
Call Gary Smith
501/227-7817

• CALIFORNIA

Los Angeles Chapter
Monthly, 4th Sat., 10 a.m.
Hawthorne Public Library
12700 S. Grevillea Ave.
Call Phillip Wasson
213/649-1428

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

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

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

Sacramento Chapter
Monthly, 4th Wed., 7 p.m.
1798-59th St., Rm. A
Call Tom Ghormley
916/444-7775

Bay Area Chapter
Silicon Valley Chapter
Monthly, 4th Sat.,
FORML 10 a.m., FIG 1 p.m.
ABC Christian School Aud.
Dartmouth & San Carlos Ave.
San Carlos
Call John Hall 415/532-1115
or call the FIG Hotline:
408/277-0668

Stockton Chapter
Call Doug Dillon
209/931-2448

• COLORADO

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

• CONNECTICUT

Central Connecticut Chapter
Call Charles Krajewski
203/344-9996

• FLORIDA

Orlando Chapter
Every two weeks, Wed., 8 p.m.
Call Herman B. Gibson
305/855-4790

Southeast Florida Chapter
Monthly, Thurs., p.m.
Coconut Grove area
Call John Forsberg
305/252-0108

Tampa Bay Chapter
Monthly, 1st Wed., p.m.
Call Terry McNay
813/725-1245

• GEORGIA

Atlanta Chapter
Call Ron Skelton
404/393-8764

• ILLINOIS

Cache Forth Chapter
Call Clyde W. Phillips, Jr.
Oak Park
312/386-3147

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

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

Rockwell Chicago Chapter
Call Gerard Kusiolek
312/885-8092

• INDIANA

Central Indiana Chapter
Monthly, 3rd Sat., 10 a.m.
Call John Oglesby
317/353-3929

Fort Wayne Chapter
Monthly, 2nd Wed., 7 p.m.
Indiana/Purdue Univ. Campus
Rm. B71, Neff Hall
Call Blair MacDermid
219/749-2042

• IOWA

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

Central Iowa FIG Chapter
Call Rodrick A. Eldridge
515/294-5659

Fairfield FIG Chapter
Monthly, 4th day, 8:15 p.m.
Call Gurdy Leete
515/472-7077

• KANSAS

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

• LOUISIANA

New Orleans Chapter
Call Darryl C. Olivier
504/899-8922

• MASSACHUSETTS

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

• MICHIGAN

Detroit Chapter
Monthly, 4th Wed.
Call Tom Chrapkiewicz
313/562-8506

• MINNESOTA

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

• MISSOURI

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

St. Louis Chapter
Monthly, 1st Tues., 7 p.m.
Thornhill Branch Library
Contact Robert Washam
91 Weis Dr.
Ellisville, MO 63011

• NEVADA

Southern Nevada Chapter
Call Gerald Hasty
702/452-3368

• NEW HAMPSHIRE

New Hampshire Chapter
Monthly, 1st Mon., 6 p.m.
Armtec Industries
Shepard Dr., Grenier Field
Manchester
Call M. Peschke
603/774-7762

• NEW MEXICO

Albuquerque Chapter
Monthly, 1st Thurs., 7:30 p.m.
Physics & Astronomy Bldg.
Univ. of New Mexico
Call Rick Granfield
505/296-8651

• NEW YORK

FIG, New York
Monthly, 2nd Wed., 8 p.m.
Queens College
Call Ron Martinez
212/517-9429

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

Rockland County Chapter
Call Elizabeth Gormley
Pearl River
914/735-8967

Syracuse Chapter
Monthly, 3rd Wed., 7 p.m.
Call Henry J. Fay
315/446-4600

• OHIO

Athens Chapter
Call Isreal Urieli
614/594-3731

Cleveland Chapter
Call Gary Bergstrom
216/247-2492

Cincinatti Chapter
Call Douglas Bennett
513/831-0142

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

• OKLAHOMA

Central Oklahoma Chapter
Monthly, 3rd Wed., 7:30 p.m.
Health Tech. Bldg., OSU Tech.
Call Larry Somers
2410 N.W. 49th
Oklahoma City, OK 73112

• OREGON

Greater Oregon Chapter
Monthly, 2nd Sat., 1 p.m.
Tektronix Industrial Park
Bldg. 50, Beaverton
Call Tom Almy
503/692-2811

• PENNSYLVANIA

Philadelphia Chapter
Monthly, 4th Sat., 10 a.m.
Drexel University, Stratton Hall
Call Melonie Hoag
215/895-2628

• TENNESSEE

East Tennessee Chapter
Monthly, 2nd Tue., 7:30 p.m.
Sci. Appl. Int'l. Corp., 8th Fl.
800 Oak Ridge Turnpike, Oak Ridge
Call Richard Secrist
615/693-7380

• TEXAS

Austin Chapter
Contact Matt Lawrence
P.O. Box 180409
Austin, TX 78718

Dallas/Ft. Worth Metroplex Chapter
Monthly, 4th Thurs., 7 p.m.
Call Chuck Durrett
214/245-1064

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

Permian Basin Chapter
Call Carl Bryson
Odessa
915/337-8994

• UTAH

North Orem FIG Chapter
Contact Ron Tanner
748 N. 1340 W.
Orem, UT 84057

• VERMONT

Vermont Chapter
Monthly, 3rd Mon., 7:30 p.m.
Vergennes Union High School
Rm. 210, Monkton Rd.
Vergennes, VT
Call Don VanSyckel
802/388-6698

• VIRGINIA

First Forth of Hampton Roads
Call William Edmonds
804/898-4099

Potomac Chapter
Monthly, 2nd Tues., 7 p.m.
Lee Center
Lee Highway at Lexington St.
Arlington, VA
Call Joel Shprentz
703/860-9260

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

• WISCONSIN

Lake Superior FIG Chapter
Call Allen Anway
715/394-8360

MAD Apple Chapter
Contact Bill Horzon
129 S. Yellowstone
Madison, WI 53705

FOREIGN

• AUSTRALIA

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

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

• BELGIUM

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

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

• CANADA

Nova Scotia Chapter
Contact Howard Harawitz
227 Ridge Valley Rd.
Halifax, Nova Scotia B3P2E5
902/477-3665

Southern Ontario Chapter
Quarterly, 1st Sat., 2 p.m.
General Sciences Bldg.
Rm. 312
McMaster University
Contact Dr. N. Solntseff
Unit for Computer Science
McMaster University
Hamilton, Ontario L8S4K1
416/525-9140 ext. 3443

Toronto FIG Chapter
Contact John Clark Smith
P.O. Box 230, Station H
Toronto, ON M4C5J2

• COLOMBIA

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

• ENGLAND

Forth Interest Group — U.K.
Monthly, 1st Thurs.,
7p.m., Rm. 408
Polytechnic of South Bank
Borough Rd., London
D.J. Neale
58 Woodland Way
Morden, Surry SM4 4DS

• FRANCE

French Language Chapter
Contact Jean-Daniel Dodin
77 Rue du Cagire
31100 Toulouse
(16-61) 44.03.06

• GERMANY

Hamburg FIG Chapter
Monthly, 4th Sat., 1500h
Contact Horst-Gunter Lynsche
Common Interface Alpha
Schanzenstrasse 27
2000 Hamburg 6

• HOLLAND

Holland Chapter
Contact: Adriaan van Roosmalen
Heusden Houtsestraat 134
4817 We Breda
31 76 713104

FIG des Alpes Chapter
Contact: Georges Seibel
19 Rue des Hirondelles
74000 Annely
50 57 0280

• IRELAND

Irish Chapter
Contact Hugh Doggs
Newton School
Waterford
051/75757 or 051/74124

• ITALY

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

• JAPAN

Japan Chapter
Contact Toshio Inoue
Dept. of Mineral Dev. Eng.
University of Tokyo
7-3-1 Hongo, Bunkyo 113
812-2111 ext. 7073

• REPUBLIC OF CHINA

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

• SWITZERLAND

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

SPECIAL GROUPS

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

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

FIGGRAPH
Call Howard Pearlmutter
408/425-8700

FORTH INTEREST GROUP
FALL Special

FORTH Dimensions

BACK VOLUMES 1 - 6

$50.00
USA AND CANADA

$59.00 FOREIGN SURFACE MAIL
$90.00 FOREIGN AIR MAIL

Containing the six issues of each volume year
(May-April) from 1979/80 through 1984/85.

AVAILABLE UNTIL NOVEMBER 29, 1985

FORTH INTEREST GROUP
P. O. Box 8231
San Jose, CA 95155

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

Address Correction Requested