FORTH
Dimensions

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

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

---

**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
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

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 LOOPS. 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 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 Parameter 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 _______________________________________

FORTH Dimensions
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 consensus. 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 t will still transmit sixteen bits. This way, sequences such as

   DO ... 1 @ ... 2 +LOOP

will still work.

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

5. New operators @ and # 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:


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:
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 toolkit 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

“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, it does not work as described in Mr. Laxen’s column. I replaced line four with the following code segment:

```
BEGIN >MARK ICSP 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 IF and 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.

(Letters continued from page 5)
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 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.
- 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 transcendentials.
- 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.

HARVARD SOFTWORKS
P.O. BOX 69
SPRINGBORO, OH 45066
(513) 748-0390
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 O 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 O 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 2.1, 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 reformatted 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
- 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
- 4.4K 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
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 \texttt{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 \texttt{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 \texttt{.S} and \texttt{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

```
: ENDM [COMPIL] 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 \texttt{SYNONYM} presented here provides a single simple way for defining synonyms. The usage of the word is

\texttt{SYNONYM <new-name> <old-name>}

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

A word defined by \texttt{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 \texttt{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 \texttt{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, \texttt{A} and \texttt{B}, have actions \texttt{X} and \texttt{Y}. Then, using an intermediate name \texttt{C}, one can define

\texttt{SYNONYM <new-name> <old-name>}

\texttt{SYNONYM C A}

\texttt{SYNONYM A B}

\texttt{SYNONYM B C}

Now \texttt{A} has the function \texttt{Y}, and \texttt{B} has the function \texttt{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.
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 ZDUP, 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. 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, linguistic 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 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.

References
Macros

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 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 semicolon. MACRO is the opposite of END-MACRO. When executed from the console, 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. Forth 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
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 \texttt{-ROT}, the compilation address of \texttt{-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 \texttt{-ROT}, the compilation address of \texttt{-ROT} 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 \texttt{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 for \texttt{-ROT}, the compilation address of \texttt{-ROT} is not compiled into the program. Instead, the compilation addresses of \texttt{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

\begin{verbatim}
IF ... ELSE ... THEN and
DO ... LOOP
\end{verbatim}

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

\begin{verbatim}
:TRIAL IF DROP THEN ;
:TRIAL IF DROP THEN ;
\end{verbatim}

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 \texttt{DNEGATE, D+, OR} and \texttt{0=}, while \texttt{D-}, \texttt{D0=} and \texttt{D=} are defined as colon definitions. These latter could be defined instead as macros:

\begin{verbatim}
MACRO D- DNEGATE D+
MACRO D0= OR 0= END-MACRO
MACRO D= D- D0= END-MACRO
\end{verbatim}

When \texttt{D=} is used in a program, it would compile not the compilation address of \texttt{D=}, but the compilation addresses of

\begin{verbatim}
DNEGATE D+ OR 0=
\end{verbatim}

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-
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

\[
\text{MACRO } 0_\# = \text{ END-MACRO}
\]

and with an equivalent colon definition

\[
: \text{0}_\# = \text{0 } = ;
\]

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

\[
= \text{ NOT} < \text{ NOT} \quad \text{AND} \quad \text{NOT} \quad \text{SWAP} \quad \text{DROP} \quad \text{ROT} \quad \text{ROT} \quad \text{OVER} \quad \text{OVER}
\]

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. A recording 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 allot's 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.
Nothing is compiled at the end of the macro definition that would correspond to the \texttt{: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 \texttt{DOES >} part of \texttt{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 \texttt{HERE} and executes it. To this end, the address of the run-time code for colon \texttt{=} is obtained by \texttt{CR} and set in place at \texttt{HERE}. Then the compiled macro words are moved into the following locations. Finally, the address of \texttt{s} is set in place at the end. Then this temporary colon definition is executed.

These definitions of \texttt{MACRO} and \texttt{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 \texttt{MACRO} and \texttt{END-MACRO} should be patterned after an array definition with \texttt{CREATE} using \texttt{]} and \texttt{]} 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 \texttt{MACRO} and \texttt{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 FORTH model\textsuperscript{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 \texttt{1 ALLOT} on screen 51 to \texttt{2 ALLOT} and add \texttt{1}+ after the \texttt{DOES >} on screen 52. (It is an interesting aspect of the Forth-83 Standard that \texttt{SYNONYM} could be written entirely within the standard, but \texttt{MACRO} and \texttt{END-MACRO} could not.)

\textbf{References}

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

(Created on page 25)
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 market-place.

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.

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 5½" disk in various formats under CP/M 2.2 or greater and 5½" MS-DOS
FULLY WARRANTED, DOCUMENTED AND SUPPORTED

NEW HP-150 & HP-110 VERSIONS AVAILABLE

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

Industrial Strength FORTH

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

NGS

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

Write it once!
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
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, metacomilation, 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, vectorized 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: 

Volume VII, No. 3
FORTH Dimensions
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*l (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:

```
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

<table>
<thead>
<tr>
<th>Word</th>
<th>#Uses</th>
<th>Avg. Msecs</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2*</td>
<td>1</td>
<td>5416.000</td>
</tr>
<tr>
<td>TWO*</td>
<td>1000</td>
<td>1.516</td>
</tr>
<tr>
<td>DUP+</td>
<td>1000</td>
<td>0.733</td>
</tr>
<tr>
<td>2*</td>
<td>1000</td>
<td>0.650</td>
</tr>
<tr>
<td>TASK</td>
<td>0</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Summary

Figure Three

```
SCR # 55
Ø ( 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
13 COMPILER MARKSTART COMPILER >R ; IMMEDIATE
14 15 : :: COMPILER R> COMPILER MARKEND [COMPILE] ; ; IMMEDIATE
```

SCR # 56
Ø ( TIMER MACRO TESTING )
1 2 : TASK ;
3 2* Z+ 1
4 : DUP+ DUP + ;
5 : (WD)* Z* ;
6 7 : T2* 234 1000 Ø DO
8 2* DROP
9 DUF+ DROP
10 DUP TWO* DROP
11 LOOP DROP ;
12 13 14 T2* SUMMARY
15
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, 2-80, 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.

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 support-
ing words. Everything here is straight-
forward 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 ap-
lication 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 applica-
tion 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

ARTIFICIAL INTELLIGENCE

To Your Programs Using a Powerful Combination

“SUPERFORTH 64”

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

Write Intelligent Programs For:
- Home Use
- Robotics
- Medical Diagnosis
- Education
- Intelligent CAI
- Scientific Analysis
- Data Acquisition

Easy Graphics & Sound Words
- Hires Plotting
- Windows
- Split Screen
- Printer/Plotter Ctrl
- Sprite & Animation Editor

Easy Control of all I/O...
- RS232 Functions
- Access all C-64 Peripherals

Utilities
- Interactive Interpreter
- Forth Virtual Memory
- Full Cursor Screen Editor
- Full String Handling
- Trace & Decompiler
- Conditional Macro Assembler

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

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% 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 LEAVES can occur within a DO LOOP along with other constructs, and since DO LOOPS 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

```
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
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 8 D LXI in LOOP-EXIT should be changed to 4 D LXI. For the 650X, LOOP-EXIT contains six PLAs. 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 LOOPs, 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 metacomplier. 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 Τ; words indicate this (see Figure Six). All T; definitions are made immediate by Τ; 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 implement but has the disadvantage of computering and interrupts.

CPU to do background processing concurrently with the I/O. The use of interrupts allows the CPU to do background processing like interrupts. 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.

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

Summary of Benchmark Performance

Table One
tains the 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 rates ranging from 1,200 to 19,200.

```forth
SCR #21
0 ( PSEUDO-INTERRUPT - PATCH, UNPATCH - EHS 29AUG83 FORTH-79 )
1 ( CRC=10792 )
2 BASE & HEX
3 CODE PATCH ( PFA ---- )
4 C3 ( JMP ) A MVI NEXT STA H POP L A MOV
5 NEXT 1+ STA H A MOV NEXT 2+ STA NEXT JMP
7 END-CODE
8
9 CODE UNPATCH ( ---- )
10 NEXT C0 A MVI NEXT STA NEXT 1+ C0 A MVI NEXT 1+ STA
11 NEXT 2+ C0 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 BASE & CONTEXT & CURRENT & DECIMAL ASSEMBLER DEFINITION
5 BACK ( ---- )
6 NEXT C3 LITERAL C,
7 NEXT 1+ C3 LITERAL C,
8 NEXT 2+ C3 LITERAL C,
9 NEXT 3 + JMP ;
10 CURRENT & CONTEXT & BASE !
11 -->

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

SCR #24
0 ( PSEUDO-INTERRUPT - BUFFER, ETC. - EHS 26MAY84 FORTH-79 )
1 ( CRC=55411 )
2 CREATE 34 WORD C0 1+ ALLOT 13 C, 10 C, ;
3 MAKE BUFFER ( ---- )
4 CREATE 34 WORD C0 1+ ALLOT 13 C, 10 C, ;
5 MAKE_BUFFER PISR BUFFER I AM RUNNING THE PRIME NUMBER BENCHMARK ( COUNT BYTE WILL BE USED AS BUFFER POINTER )
6
7 0 CONSTANT BAUD
8 4 CONSTANT DATA
9 6 CONSTANT CSR
10 4 CONSTANT READY?
11 10 CONSTANT LF
12 14 D VARIABLE CHAR
13
14 -->
```
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 CIFR_BUFFER H LXI M E MOV E INR E N MOV ( INC FTR )
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 CHAR5 2+ SHLD ( INCREMENT CHARS )
10 L A MOV H ORA 0= IF ( CARRY TO HIGH WORD? )
11 CHAR'S LHLD H INX CHAR'SHLD THEN
12 THEN
13 BACK
14 END-CODE
15 -->

SCR #26
0 ( PSEUDO-interrupt - SERIAL FORt - 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, 02 C, 01 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 + C3 CSR P' ( N BAUD )
11 LOOP ;
12 ( 10 2400 )
13 : BAUD_SET ( N --- ) BAUD P' ;
14 BASE !
15 ( 14 9600 )
16 BASE !
17 ( 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 BIAS CONSTANT SIZE VARIABLE FLAGS SIZE 2+ ALLOW
3 : DO-PRIME ( --- )
4 0 PISR_BUFFER C! 0 0 CHAR'S D! " PISR PATCH
5 10 0 DO
6 FLAGS SIZE 1 FILL ( SET ARRAY ) 0 ( ZERO PRIMES COUNT )
7 SIZE 0 DO FLAGS i C3
8 IF I DUP + 3 + DUP i +
9 BEGIN DUP SIZE <
10 WHILE 0 OVER FLAGS + C! OVER + REPEAT
11 L A DROP DROP i +
12 THEN
13 LOOP CR . " PRIMES "
14 LOOP
15 UNPATCH 4 SPACES CHAR'S D! D. " CHAR'S 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.
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 un-inspected 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.

<table>
<thead>
<tr>
<th>79-Standard</th>
<th>83-Standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>An Augmented TRACE 5/18</td>
<td>Enhanced DO LOOP 6/18</td>
</tr>
<tr>
<td>Automatic Capitalization in Forth 1/20</td>
<td>Ericson, K. 2/32</td>
</tr>
<tr>
<td>Forth-83 Program to Run Forth-79 1/20</td>
<td>The Far Right Stuff 6/21</td>
</tr>
<tr>
<td>Forth List Handling 1/36</td>
<td>Feucht, D. 2/32</td>
</tr>
<tr>
<td>Forth Semaphores 4/23</td>
<td>fig-FORTH</td>
</tr>
<tr>
<td>In-Word Parameter Words 6/9</td>
<td>fig-FORTH Interpreters 1/12</td>
</tr>
<tr>
<td>Mathquiz 6/13</td>
<td>Forth P-Code Interpreter 4/9</td>
</tr>
<tr>
<td>Re-Defining a Colon Word 3/20</td>
<td>Local Definitions 6/16</td>
</tr>
<tr>
<td>Think Like a User, Write Like a Fox 3/23</td>
<td>More Screens for the Apple 1/22</td>
</tr>
<tr>
<td>Upgrading Programs to Forth-83 3/26</td>
<td>Quicksort and Swords 5/25</td>
</tr>
<tr>
<td></td>
<td>TI 99/4A Screen Dump 6/11</td>
</tr>
<tr>
<td></td>
<td>Forchheimer, R. 5/32</td>
</tr>
<tr>
<td></td>
<td>FORML</td>
</tr>
<tr>
<td></td>
<td>1984 Asilomar Conference 5/34</td>
</tr>
<tr>
<td></td>
<td>China Tour 1984 5/38, 6/33</td>
</tr>
<tr>
<td></td>
<td>Forth-79 Standard see 79-Standard</td>
</tr>
<tr>
<td></td>
<td>Forth-83 Program to Run Forth-79 Code 4/28</td>
</tr>
<tr>
<td></td>
<td>Forth-83 Standard see 83-Standard</td>
</tr>
<tr>
<td></td>
<td>Forth Control Structures 2/20</td>
</tr>
<tr>
<td></td>
<td>Forth in Rehabilitation Applications 2/28</td>
</tr>
<tr>
<td></td>
<td>Forth List Handling 1/36</td>
</tr>
<tr>
<td></td>
<td>Forth P-Code Interpreter 4/9</td>
</tr>
<tr>
<td></td>
<td>Forth Semaphores 4/23</td>
</tr>
<tr>
<td></td>
<td>Gates, W. C. 1/24, 4/33</td>
</tr>
<tr>
<td></td>
<td>Goppold, A. 5/18</td>
</tr>
<tr>
<td></td>
<td>Grossman, N. 3/10, 6/25</td>
</tr>
<tr>
<td></td>
<td>Hall, J. D. 1/39, 3/35, 4/40, 5/41, 6/40</td>
</tr>
<tr>
<td></td>
<td>Ham, M. 3/23, 4/19</td>
</tr>
<tr>
<td></td>
<td>Harraison, D. W. 2/20</td>
</tr>
<tr>
<td></td>
<td>High-Level Packet Communication 5/32</td>
</tr>
<tr>
<td></td>
<td>Hore, M. 6/18</td>
</tr>
<tr>
<td></td>
<td>In-Word Parameter Words 6/9</td>
</tr>
<tr>
<td></td>
<td>Interactive Editing 1/24</td>
</tr>
<tr>
<td></td>
<td>Interpreters fig-FORTH 1/12</td>
</tr>
<tr>
<td></td>
<td>Forth P-Code Interpreter 4/9</td>
</tr>
<tr>
<td></td>
<td>Baden, W. 5/25</td>
</tr>
<tr>
<td></td>
<td>Berkey, R. 3/26, 4/28</td>
</tr>
<tr>
<td></td>
<td>Capitalization 1/20</td>
</tr>
<tr>
<td></td>
<td>CASE Statement, “YACS” 6/38</td>
</tr>
<tr>
<td></td>
<td>Chen, S. S. 6/9</td>
</tr>
<tr>
<td></td>
<td>China Tour 1984, FORML 5/38, 6/33</td>
</tr>
<tr>
<td></td>
<td>CODE, Mixing With High-Level Forth 4/37</td>
</tr>
<tr>
<td></td>
<td>Colon Word 3/20</td>
</tr>
<tr>
<td></td>
<td>Control Structures 2/20</td>
</tr>
<tr>
<td></td>
<td>Data Transfer 2/32</td>
</tr>
<tr>
<td></td>
<td>Debugging</td>
</tr>
<tr>
<td></td>
<td>An Augmented TRACE 5/18</td>
</tr>
<tr>
<td></td>
<td>Techniques 2/38</td>
</tr>
<tr>
<td></td>
<td>DO LOOP, Enhanced 6/18</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 simply the development of your applications.

MVP Books - A Series
□ Vol. 1, All about FORTH by Hayden. 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
□ Vol. 3, Floating Point with source code by Koopman $25
□ Vol. 4, Expert System with source code by Parx $15
□ Vol. 5, File Management System with interrupt security by Moreton $25
□ Vol. 6, Expert Tutorial for Volume 4 by M & L Derick $15
□ Vol. 7, FORTH GUIDE to MVP-FORTH by Hayden $20

MVP-FORTH Software - A transportable FORTH
□ MVP-FORTH Programmer's Kit including disk documentation Volumes 1, 2, 7 and 8.
□ MVP Series, and Starting FORTH. CP/M, IBM PC, XT/AT & compatibles. CP/M/MS-DOS. 1MB storage.
□ Apple with MicroDecisions, 1MB RAM. 5201 and 8086-80/1200. IBM $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 debugging, calldiagnosing, and debugging. $200
□ MVP-FORTH Cross Compiler for CP/M Programmer's Kit Generates heading code for HOM 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 PC or Apple I, II, IIe or III. An integrated system for customizing your FORTH programs and applications. The editor includes a bidirectional 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 $30
□ MVP-FORTH Floating Point & Matrix Math see above $25
□ MVP-FORTH Graphics Extension see above $65
□ MVP-FORTH EXPERT-2 System for learning and developing knowledge based programs. Both then procedures and analytical subroutines are available. Source code is provided. Specify Apple II, IBM. or CP/M. Includes MVP Books. Vol 4 & 6 $100
□ NewWorld/Hello A word processor for the IBM PC/XT/AT with 256K. MVP-FORTH compatible kernel with files. Edit and Print systems. Includes Disk and Calculate systems and ability to add additional FORTH words. $150

FORTH DISKS
□ APPLE by MM $125
□ APPLE by MM F & G $245
□ ATARI vs FORTH $125
□ ATARI by PNS, F. & G. X $150
□ C64 by HES Commodore $300
□ 64 Cartridge $125
□ C64 with EXPERT-2 by PS $29
□ Cartridge $25
□ APPLE by MM $125
□ CP/M by MM $125
□ IBM-PC by MM $185
□ IBM-PC by Cassady $125
□ IBM-PC by Cassady $65
□ Macintosh by MM $125

FORTH MANUALS, GUIDES & DOCUMENTS
□ Thinking FORTH by Lea Brodie. author of best selling "Starting FORTH" $16
□ ALL ABOUT FORTH by Hayden. MVP Glossary $25
□ FORTH Encyclopedia by Derek & Baker $25
□ ...FORTH from the Netherlands by User Manual $25
□ Source Listing $25
□ FORTH Tools and Applications by Forberich $15
□ The Complete FORTH by Winfield $16
□ Learning FORTH by Armstrong $17
□ Understanding FORTH by Reymann $3
□ An Applications Approach by Approach $20
□ FORTH Applications by Roberts $13
□ Mastering FORTH by Anderson & Tracy $18
□ Beginning FORTH by Chlikan $17
□ FORTH Encl. Pocket Guide $7
□ And G FORTH by Huang $17
□ FORTH Programming by Sharron $17
□ STARTING FORTH by Borel $17
□ Installation Manual for FORTH $20
□ Source Listings for FORTH. Specify CPU or Computer $15
□ Timex by HW cassette $125
□ 1/5000/2X-81 $25
□ 2068 $30
□ 780 by LM $100
□ 8086/86 by LM $100
□ 68000 by LM $250
□ VIC FORTH by HES. VIC20 Cartridge $20
□ Extensions for LM. Specify IBM 280 or 8086 $125
□ Software Floating Point $100
□ Software Floating Point $100
□ Color Graphics (280 or 8086) $100
□ Data Base Management $200

Key to Vendors:
□ LEW Hawk Wild Software
□ F - Floating Point
□ MicroMotion
□ F - Graphics
□ Apple
□ F - Graphics
□ PMK Pink Noise Studio
□ PS - Pas Sec
□ X - Other Extras

Codes:
□ M - Money
□ D - Display
□ P - Print
□ K - Keyboard
□ L - Loader

Ordering Information: Check, Money Order payable to MOUNTAIN VIEW PRESS. Inc.
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.

```
<esc> Restore Initial Value  <esc> Restore Initial Value
[0] Accept Sign  [0] Accept Sign
Delete Last Digit  AX
Clear Initial Value & Append Digit
Digit ≤  
Delete Last Digit

S0 INITIAL STATE  S1 ADD DIGIT STATE

<esc> Restore Initial Value  <esc> Restore Initial Value
Digit ≤  AX
Delete Last Digit

S2 FULL STATE  S3 ACCEPT STATE

(Max Digits Reached)
```

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.

```
: NUMBER-EDITOR--5.2
```

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.

```
: NUMBER-EDITOR--5.2
```

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.

```
: NUMBER-EDITOR--5.2
```

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.

```
: NUMBER-EDITOR--5.2
```

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.

```
: NUMBER-EDITOR--5.2
```

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.

```
: NUMBER-EDITOR--5.2
```

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.

```
: NUMBER-EDITOR--5.2
```

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.

```
: NUMBER-EDITOR--5.2
```

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.

```
: NUMBER-EDITOR--5.2
```

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.

```
: NUMBER-EDITOR--5.2
```

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.

```
: NUMBER-EDITOR--5.2
```
Valid keys are defined as follows:

**Keys 0 - 9**

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;cr&gt;</td>
<td>Accept current value</td>
</tr>
<tr>
<td>&lt;esc&gt;</td>
<td>Get initial value</td>
</tr>
<tr>
<td>X</td>
<td>Delete last digit</td>
</tr>
<tr>
<td>+</td>
<td>Change sign to positive</td>
</tr>
<tr>
<td>-</td>
<td>Change sign to negative</td>
</tr>
</tbody>
</table>

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.
MicroMotion Forth also has a set of string-handling words. I’ve used \texttt{SI} and \texttt{STRING} to create a single-dimension array of characters for my own word, \texttt{SELECT}.

\texttt{STRING} is a defining word that makes space for a character string. For example,

\begin{verbatim}
10 STRING MUSH
creates a dictionary entry called \textbf{MUSH} with space for ten characters. When you type \texttt{MUSH} it leaves the address of the first character and the length of the string on the stack. You could approximate it with

\begin{verbatim}
CREATE MUSH 10 C, 10 ALLOT
and then, when calling \texttt{MUSH}, type
\end{verbatim}

\textbf{MUSH COUNT}

to get the string pointer and length. The word \texttt{SI} moves a character string into the array part of the string variable. Some versions of Forth have the word "\texttt{,}" 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 \texttt{<esc>}, \texttt{<cr>} and \texttt{*X} as character literals in Forth.

\textbf{SELECT}

\texttt{SELECT} is a small utility with uses outside the \texttt{NUMED} package. Given a string of characters, \texttt{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 \texttt{SELECT} to \texttt{CASE} on the keystroke, frequently done in menu-driven programs.

\textbf{Implementation Notes}

\texttt{NUMED} has been implemented as a state machine where each state consists of an action part (within the \texttt{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
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  \(s\text{-addr len \text{--- index}}\)

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

NUMBER-EDITOR  \(row col dvall \text{--- dvall}\)

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.

FORTH Dimensions 40 Volume VII, No. 3
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

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 Forth–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 bus 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.
• 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 Sat., 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. Grevelle 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 Arne Flones
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., 7 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, 7 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
MINFIG 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.
Elliottville, MO 63011

• NEVADA
Southern Nevada Chapter
Call Gerald Hasty
702/452-3368

• NEW HAMPSHIRE
New Hampshire Chapter
Monthly, 1st Mon., 6 p.m.
Armit Equipment Industries
Sneape 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
716/233-0168

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

Cincinnati 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

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

• CENTRAL OKLAHOMA
Suite 612
Call Gary M. Granger
Dayton, OH
Central Oklahoma Chapter
2410 N.W. 49th
Oklahoma City, OK 73112
Health Tech. Bldg., OSU Tech.
Call Larry Somers
2410 N.W. 49th
Oklahoma City, OK 73112

• 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

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

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 Melanie Hoag
215/895-2628

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

• 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
703/880-9260

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

• 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
Lariksdrf 20
2120 Schoten
03/658-6343

Southern Belgium FIG Chapter
Contact Jean-Marc Bertinchamps
Rue N. Monnom, 2
B-6290 Ninines
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., 7 p.m., Rm. 408
Polytechnic of South Bank
Borough Rd., London
D.J. Neale
02/645-8688

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

• 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 Linschke
Common Interface Alpha
Schanzenstrasse 27
2000 Hamburg 6

• HOLLAND
Holland Chapter
Contact: Adriaan van Roosmalen
Heusden Houtestraat 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

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

• JAPAN
Japan Chapter
Contact Toshihiko 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
Stationstrasse
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
504/292-0395

FIGFRAME
Call Howard Pearlmuter
408/425-8700

Volume VII, No. 3 43
FORTH Dimensions
FORTH INTEREST GROUP
FALL Special

FORTH Dimensions
BACK VOLUMES 1 - 6

$50.00 $59.00 FOREIGN SURFACE MAIL
$90.00 FOREIGN AIR MAIL

USA AND CANADA

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