MULTITASKING MODEM PACKAGE

A FASTER NEXT

RELOCATABLE F83 FOR THE 68000

EDUCATING FORTH USERS
SUPERFAST FORTH SYSTEMS

AT/FORCE COPROCESSOR: 10 TO 50 FORTH MIPS

- Five chip Harris Core Set: 10 MHz Forth RISC Core
- 1-cycle 16 × 16 multiplier
- 1-cycle 15-channel interrupt
- two 64-word stack controllers
- Plugs into AT or 386
- Forth Software included

- 32K bytes main memory
- Expandable to 128K bytes
- AT shared memory space
- 2” × 3” prototyping area
- All Core signals available
- Runs concurrently with AT
- 2 weeks ARO: $3995

PC4000 COPROCESSOR: 4 TO 10 FORTH MIPS

- NC4016 Novix Forth Engine
- Plugs into PC/XT/AT/386
- 512K bytes main memory
- PC shared memory space
- Multiple PC4000 operation

- Forth Software included
- NC4016 signals available
- Runs concurrently with PC
- Compact ¾ length format
- 2 weeks ARO: $1295

Both ideal for real-time control, data acquisition and reduction, image or signal processing, or computationally intensive applications. For additional information, please contact us at:

SILICON COMPOSERS, 210 California Ave., Palo Alto, CA 94306  (415) 322-8763

SILICON COMPOSERS
MULTITASKING MODEM PACKAGE • BY JEFFREY R. TEZA

This terminal emulator is designed to be a useful example of Forth multitasking. If you spend much time telecommunicating, especially downloading and uploading files, you will appreciate the local processing provided by this program for F83 and other multitasking Forths. Add your favorite functions and share them with us!

DUMPING WORDSTAR FILES • BY PAUL A. COOPER

With or without embedded codes, your files can fly in Forth. Don’t choose between your favorite word processor and your Forth environment — get them working together.

A FASTER NEXT LOOP • BY CARL A. WENRICH

If you are willing to forgo byte-boundary addressing in your fig- or MVP-FORTH system for the 8086, a little tinkering can improve the execution speed of all your Forth code.

RELOCATABLE F83 FOR THE 68000 • BY ROBERT J. EAGER

No more excuses—not only can you compile F83 to run from any location in RAM under CP/M-68K, you can keep several distinct Forth kernels in memory at the same time.

EDUCATING FORTH USERS • BY BILL KIBLER

Would your first encounter with Forth have been different if it came with an on-line tutor to help with system functions, or to provide customized exercises linked to the chapters in Starting Forth? Here is a way to get users up and running....

PROFILES IN FORTH: MARTIN TRACY

Martin Tracy is a natural leader and expert programmer, until recently a Forth vendor, and is a current member of the Forth Interest Group’s Board of Directors.

CHARLES MOORE’S FIRESIDE CHAT • REVIEWED BY SCOTT SQUIRES

A perennially favorite event at the National Forth Convention is the "Fireside Chat" by Mr. Charles Moore, creator of Forth. Here, Scott Squires shares the notes he took as he listened to the informal session.

EDITORIAL

LETTERS

VOLUME EIGHT INDEX • BY MIKE ELOLA

Volume IX, Number 6

Forth Dimensions
EDITORIAL

I went to this year’s MacWorld Expo in San Francisco hoping to save big bucks on a hard disk, not because I expected to find much relevant to these pages. And after elbowing my way through the gridlocked aisles, comparing prices, and finding my purchase at a retail booth that resembled a Wall Street trading pit during last October’s frenzy, I cared less about scanning exhibits for familiar faces than about protecting my investment and getting out.

So it was a pleasant surprise to find Don Colburn, looking comfortable in the Creative Solutions booth, talking with passersby about his company’s NuBus products. It was good to see someone showing Forth’s strength on the current generation of machines, where the innovation and excitement is reminiscent of the elder days of microcomputing. With few exceptions, the absence of leadership shown by Forth companies to crowds like the one that packed Moscone Center on that particular day is surprising.

This is the first publication I’ve worked on where so few of those who could gain the most by sharing their ideas and business activities actually do so. A few even have the attitude that if someone doesn’t arrive at the office, elicit the information, and frame it in a meaningful context for them, they’ll just keep it to themselves. Well, FIG does provide Forth Dimensions to facilitate communication with, and among, its members, and FIG’s modest membership fee allows this to take place regularly and reliably in these pages, on GEnie, and at annual meetings. But FIG cannot do this for business owners, nor can the Forth Vendors Group. You have to make it happen. The advantages to association must be shared by all, but a strong business association relies on strong business members.

One final aside: I suggest it is time for the comatose Forth Vendors Group to be taken off its support systems (if any). Have a brief post-mortem exam, then reorganize. It’s springtime, in the northern hemisphere at least, and a good time for new beginnings. Give the entity a decent public burial and see what crops up. Maybe someone will propose a comfortable way for the FVG to organize under FIG to ensure continuity, communication, useful agendas, and to ease the administrative tasks. Whatever its form, the vendors need it and the Forth community needs it.

—Marlin Ouwerson
Editor

Forth Dimensions
Published by the
Forth Interest Group
Volume IX, Number 6
March/April 1988
Editor
Marlin Ouwerson
Advertising Manager
Kent Safford
Design and Production
Berglund Graphics
ISSN#0884-0822

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

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

Copyright © 1987 by Forth Interest Group, Inc. The material contained in this periodical (but not the code) is copyright by the individual authors of the articles and by Forth Interest Group, Inc., respectively. Any reproduction or use of this periodical as it is compiled or the articles, except reproductions for non-commercial purposes, without the written permission of Forth Interest Group, Inc. is a violation of the Copyright Laws. Any code bearing a copyright notice, however, can be used only with permission of the copyright holder.

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

"Forth Dimensions" is published bi-monthly for $30/42 per year by the Forth Interest Group, 1330 S. Bascom Ave., Suite D, San Jose, CA 95128. Second-class postage pending at San Jose, CA 95101. POSTMASTER: Send address changes to the Forth Interest Group, P.O. Box 8231, San Jose, CA 95155.”
Security Breach
Dear Marlin,
I have no excuses, I don’t know what happened. Clearly, the code printed with my letter on “F83 Execution Security” is wrong, with several errors in it. The enclosed, new code should work.

Sincerely,
G.R. Jaffray, Jr.
3536 Angelus Avenue
Glendale, California 91208

Security Breach
Dear Mr. Overson:
I’m a graduate student at the Florida Institute of Technology, doing extensive work in natural language processing (interfacing to an expert system and semantic knowledge base) with Forth under the direction of Dr. T.O. Hand. We intend to become one of the most advanced institutions for AI applications using Forth.
I am writing to convey my strong protest to the use of assembly code in source listings (e.g., “Local Variables,” Peter Ross, FD IX/4). It doesn’t provide the slightest bit of portability, and thus makes the source code absolutely worthless to those not using the same system as the author. By the same token, source listings should not be machine or implementation dependent (e.g., “Extensions for F83,” Scarpelli, FD IX/4).

In my four short years of Forth programming, I have always found a lack of useful programming, development, and debugging tools for this very reason. I have also come to the conclusion that any attempt to make Forth a structured programming language severely violates all aspects of Forth. If you want a structured language, use Pascal; it’s about as worthless as structured Forth. I am not trying to insult Carl Wenrich (“Readable Forth,” FD IX/4), I am merely saying that the structure he proposes is more of a hindrance than a programming aid. If readability is what Mr. Wenrich is after, I suggest he read, or re-read, Thinking Forth by Leo Brodie. There are better ways to make Forth more readable. Has anyone thought of making Forth more object oriented? We at FIT have. It has potential, and Dr. Hand has already begun exploration in this area.

This is my opinion as of this point in time. As my Forth experience increases, and as Forth continues to evolve, I may change these views; but not until the opposing facts are staring me in the face.

Sincerely,
Joe Sternlicht
3630 Misty Oak Dr., #1607
Melbourne, Florida 32901

Worthless Like Pascal
Dear Mr. Overson:
In my four short years of Forth programming, I have always found a lack of useful programming, development, and debugging tools for this very reason. I have also come to the conclusion that any attempt to make Forth a structured programming language severely violates all aspects of Forth. If you want a structured language, use Pascal; it’s about as worthless as structured Forth. I am not trying to insult Carl Wenrich (“Readable Forth,” FD IX/4), I am merely saying that the structure he proposes is more of a hindrance than a programming aid. If readability is what Mr. Wenrich is after, I suggest he read, or re-read, Thinking Forth by Leo Brodie. There are better ways to make Forth more readable. Has anyone thought of making Forth more object oriented? We at FIT have. It has potential, and Dr. Hand has already begun exploration in this area.

This is my opinion as of this point in time. As my Forth experience increases, and as Forth continues to evolve, I may change these views; but not until the opposing facts are staring me in the face.

Sincerely,
Joe Sternlicht
3630 Misty Oak Dr., #1607
Melbourne, Florida 32901

Security Breach
Dear Mr. Overson:
I’m a graduate student at the Florida Institute of Technology, doing extensive work in natural language processing (interfacing to an expert system and semantic knowledge base) with Forth under the direction of Dr. T.O. Hand. We intend to become one of the most advanced institutions for AI applications using Forth.
I am writing to convey my strong protest to the use of assembly code in source listings (e.g., “Local Variables,” Peter Ross, FD IX/4). It doesn’t provide the slightest bit of portability, and thus makes the source code absolutely worthless to those not using the same system as the author. By the same token, source listings should not be machine or implementation dependent (e.g., “Extensions for F83,” Scarpelli, FD IX/4).

In my four short years of Forth programming, I have always found a lack of useful programming, development, and debugging tools for this very reason. I have also come to the conclusion that any attempt to make Forth a structured programming language severely violates all aspects of Forth. If you want a structured language, use Pascal; it’s about as worthless as structured Forth. I am not trying to insult Carl Wenrich (“Readable Forth,” FD IX/4), I am merely saying that the structure he proposes is more of a hindrance than a programming aid. If readability is what Mr. Wenrich is after, I suggest he read, or re-read, Thinking Forth by Leo Brodie. There are better ways to make Forth more readable. Has anyone thought of making Forth more object oriented? We at FIT have. It has potential, and Dr. Hand has already begun exploration in this area.

This is my opinion as of this point in time. As my Forth experience increases, and as Forth continues to evolve, I may change these views; but not until the opposing facts are staring me in the face.

Sincerely,
Joe Sternlicht
3630 Misty Oak Dr., #1607
Melbourne, Florida 32901

Worthless Like Pascal
Dear Mr. Overson:
In my four short years of Forth programming, I have always found a lack of useful programming, development, and debugging tools for this very reason. I have also come to the conclusion that any attempt to make Forth a structured programming language severely violates all aspects of Forth. If you want a structured language, use Pascal; it’s about as worthless as structured Forth. I am not trying to insult Carl Wenrich (“Readable Forth,” FD IX/4), I am merely saying that the structure he proposes is more of a hindrance than a programming aid. If readability is what Mr. Wenrich is after, I suggest he read, or re-read, Thinking Forth by Leo Brodie. There are better ways to make Forth more readable. Has anyone thought of making Forth more object oriented? We at FIT have. It has potential, and Dr. Hand has already begun exploration in this area.

This is my opinion as of this point in time. As my Forth experience increases, and as Forth continues to evolve, I may change these views; but not until the opposing facts are staring me in the face.

Sincerely,
Joe Sternlicht
3630 Misty Oak Dr., #1607
Melbourne, Florida 32901

LETTERS
[I certainly agree that authors whose code's performance relies on assembly routines should also provide high-level Forth definitions for publication. F83-specific code, as you can see in this issue, will remain as long as many of our readers find it useful or educational. We optimistically believe that even most of the system-specific code we publish has value in terms of learning from others' techniques. And if you also learn a little about how a different Forth dialect or implementation works, all has not been lost. Still, we do give preference to work that is generalized for our readership without losing its pizzazz. As for the debate over structured programming, I'll let its proponents defend themselves, if they care to. —Ed.]

Ailing Acronyms

Dear Marlin:

A minor nit-pick: a Forth word may contain a name field, link field, code field, and parameter field. Because Forth so often keeps track of items by putting their addresses on the stack, we frequently talk about the addresses of those fields: the name-field address (NFA), link-field address (LFA), code-field address (CFA), and parameter-field address (PFA).

Many Forth writers and conference speakers confuse the two concepts, saying, for example, that the value of a constant is stored in its PFA. If we can successfully talk to computers (which do exactly what we tell them to do, whether we mean it or not), we should be capable of a bit more precision when communicating with each other. Perhaps the alphabet-soup addicts among us could be mollified by the introduction of the abbreviations NF, LF, CF, and PF for referring to the fields themselves.

Sincerely,
Carol Pruitt
University of Rochester
Lab for Laser Energetics
250 East River Road
Rochester, NY 14623

[Grammarians have been warning technical writers for some time about overusing acronyms, and you aren't the first Forth programmer to point out this particular problem. Is an author talking about the address of the field, or an address stored in the field? Your solution may be the least confusing so far, but authors should remember that acronyms don't make convoluted or repetitious writing any better, only shorter. —Ed.]
FUNCTIONALITY

More important than how fast a system executes, is whether it can do the job at all. Can it work with your computer. Can it work with your other tools. Can it transform your data into answers. A language should be complete on the first two, and minimize the unavoidable effort required for the last.

HS/FORTH opens your computer like no other language. You execute function calls, DOS commands, other programs interactively, from definitions, or even from files being loaded. DOS and BIOS function calls are well documented HS/FORTH words, we don't settle for giving you an INCALL and saying 'have at it'. We also include both fatal and informative DOS error handlers. installed by executing FATAL or INFORM.

HS/FORTH supports character or blocked, sequential or random I/O. The character stream can be received from/ sent to console, file, memory, printer or com port. We include a communications plus upload and download utility, and foreground/background music. Display output through BIOS for compatibility or memory mapped for speed.

Our formatting and parsing words are without equal. Integer, double, quad, financial, scaled, time, date, floating or exponential, all our output words have string formatting counterparts for building records. We also provide words to parse all data types with your choice of field definition. HS/FORTH parses files from any language. Other words treat files like memory, nnH and nnH read or write from/to a handle (file or device) as fast as possible. For advanced file support, HS/FORTH easily links to BTRIEVE, etc.

HS/FORTH supports text/graphic windows for MONO thru VGA. Graphic drawings (line rectangle ellipse) can be absolute or scaled to current window size and clipped, and work with our penplot routines. While great for plotting and line drawing, it doesn't approach the capabilities of Metawindows (tm Metagraphics). We use our Rosetta Stone Dynamic Linker to interface to Meta windows. HS/FORTH with MetaWindows makes an unbeatable graphics system. Or Rosetta to your own preferred graphics driver.

HS/FORTH provides hardware/software floating point, including trig and transcendental. Hardware fp covers full range trig log, exponential functions plus complex and hyperbolic counterparts, and all stack and comparison ops. HS/FORTH supports all 8087 data types and works in RADIANS or DEGREES mode. No process set? No problem. OpenGL (mostly fast machine code) and parse/format words cover numbers through 18 digits. Software fp eliminates conversion round off error and minimizes conversion time.

Single element through 4D arrays for all data types including complex use multiple cfa's to improve both performance and compactness. Z = (X-Y) / (X + Y) would be coded: X Y - X Y + / IS Z (16 bytes) instead of: X = Y @ - Y @ + /izards Arrays can ignore 64 boundaries. Words use SYNONYM's for data type independence. HS/FORTH can even prompt the user for retry on erroneous numeric input.

The HS/FORTH machine coded string library with up to 32 arrays is without equal. Segment spanning dynamic string support includes insert, delete, add, find, replace, exchange, save and restore string storage.

Our minimal overhead round robin and time slice multitaskers require a word that exists cleanly at the end of subtask execution. The cooperative round robin muti tasker provides individual user stack segments as well as user tables. Control passes to the next task/user whenever desired.

APPLICATION CREATION TECHNIQUES

HS/FORTH assembles to any segment to create stand alone programs of any size. The optimizer can use HS/FORTH as a macro library, or complex macros can be built as colon words. Full forward and reverse labeled branches and calls complement structured flow control. Complete syntax checking protects you. assembler programming has never been so easy.

The Metacomplier produces threaded systems from a few hundred bytes, or Forth kernels from 2k bytes. With it, you can create any threading scheme or segmentation architecture to run on disk or ROM.

You can turnkey or seal HS/FORTH for distribution, with no royalties for turnkeyed systems. Or convert for ROM in saved, sealed or turnkeyed form.

HS/FORTH includes three editors, or you can quickly shell to your favorite program editor. The resident full window editor lets you reuse former command lines and save to or restore from a file. It is both an indispensable development aid and a great user interface. The macro editor provides reinsurance functions, cut, paste, file merge and extract, session log, and RECOMPILE. Our full screen Forth editor edits file or sector mapped blocks.

Debug tools include memory stack dump, memory map, decompile, single step trace, and prompt options. Trace scope can be limited by depth or address.

HS/FORTH lacks a "modular" compilation environment. One motivation toward modular compilation is that, with conventional compilers, recompiling an entire application to change one subroutine is unbearably slow. HS/FORTH compiles at 20,000 lines per minute, faster than many languages link. We don't let alone compile! The second motivation is linking to other languages. HS/FORTH links to foreign subroutines dynamically. HS/FORTH doesn't need the extra layer of files, or the programs needed to manage them. With HS/FORTH you have source code and the executable file. Period. "Development environments" are cute, and necessary for unnecessarily complicated languages. Simplicity is so much better.

HS/FORTH Programming Systems

Lower levels include all functions not named at a higher level. Some functions available separately.

Documentation & Working Demo

(3 books, 1000 + pages, 6 lbs) $95.
Student $145.
Personal optimized, scaled & quad integer $245.
Professional $95.
Native $395.
RDV editor $145.
RSDL linker $95.
Physical screens $295.

HARVARD SOFTWORKS

PO Box 69
Springboro, OH 45066
(513) 748-0390

More information is available from our sales department on these and other products.

Yes, there is a better way.

A Forth that actually delivers on the promise.
A few modem I/O programs have been published in Forth Dimensions [JAM85] [ERI84] [ACK83]. These have provided good examples of serial-line interface basics. Armed with this knowledge, here is a slightly more advanced terminal emulator, designed to be a useful terminal package and to serve as an example of a Forth multitasking application.

One useful feature of using a computer to emulate a dumb terminal is the ability to do local processing. Services such as automatic dialing, phone lists, and file upload/download at your fingertips can make sitting at a slow modem more tolerable. This can be a touchy thing to program, however, since the real-time nature of a modem package requires that a local process run to completion within a character time. If this restriction is violated, the modem may lose incoming data.

Forth's asynchronous approach to multitasking provides a very fast context switch between tasks. Often, this is just a few machine instructions, and can be as fast as a "busy," high-level Forth loop. A terminal emulator is usually coded as just an infinite loop passing characters back and forth from modem to console. The code in screen 9 shows two tasks which could be written as one *BEGIN* *AGAIN* loop, but instead use a Forth multitasker to glue the two together. By running one in the "background" and the other in the "foreground," this structure has an advantage for a terminal emulator. It allows the *KEYBOARD* task to spend some time doing different functions, while the *MODEM* (background) task continues to pay attention to any characters being received at the serial port.

Now, one of the problems with these two tasks going about their merry way is what to do with characters coming in from the modem while the *KEYBOARD* task is goofing off. This is where screens 3 and 4 come in. These two screens create a first-in, first-out buffer, which allows the two tasks to communicate on a slightly relaxed schedule. Stubborn characters that refuse to wait for the *KEYBOARD* task to complete a job are stored in this FIFO buffer to be picked up later by *KEYBOARD* and displayed to the user.

"What shall we do with all this time?"

Great. Now we can take a little vacation in the *KEYBOARD* routine without feeling pressured to whip through the loop in time for another character. What shall we do with all this time? Many things come to mind, some of which are shown in the example. Screen 8 creates a jump table that detects a control key pressed at the keyboard and sends the *KEYBOARD* task off on vacation. I've coded a few interesting tools for a user sitting at a modem talking to another computer.

The first is taken from an elegant little piece published by Leo Brodie [BRO83]. It is a "breakpoint interpreter" which runs a Forth QUIT loop (shell). This essentially allows the user to jump up to a Forth interpreter riding on "top" of the modem software.

This is shown on screens 5 and 6, and is entered into the jump table in the ASCII 6 (Ctrl-F) key slot. Now you have complete access to the Forth dictionary — which should provide an adequate selection of local-processing tools!

Another tool, shown on screens 7 and 11, provides a telephone list. These numbers can be assigned to a key and automatically dialed by the modem with the mere stroke of a control key; or they can be stored in a vocabulary (*PHONE*), to be executed from the breakpoint interpreter.

I find this to be a clean and useful application, and have endeavored to provide good comments. The end-user word is *CONVERSE*, which takes a baud rate as a parameter and launches the two processes (for example, type 1200 *CONVERSE*).

A few words about dialect. The code is written in Laxen and Perry's F83. I've tried to comment any non-83-Standard code in the shadow screens, but the multitasking word *BACKGROUND* may have to be changed according to your multitasking word set. My apologies to people without a multitasker. Some of the ideas here can be implemented in a single-task system. But considering the simplicity of Forth multitasking, and with Henry Laxen's excellent tutorial [LAX84] [LAX83], serious Forth vendors should consider providing this important aspect of a Forth environment.

This code runs fine at 1200 baud on my 8 MHz 80186 system. Lost characters
could still be a problem for very slow PAUSE loops on slower machines. If this is
a problem, all I can suggest is to code the MODEM incoming-character receiver in
assembler, or to make it interrupt driven.
As demonstrated here, the speed of properly optimized, Forth multitasking loops is
often a desirable alternative to a high-level Forth loop. Chances are, a slow computer
would require a bit of assembler, even for a simpler terminal program.

References

1983.

1983.

[ERI84] Ericson and Feucht. “Simple Data Transfer Protocol,” Forth Dimensions, Vol 6 No 2,

[JAM85] James, John S. “Simple Modem I/O Words,” Forth Dimensions, Vol 6 No 5, Jan/Feb
1985.


Dumb Terminal

Dumb Terminal local processing

0 \ Dumb Terminal \ local processing
1 \ DIAL (S addr len f--) OUTGOING QTYPE OUTGOING OCR ;
2 3 FORTH DEFINITIONS
4 \ BOOK WORDS ;
5 VOCABULARY PHONE PHONE DEFINITIONS
6 7 11 LOAD \ Phone numbers
8 \ CALL* HEADER ASCII " WORD COUNT DIAL ;
9 10 TALKING DEFINITIONS
11 12 13 14 15

Dumb Terminal \ keyboard/modem tasks

0 \ Dumb Terminal \ keyboard/modem tasks
1 2 BACKGROUND: MODEM
3 BEGIN OUTGOING QUEUE IF EMIT THEN
4 KEY? IF KEY INCOMING QUEUE THEN PAUSE AGAIN ;
5 6 7 : KEYBOARD
8 BEGIN INCOMING QUEUE IF EMIT THEN
9 KEY? IF KEY OUTGOING FILTER THEN PAUSE AGAIN ;
10 11 12 13 14 15

Dumb Terminal \ converse/quiet

0 \ Dumb Terminal \ converse/quiet
1 \ Converse invokes the dumb terminal
2 \ CONVERSE (S baud--) INITIALIZE INCOMING OQUEUE
3 OUTGOING OQUEUE MULTI MODEM WAKE KEYBOARD ;
4 5 : QUIET (S --) MODEM SLEEP [\] (?)ERROR IS ?ERROR ABORT ;
6 7

0 \ Phone Numbers
1 \ HEADER " ATDT " OUTGOING QTYPE ;
2 : FIG HEADER " 4155333580" DIAL ;
3 4 -->
5 6 7

Forth Dimensions
This is the load block for the dumb terminal emulator.

This dumb terminal emulator uses the Laxen & Perry F83 multitasking capabilities by defining separate keyboard and modem tasks. These two tasks communicate via a FIFO queue. This structure allows local processing without losing characters.

A convenient technique for invoking local processing words is used via a control character table as in F83. One local word that can be invoked in this example is a breakpoint interpreter as published by Leo Brodie. This allows you to exit to a higher forth QUIT "shell" giving you complete access to the forth dictionary while terminal emulating.

This screen contains all of the hardware specific code to talk to the modem port. These words are analogous to FORTH i/o words.

ODEPTH is the depth of the FIFO queues. This depth should be adjusted according to how smooth the multitasking loop is running (total task activity). Note that at 1200 baud a character comes in every 5s or so and without interrupt driven modem control the loop must average shorter than this to avoid queue overflow and lost characters.

INCOMING and OUTGOING are both byte queues of ODEPTH length whose first cell is a pointer to the front of the list and second cell is a pointer to the back of the list.

QUEUE initializes a queue so the two pointers point at the same queue entry.

QUEUE increments a queue pointer circularly.

!QUEUE pushes a byte at the front of a queue. It first increments the front pointer then checks for an overflow. Note that if an overflow occurs it will continue to place characters in the queue causing the queue to be dumped by incrementing the front pointer past the rear pointer.

@QUEUE removes a byte from the back of the queue if one is available and returns either the character and a true or a false flag if the queue was empty.

This breakpoint interpreter was published by Leo Brodie in FD vol15 no1. MVP-FORTH changes are shown as inline comments.

BREAK invokes an outer interpreter or "shell".

RESUME is used to resume from the BREAK shell. If the return stack is messed up use KEYBOARD to restart.

?BREAKERROR? ERROR is the F83 vectored ABORT error handler.

This word is used to return to the BREAK shell on errors.

BREAK calls the breakpoint interpreter after cleaning up the stack and vectoring the new error handler.

BEEP rings the bell after cleaning up the stack.

PRINT toggles the printer on/off in an F83 system.

QTYPE types a string to a queue.

QCR puts a carriage return ( ascii 13) in a queue.

DIAL types a string to the outgoing queue followed by a cr.

Put the PHONE vocabulary in the FORTH vocabulary.

Use PHONE BOOK to see the phone numbers.

Put the phone numbers in the PHONE vocabulary.

Use PHONE 〈phone#〉 e.g. PHONE FIG to dial a number.

Use CALL "XXXXXX" to call a number not in the PHONE BOOK.

Back to the application vocabulary.
CONVERSE takes a baud rate as a parameter, initializes the modem port, zero's the queues, fires up the multitasker and enters the KEYBOARD infinite loop.

QUIET puts the modem task to sleep and aborts the system.

FILTERTABLE is a table indexed into by a control character. Note it contains 32 entries which can be any FORTH word which will be invoked by a control character pressed at the keyboard.

FILTER takes a character and queue address and looks up control characters in FILTERTABLE for execution, otherwise sticks it in the queue. (PERFORM is @EXECUTE)

MODEM is a background task which gets outgoing characters and writes them to the modem port and incoming characters and writes them in the incoming queue.

KEYBOARD is the terminal task which gets incoming characters and prints them. Keyboard entered characters are FILTER'd which either does something or sends them to the outgoing queue.
This article deals with the problem of implementing large data files which are to be sent to another device (other than a local printer or video display) while directly in Forth.

It's relatively easy to use an editor utility, and many of them are on the market. Today, one finds a plethora of word processor utilities available, but a survey I did recently showed that the preponderance of those in use has the name — you guessed it — WordStar.

The ubiquitous WordStar is a difficult learning experience for most; and one which, when learned, is almost impossible to remove from one's use. That being said, let's assume for the purposes of discussion, that you love the utility and, at the same time, you use Forth quite a bit, too. Well, you've probably found that, from time to time, you'd like to access a WordStar file directly while inside Forth.

That's exactly the situation I found myself in, because I wanted to be able to send data files over serial lines or over the air via amateur radio bands. And I wanted to do it while in Forth, because Forth is my language of choice. But I knew there was a problem with this because WordStar uses strange little codes embedded in the text files to make wonderful little things happen at the printer and console. I had to devise a way to read those codes and either use or discard them. And I realized that this situation would be magnified if I wished to use a word-wrapped file.

In the transmission of ASCII data, we are really only concerned with characters represented by ASCII 32 - 126 but including 13 and 10, which make up the carriage return. If WordStar included only these codes, we would just need to get into DOS, read a sector of the disk file, output that sector, and repeat until we exhaust the file. As stated, the control codes used by WordStar cause a big problem. But anything can be solved in Forth, right?

As a matter of fact, within a word-wrapped file, WordStar uses a great many codes from ASCII 160 - 254 and A0 - FE, among others, in addition to the standard codes mentioned above — the ones we'd like to use by themselves, but can't. The trick, therefore, is to use a lookup table to determine which character to output.

I have provided seven screens of code in LMI's PC-Forth version 3.1. It will be necessary for readers using another system to incorporate their version of the assembler and DOS interface. LMI provides its customers with a quick disk interface, which speeds up sector access greatly.

Screen #1 contains an assembly language word INDARR that sets up indexed user arrays. In our case, we use this array to set up a 254-section lookup table (in screen #3), appropriately named WRAP. The code words << and >> mark the stack top for the array, then mark the end-of-fill process. In screen #2, the DOS interface is invoked, a 128-byte buffer is established, and our interrogated file is subnamed; the end-of-file flag (-1) is made a constant, and several commands to open, close, and read are defined.

In screen #4, a case word WRAPCASE explicitly actions three different hex values: 8D,
which WordStar uses as a hidden carriage return; OD, the standard carriage return; and 1A, used as a \rightarrow to mark leftover, unused bytes in each line on the monitor. (One would think these would be spaces or 20 hex, but in the words of Shakespeare, alas and alack, no. MicroPro entertainers had their reasons, I’m certain.) These 1A codes are dropped. The balance of any codes read are diverted to the lookup table, which leaves on the stack the proper ASCII representation; if a code 00 hex is left, it is dropped as irrelevant to our use.

The word DUMP-FILEBUFFER in screen #4 is the main action word. It looks at each byte that is read into FILEBUFFER and uses WRAPCASE if the code is below decimal 32 (hex 20) or above decimal 126 (hex 7E); if not, it merely emits it. Screen #5 contains some keyboard interaction words that allow immediate escape from a dump or a hold; if in hold, one may continue the dump or escape. Screen #6 holds the main word DUMP-FILE; the operator is prompted for the path and filename. Let’s say you want to dump the file named MYFILE.EXT that is on drive A. Just respond with A:MYFILE.EXT<cr>. Your file will completely dump in the original form as shown on the WordStar dump, but it won’t have any of the special codes (e.g., printer codes). An added advantage of this Forth dump is that you will have a continuous printout that you can stop anywhere, instead of using ^C to keep going. Just hit a key to stop/continue; if you wish to get out, press Esc.

In screen #7, I show a separate DUMP-FILEBUFFER word. This can be used in place of the word of the same name in screen #4 if you want to display all WordStar codes on your screen. This was written for my own use in tracking down the various codes needed to set up the lookup table WRAP and WRAPCASE. But it might be fun for some to see just what WordStar does inside a word.
wrapped file. Should you want to actually place, say, printer codes in the data you transmit, you can place the appropriate hex value within the lookup table.

For example, the value 01 hex is used for boldface printing. Therefore, by placing 01 in the second value in the lookup table (the second 00 currently to the right of << on line 2 of screen #3), the dump would emit the symbol for 01 hex. If your data were going to a remote printer on the other end of a modem, after going through a receiving program, that printer would start printing in boldface (assuming the program were set up to action WordStar codes).

In order to send this data, byte by byte, out the serial port, use an output word in place of the word EMIT. If your system has the facility to redirect output from the console to the serial port, en masse, this is also a choice. I prefer the byte method because I feel I have more direct control of the machine.

To use this facility, you must first load your assembler and DOS interface (if you don't use the program as it is). Again, any other Forth system will require some massaging of the assembler words and implementation of the DOS interface.

Copyright © 1987 by Paul A. Cooper. All rights reserved. Permission granted for any individual reader of Forth Dimensions to use for personal use only.

---

**Screen # 4**

(Dump Wordstar word wrap file pac 16:55 12/26/86)

```
HEX
: WRAPCASE \ use lookup table to action Wordstar codes
  DUP
  CASE 8D OF DROP CR ENDOF \ if, do a CR
  1A OF DROP NOOP ENDOF \ if, drop
  0D OF DROP CR ENDOF \ same as 8D
  WRAP @ DUP 00 = IF DROP \ if 00, drop
  ELSE EMIT THEN ENDCASE \ anything else, emit

DECIMAL
: DUMP-FILEBUFFER \ addr n ---
  OVER + SWAP DO I 16 + I \ look at each disk byte sector
  DO I CR DUP 32 < OVER 126 > OR IF WRAPCASE \ if, do
  ELSE EMIT \ if not, emit the standard character
  THEN LOOP 16 +LOOP ; \ go back and do it again
```

**Screen # 5**

(Dump Wordstar word wrap file pac 16:55 12/26/86)

```
: EXITWORD \ advise status of action
  CR CR \ You have exited the DUMP-FILE routine! CR CR ;
: KEYACTION \ query keyboard and take action if necessary,
  \ if ESC, quit; if any other key, wait
  ?TERMINAL IF KEY
  DUP 27 = IF DROP EXITWORD CLOSE-IT QUIT
  ELSE DROP KEY 27 = IF EXITWORD CLOSE-IT QUIT
  THEN THEN THEN ;

-->
```

**Screen # 6**

(Dump Wordstar word wrap file pac 16:55 12/26/86)

```
: DUMP-FILE \ get file from Wordstar and emit
  GET-FIENNAME \ enter path and filename
  OPEN-IT \ open it
  BEGIN KEYACTION READ-IT
  EOF <> \ end of the file or not?
  WHILE DROP \ drop the status flag
  FILEBUFFER 128 DUMP-FILEBUFFER \ read and dump
  REPEAT
  CLOSE-IT \ close the file
  CR ;
```

**Screen # 7**

(Dump Wordstar word wrap file - emit all pac 16:55 12/26/86)

```
: DUMP-FILEBUFFER \ addr n ---
  OVER + SWAP DO I 16 + I \ look at each disk byte sector
  DO I CR DUP 32 < OVER 126 > OR IF WRAPCASE \ if, do
  ELSE EMIT \ if not, emit the standard character
  THEN LOOP 16 +LOOP ;
```

---
One of the advantages of using Forth in preference to other high-level languages is the speed of the compiled code. This speed is in large part due to the efficiency of the inner interpreter, known as the NEXT loop. All it has to do is fetch the address of the instruction to be interpreted (IP), save it in the working register (W), and then increment IP by two to point to the next instruction in the list. The processor then falls into a section of code called NEXT1 that places the address pointed to by W into the processor's PC, and the jump is made. Glen B. Haydon describes these functions in high-level Forth terms in his book All About Forth, as follows:

```forth
: NEXT
  IP @ @
  W ! 2 IP +!
  NEXT1 ;

: NEXT1
  W @ @ PC ! ;
```

This article proposes to demonstrate how two existing 8086 implementations (fig-FORTH for IBM PC 1.0 and MVP-FORTH version 1.0305.03) can be made to run faster by decreasing the NEXT loop overhead. It involves changing a few other portions, such as DOCOL, DOCON, DOVAR, DO-DOES, DOUSE, and EXEC, but the increased speed of the loop seems well worth the effort. If you are running Forth on a different processor, you should be able to make similar changes with similar results.

Figures One and Two show assembly source code for the fig-FORTH and MVP-FORTH versions of the NEXT loop implementation. The LODSW instruction in the fig-FORTH version does what the first three instructions of the MVP-FORTH version do. The source index register (SI) on the 8086 is used as the interpreter pointer. AX and BX are used as general-purpose registers, and DX is assigned as the Forth working register W.

Both versions pick up the address of the instruction to be interpreted, increment IP, and then jump to the definition. And both increment the working register W by one before making the jump. The other required increment is deferred to the defining word interpreters DOCOL, DOCON, DOVAR, DO-DOES, and DOUSE. This was probably done to allow for byte boundary addressing upon entry to new defining word routines. But I have yet to see any need for this capability. So, not having any particular use for it, I eliminated it.

Figure Three shows a version that executes about 12% faster than the fig-FORTH version, and about 25% faster than the MVP-FORTH version. BX is now used as the working register, and DX is completely out of the picture. It also uses a little less memory, but that is of little consequence. What is significant is the time saved, since NEXT is executed so often.

---

**MVP-FORTH NEXT**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>No. of 8086 Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEXT: MOV AX, [SI]</td>
<td>13</td>
</tr>
<tr>
<td>INC SI</td>
<td>2</td>
</tr>
<tr>
<td>INC SI</td>
<td>2</td>
</tr>
<tr>
<td>MOV BX, AX</td>
<td>2</td>
</tr>
<tr>
<td>NEXT1: MOV DX, BX</td>
<td>2</td>
</tr>
<tr>
<td>INC DX</td>
<td>2</td>
</tr>
<tr>
<td>JMP WORD PTR [BX]</td>
<td>15</td>
</tr>
</tbody>
</table>

Total no. of 8086 cycles required: 38

**Abbreviated NEXT**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>No. of 8086 Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEXT: LODSW</td>
<td>12</td>
</tr>
<tr>
<td>MOV BX, AX</td>
<td>2</td>
</tr>
<tr>
<td>JMP WORD PTR [BX]</td>
<td>15</td>
</tr>
</tbody>
</table>

Total no. of 8086 cycles required: 29
Of course, there is a small price to pay. The defining word interpreters have to be modified, and EXEC has to be changed to eliminate the dependence on NEXT1. But the changes are easy to implement, and the difference in memory and cycle-time requirements is minimal. Figures Four through Ten show how the changes can be implemented.

```
NEXT:  LODSW
   MOV  BX,AX
NEXT1: MOV DX,BX
   INC DX
   JMP WORD PTR [BX]
```

Figure One. fig-FORTH NEXT.

```
NEXT:  MOV AX,[SI]
   INC SI
   INC SI
   MOV BX,AX
NEXT1: MOV DX,BX
   INC DX
   JMP WORD PTR [BX]
```

Figure Two. MVP-FORTH NEXT.

```
NEXT:  LODSW
   MOV BX,AX
   JMP WORD PTR [BX]
```

Figure Three. New NEXT.

```
DOCOL:  INC DX  ; FIG & MVP
   DEC BP
   DEC BP
   MOV [BP],SI
   MOV SI,DX
   JMP NEXT
```

Figure Four. fig-FORTH and MVP-FORTH DOCOL, and new version.
1. Buy MMSFORTH before year's end,
to let your computer work harder and
clean.

2. Then MMS will reward it (and you)
with the MMSFORTH GAMES DISK,
a $99.95 value which we'll add on at
no additional charge.

MMSFORTH is the unusually smooth
and complete Forth system with the
great support. Many programmers report
four to ten times greater
productivity
with this outstanding system, and MMS
provides advanced
epplications
programs in Forth for use
by
beg~nners
for custom
mod~fications.

Unlike many
Forths
on the market, MMSFORTH gives
you a rich set of the instrucrens, edittng
and debuggtng tools that professronal
programmers want. The licensed user
gets continuing, free phone
ten
tips and a
MMSFORTH Newsletter
IS
available.

The MMSFORTH GAMES DISK includes
arcade games (BREAKFORTH, CRASH-
FORTH and, for TRS-80, FREEWAY),
board games (OTHELLO and
TIC-TAC-
FORTH), and a top-notch
CRYPTO-
QUOTE HELPER with a data file of
coded messages and the ability to en-
code your own.

All
of these come with
Forth source code, for a valuable and
enjoyable demonstration of Forth pro-
gramming techniques.

Hurry, and the GAMES
DISK
will be our
free gift to you. Our brochure is free,
and our knowledgeable staff
is
ready to answer your questions. Write.
Better yet, call 817/653-6138.

DOCON: INC DX ; FIG & MVP
MOV BX,DX
MOV AX,[BX]
JMP APUSH

DOCON: INC BX ; NEW
INC BX
MOV AX,[BX]
JMP APUSH

Figure Five. fig-FORTH and MVP-FORTH DOCON, and new version.

DOVAR: INC DX ; FIG & MVP
PUSH DX
JMP NEXT

DOVAR: INC BX ; NEW
INC BX
PUSH BX
JMP NEXT

Figure Six. fig-FORTH and MVP-FORTH DOVAR, and new version.

DOUSE: INC DX ; FIG & MVP
MOV BX,DX
MOV BL,[BX]
SUB BH,BH
MOV DI,UP
LEA AX,[BX+DI]
JMP APUSH

DOUSE: INC BX ; NEW
INC BX
MOV BL,[BX]
SUB BH,BH
MOV DI,UP
LEA AX,[BX+DI]
JMP APUSH

Figure Seven. fig-FORTH and MVP-FORTH DOUSE, and new version.
**DODOE:**

```
XCHG  BP,SP  ; FIG
PUSH  SI
XCHG  BP,SP
INC   DX
MOV   BX,DX
MOV   SI,[BX]
INC   DX
INC   DX
PUSH  DX
JMP   NEXT
```

**DODOES:**

```
XCHG  BP,SP  ; NEW
PUSH  SI
XCHG  BP,SP
INC   BX
INC   BX
MOV   SI,[BX]
INC   BX
INC   BX
PUSH  BX
JMP   NEXT
```

*Figure Eight.* fig-FORTH DODOE, and new version.

```
DODOES:  INC   BX  ; MVP
DEC   BP
DEC   BP
MOV   [BP],SI
POP   SI
PUSH  DX
JMP   NEXT
```

```
DODOES:  INC   BX  ; NEW
INC   BX
DEC   BP
DEC   BP
MOV   [BP],SI
POP   SI
PUSH  BX
JMP   NEXT
```

*Figure Nine.* MVP-FORTH DODOES, and new version.

```
EXEC  DW  $+2  ; FIG & MVP
POP   BX
JMP   NEXT1
```

```
EXEC  DW  $+2  ; NEW
POP   BX
JMP   WORD PTR [BX]
```

*Figure Ten.* fig-FORTH and MVP-FORTH EXEC, and new version.
Are you an avid user of Laxen and Perry’s F83? Do you own or use a 68000-based machine under CP/M-68K? And do you find that F83 for the 68000 does not run on your system? If you answered yes to these questions, you will find the following article of interest. This paper describes the modification to F83 that enables the user to compile and execute F83 anywhere within the 68000’s address space. The techniques discussed also provide the basis for creating a multiple-image Forth system.

The creation of F83 was a major milestone for the Forth community. Here was a public-domain Forth system with all the essence of a professional package. And with the release of the 68000-based version, F83 became a truly powerful implementation.

One of the many design decisions Laxen and Perry had to make was how to handle F83’s I/O. The result was a tradeoff between minor performance degradation for computer transportability. By constructing F83 to utilize basic DOS and I/O routines defined by the two most common operating systems (CP/M and MS-DOS), Laxen and Perry effectively reduced implementation dependency down to the CPU level. As a result, F83 for the 808x was written to run on either CP/M- or MS-DOS-based computers (i.e., Kaypro, IBM PC, etc.); and F83 for the 68000 was written to execute on CP/M-68K-based computers (i.e., Sage IV, MASCOMP, HP9920, etc.).

When Laxen and Perry implemented their 68K version, however, they diverged from their original goal of portability by imposing an addressing restriction on their code. This restriction forced their implementation to work only in the lowest 64K of the 68000’s address space. As a result, only systems with RAM memory at this location could run it. Many machines, however, have their ROM memory located in this region and their RAM memory elsewhere in the 16Mb address space. Figure One displays the memory map of such a computer (the Hewlett-Packard 9000 Family 200 Series Technical Workstation).

“L&P implemented part of this construct.”

Fortunately, a study of F83’s source code revealed a means to extend the system’s addressing with minor impact on code size and efficiency. This ‘extended’ addressing would enable the programmer to target the compiled code to any 64K bounded area, henceforth described as a page of memory. For the 68000, each page of memory has a unique highest-address-byte value. The hex address representation is SXX YYYY, where XX represents the unique page address and YYYY represents the local address within that particular page. For example, the first addressable page starts at $000000, the second at $010000, and the last at $FF0000.

By constraining the 68000’s addressable memory to exist between these pages of memory, the original F83’s addressing can be used with minimal address-conversion overhead. The conversion involves the use of a CPU register to address-extend Forth’s local addresses into their absolute address equivalent. On boot-up, a designated register has its upper 16 bits initialized with the memory page address in which the executing Forth system resides. Subsequent memory accesses are done by taking the Forth’s 16-bit local address and copying it into the lower 16 bits of this register, and then using the whole register (all 32 bits) to address memory. (See Figure Two.)

Laxen and Perry, interestingly enough, have already implemented part of this conversion construct in order to correct for an addressing anomaly unique to the 68000. Whenever a short address (16 bits) is loaded into an address register, the 68000 extends its sign bit to form a 32-bit address. This results in a 64K region of addressable memory that starts in the lowest 32K ($000000 through $007FFF) and skips to the highest 32K ($FF8000 through $FFFFF). (See Figure Three.)

To compensate for this unorthodox addressing behavior, F83 loads the short address into a data register (D7), then copies it into an address register (A0) as a 32-bit value. It is this address that the original Forth system uses to access memory. By placing the desired page address (0000 through 00FF) into the upper 16 bits of D7 on boot-up, we can use the original code to automatically expand Forth’s local addresses into their 32-bit, absolute equivalents. This modification and other recoding resulted in the new Forth system that can execute at any page of memory the system was compiled for.

As a result of these code changes, all high-level Forth words are unaffected.
both in function and code size. Assembly code words which do not contain short absolute addresses are also unaffected. For those that do, their code has been modified to use long absolute addressing. For the programmer who is coding in assembly, only two adjustments in coding technique must be made. Follow each short absolute address with the system’s base-page address (BPAGE) and use the long absolute assembly mnemonic [i.e., L# vs. #)] whenever addressing memory directly.

For example, to copy the 16-bit value from the variable SPAN to D1, instead of using:

\[
\text{SPAN \\
#) D1 MOVE
\]

write:

\[
\text{SPAN BPAGE \\
L#) D1 MOVE
\]

where BPAGE is a 16-bit constant which returns the 64K page address to which the system was compiled. A second constant BOFFSET specifies the starting address of the kernel within this page. These two words are defined twice in the kernel source: Screen #1 contains the definitions used by the metacompiler to target the kernel, and screen #84 contains the definitions for use by the Forth programmer.

To compile the kernel at a new target address, simply edit the definitions BPAGE and BOFFSET in screen #1 of the kernel, and compile the system as described by Laxen and Perry. (See Listing One, screen #1.) Note: each listing contains the original source screens in the left column, and its corresponding ‘extended’ source screens in the right column. Screen lines (in the ‘extended’ source) that actually contain modified code are marked with a vertical bar at the beginning of the line.

Extending F83 resulted in 10 words and five source screens requiring modifications in the KERNEL68 file (see Listing One), and 10 words and one source screen in the EXTEND68 and CPU68000 files (see Listing Two). No modifications were required in either the METACOMPILER or UTILITY files.

The effect on code size and execution speed was as follows: a 640 byte growth in size and a 3% increase in execution time (based on the benchmark published in Forth Dimensions VIII/4 — see Listing Three).

To run this modified F83 (F83X), your system must meet only three requirements. First, the computer must be 680X0 based. Second, the computer must use CP/M-68K as its operating system. Third, the Transient Program Area (TPA) of the CP/M-68K system must define a region of memory that contains at least one 64K bounded page of RAM (i.e., $F2 0000 through $F2 FFFF). (See Figure Four.)

For those of you interested in loading F83X onto your system, I have written a CP/M-68K relocatable program called GENF83X.REL that enables the user to bootstrap a relocatable version of F83X, called F83XREL.HEX, onto their system. To install F83X, the users copies GENF83X.REL and F83XREL.HEX onto an empty, formatted diskette and places it into the default drive. The user then ex-
After loading, the program will display the system's available TPA and prompt the user for the 64K page they wish to target the code for. Pressing <cr> without entering a number will result in the F83X code being targeted for the first available 64K page of memory. The resulting code will be written to the default drive, with the filename F83X.68K.

With these enhancements to F83, anyone who runs a 68000-based computer under CP/M-68K, with at least 64K of programming space, can run F83X. For those wishing to take advantage of their extra memory pages, one could load several versions of F83X into memory, each tailored for a specific task (i.e., word processing, spreadsheet, terminal emulation, etc.). By adding code to allow the user to jump between these self-contained systems, it is relatively simple to create a multiple-Forth environment that enables the user to access several applications at the stroke of a word.

The source code for F83X, and the relocation program and its source, are available for downloading from the Forth RoundTable on GEnie, or by mail on 8" SSSD CP/M, 5 1/4" DSDD MS-DOS, or 3 1/2" SS HP CP/M-68K format. To get a copy, send a sufficient number of disks (8" and 3 1/2" require four diskettes; 5 1/4" requires three) and a stamped, self-addressed envelope with $10 for handling to: Robert J. Eager, 3500 NW Glenridge Pl., Corvallis, Oregon 97330.

**Recommended References**


The author currently works as a software engineer at Wright Patterson Air Force base.
I volunteer

Please direct all questions, comments, and

miscellaneous personal abuse to:

Henry Lassen or Michael Perry

1259 Coral Way

Berkeley, California

94704

...
EDUCATING
FORTH USERS

BILL KIBLER - SACRAMENTO, CALIFORNIA

Over the past few years I have been studying and trying to use Forth. I still remember my first experiences with the language, and the frustration. Although I had read all the material with the program, and had picked up a book, I was amazed with how easily I got lost. I had learned enough to consider Forth a language for me, but was finding it a bit too cryptic to start using.

Time has passed, and I have learned more about using Forth. I have most of the books published on the subject, yet something still seems to be missing. While earning a masters degree on Computers and Education, I have been able to understand some of Forth’s problems. It is not the lack of a good language or means of expressing Forth’s operations, but a matter of providing educational support for users, especially new users.

Most of us have used Turbo Pascal and have seen the reason for its success: speed of compilation. Turbo’s editing and compiling can be considered a little like Forth’s screen compilation. Looking at that, I wonder why Forth hasn’t achieved some of Turbo’s success. The answer was not internal speed, but educational support through a free spreadsheet and an inexpensive tutor program. The free program gave the user a program to run immediately, as well as the source code from which to steal and gleam ideas and techniques.

What I propose, then, is more fuel for standardization of Forth. This standardization is not of words but of packaging. No version of Forth should be considered complete without a free, bundled program and enough tutorial information to get a new user on line the same day. To that end, I believe I have written just such a program: TUTOR.BLK.

TUTOR.BLK came about as my masters project, and was a two-sided program: it was a more detailed look at Forth (by explaining the language to non-users), and I was creating tutorial information about a programming language that I felt was not getting the exposure it needed. The program screens contain enough information about Forth for a new user to start using the program the same day. The new words defined also show the ease with which new words are created and old words modified to suit special needs.

The program screens included here cover introductions and new words to manipulate the tutorial program. The entire program is over 100 screens, but only the first ten are program screens. There are 90 screens of text, including some blank screens, as well as a glossary at the beginning of each section or chapter. I would add the first ten screens into my version of Forth; the user would then see a list of help words and program information at boot-up time. This provides a directed entry into the program, and guarantees the first-time user a positive experience.

This program was written for F83, the public-domain, Forth-83 program. Although Laxen and Perry have done a fine job, their support for first-time users is rather minimal. This program, along with Starting Forth by Leo Brodie, which the tutorial material supports (via its chapter references), should help F83 users get started. The TUTOR.BLK program is public domain, and can be included with commercial packages (as long as I get credit). Commercial users will want to customize their own program to highlight particular features.

This program is by no means complete. I had intended to make a glossary section, too; the glossary would give the user either a short or long definition of most words. For beginners and novice users, the ability to have on-line explanations has proven to be many a program’s successful sales strategy. Forth developers can use that same strategy, and with less overhead.
SCR #0 TUTOR.BLK
(INTRO TEXT FOR SCREEN ZERO
BDK112186)
******************************************************************************
***** F83 TUTOR AND HELP PROGRAM
*****
***** Written by Bill Kibler
***** PO BOX 487 Cedarville, CA 96104
*****
***** Donated into PUBLIC DOMAIN, with
***** ALL Commercial rights reserved
******************************************************************************

SCR #1 TUTOR.BLK
(LOAD BLOCK AND START OF TUTOR PROGRAM)

53 load 23 tree 15 spaces
.(PLEASE WAIT WHILE LOADING TUTOR SCREENS .TUTOR.BLK )
CR CR CR CR CR

(variables and display routines)
VARIABLE ETUTOR
(END DISPLAYING TUTOR SCREENS)
VARIABLE STUTOR
-BEGINNING SCREEN OF CURRENT GROUP)
VARIABLE NTUTOR
(NEXT TUTOR SCREEN 0 GROUP)

: L$$K DUP 36 = IF 1 ETUTOR! THEN ;
(CHECK FOR $$)

: DISPLAY (DISPLAY SCREEN OF TEXT)
1 ?ENOUGH DUP SCR! L/SCR 1 DO DO 5 SPACES
DUP BLOCK I C/L * + C/L TUCK PAD SWAP CMOVE PAD SWAP
(>TYPE WITHOUT THE TYPE)
0 ?DO DUP C@ L$$K EMIT 1+ LOOP DROP
(TYPE WITH L$$K)
CR KEY? ?LEAVE LOOP DROP ;

SCR #2 TUTOR.BLK
(go get screens of information - gotutor tutor)

: WTPRT ." CURRENT SCREEN IS " SCR ? 2 SPACES
." ESC = EXIT" 2 SPACES
." USE SPACE BAR FOR NEXT SCREEN"
;

: ESCCHK DUP 27 = IF 1 ETUTOR! 32 THEN ;
.SET ESC FLAG

: WAIT WTPRT 13 EMIT

(PRINT THEN CR WITHOUT LF)
BEGIN KEY ESCCHK 32 = UNTIL ;
(LOOP TIL SPACE KEY)

: GOTUTOR
(DISPLAYS SCREEN ON STACK THEN WAITS )
CR DUP SCR! 15 SPACES .SCR CR
BEGIN DISPLAY WAIT NTUTOR @ 1 + DUP
DUP NTUTOR! 1 ETUTOR @ = UNTIL CR
CR 3 SPACES
." REPT = REPEAT LAST LESSON ... GET = NEXT LESSON"
." MENU = MENU " CR CR CR ;

SCR # 3 TUTOR.BLK
(INITIALIZE AND START THE LOOPS..GET..REPT...)

: GET (GO GET NEXT GROUP OF SCREENS )
NTUTOR @ TUTOR ;

: REPT
(GO BACK AND REPEAT SET OF SCREENS)
STUTOR @ TUTOR ;

: START-TUTOR
(START WITH FIRST SCREEN OF TUTOR)
10 TUTOR ;

: HELP
(GIVE INTRO MESSAGE)
10 TUTOR ;

SCR #4 TUTOR.BLK
(DEFINING MODULES OF INFORMATION...)

: INTRO 10 TUTOR ;
: CHP1 12 TUTOR ;
: CHP2 18 TUTOR ;
: CHP3 25 TUTOR ;
: CHP4 34 TUTOR ;
: CHP5 40 TUTOR ;
: CHP6 48 TUTOR ;
: CHP7 55 TUTOR ;
: CHP8 66 TUTOR ;
: CHP9 73 TUTOR ;
: CHP10 84 TUTOR ;
: CHP11 92 TUTOR ;

Forth Dimensions 28
This program will help beginners and past Forth users alike. The screens contain information on Forth-83 and are related to the book "STARTING FORTH" by Leo Brodie, which should be used as a textbook with this program. Each chapter or series of screens is organized to present the words used in the chapter in a glossary form. Forth users will find this glossary important to see the differences between F83 and other versions. Typing HELP will repeat these screens, then type:

NEW F83 WORDS

The following words are important utilities in F83 and may be different from previous versions. WORDS will display a list of F83 words used. OPEN allows use of an existing file, 10 MORE is used to add 10 screens, and 30 CREATE-FILE NAME.BLK (opens 30 screens). INDEX displays a list of line 0. 1 20 INDEX will list screens 1 to 20. 1 30 SHOW will print 6 screens to a page on your printer in condensed mode (use: 'EPSON IS INIT-PR for epson printers). 1 30 TRIAD prints three to a page if condensed print is not available. 1 30 SHADOW SHOW will print both the

regular screens and the information screens on a page (not used in TUTOR but in UTILITY.BLK). SEE xxxx disassembles the word xxxx, while VIEW will open the source file (on A: drive) and list the screen it is in. VOCS will list the vocabularies in the dictionary, while ORDER displays the path of the directory search. Use DOS WORDS to see a list of the DOS dictionary words. CAPACITY will print the number of screens in a open file. A L will toggle between the shadow and the source screens. N L will display the next screen, L will list current screen, B L will list previous screen. 1 EDIT will invoke the line editor with screen 1 ready to edit. 0 NEW will start editing at line 0 and allow the text to be entered one line after the other. HEX 100 80 DUMP will do a hex dump of memory location 100h to 180h. DEBUG LIST will allow stepping through list when used next as in 1 LIST. Use BYE to exit to DOS.

GET = next chapter
REPT = begin chapter again
HELP = repeat these screens
START-TUTOR = start at INTRO
SPACE BAR = next screen
ESC = stops display
BYE = EXITS to DOS
MENU = displays this screen
PRTSCR = displays this screen

Scr #5 TUTOR.BLK
(MORE ROOM FOR LESSON WORDS....)

: MENU 9 DISPLAY ;
( will display infor screen )

: PRTSCR CR ." CURRENT GET SCREEN IS "
NTUTOR @ .
CR ." REPT SCREEN OF INFORMATION IS "
STUTOR @ . CR ;

HELP

Scr #6 TUTOR.BLK
(PRINT SCREENS FOR TUTOR INFORMATION....)

FORTH-83 TUTOR PROGRAM AND HELP SCREENS
WRITTEN BY BILL KIBLER
(c) 1987
DONATED into PUBLIC DOMAIN, with
ALL COMMERCIAL RIGHTS RESERVED

Scr #7 TUTOR.BLK
(second intro screen with list of words....)

the chapter number for the area of help needed.
Typing ESC key will exit the screens and return
to the system prompt. GET will display next
chapter of information, while REPT will start
with the first screen of the chapter again.
START-TUTOR will start with the introduction
chapter.

NEW F83 WORDS

The following words are important utilities
in F83 and may be different from previous ver-
sions. WORDS will display a list of F83 words
used. OPEN allows use of an existing file, 10
MORE is used to add 10 screens, and 30 CREATE-
FILE NAME.BLK (opens 30 screens). INDEX displays
a list of line 0. 1 20 INDEX will list screens 1
to 20. 1 30 SHOW will print 6 screens to a page
on your printer in condensed mode ( use: 'EPSON IS INIT-PR for epson printers). 1 30 TRIAD
prints three to a page if condensed print is not
available. 1 30 SHADOW SHOW will print both the

Scr #8 TUTOR.BLK
(THIRD PRINT SCREEN OF TUTOR INFORMATION....)

TUTOR WORDS

INTRO = introduction
CHP1 = fundamentals
CHP2 = RPN and STACK
CHP3 = editor commands
CHP4 = conditionals,nests
CHP5 = fixed point operations
CHP6 = loops ( & nested)
CHP7 = number types
CHP8 = var. const. arrays
CHP9 = F83 structure
CHP10 = Input/Output
CHP11 = extensions

GET = next chapter
REPT = begin chapter again
HELP = repeat these screens
START-TUTOR = start at INTRO
SPACE BAR = next screen
ESC = stops display
BYE = EXITS to DOS
MENU = displays this screen
PRTSCR = displays this screen

1988 Rochester Forth Conference On Programming Environments

June 14 - 18, 1988
University of Rochester

CALL FOR PAPERS

The Eighth Rochester Forth Conference will be held at the University of Rochester and is sponsored by the Institute for Applied Forth Research, Inc.

The focus will be on Programming Environments. The invited speakers include Cliff Click and Paul Snow on their Postscript implementation in Fifth and William Wickes, software project leader for the HP28 calculator discussing Reverse Polish Lisp. Other speakers will discuss environments for scientific calculation, simulation and programming workbenches.

There is a call for papers on topics in the following areas:

**Environments**
- Object-oriented Forth
- Forth as an AI platform
- Postscript
- Reverse Polish Lisp
- Workstations
- Simulation systems
- Business and Scientific languages
- Threaded compilers for Basic and C

**Technology**
- Forth processors
- Peripheral controllers
- State machines
- Metacompliers
- Forth in VLSI

**Applications**
- Laboratory, space-based, medical, AI, real-time, business, database, financial

**Dialects**
- ACTOR, ASYST, Fifth, MAGIC/L, NEON, Saavy, PLOG, RPL, SPHERE, STOIC

Papers may be presented in either platform or poster sessions. Please submit a 200 word abstract by May 15th. Papers should be received by June 1st, and are limited to a maximum of four single spaced, camera-ready pages.

Longer papers may be presented at the Conference but should be submitted to the refereed Journal of Forth Application and Research. Abstracts and papers should be sent to the Conference Chairman: Lawrence P. Forsley, Institute for Applied Forth Research, Inc. 70 Elmwood Avenue, Rochester, New York 14611. For more information please write the Conference Chairman or call (716) 328-6426.
Martin Tracy has been intimately involved in the Forth community for years, bringing many contributions in his roles as Forth vendor, leader, expert programmer, and current member of the Forth Interest Group's Board of Directors. Mike Ham interviewed Martin for Forth Dimensions and got frank talk about Forth and FIG, and some quick glimpses into Martin's eclectic life.

MH: Are you still working for Forth, Inc.?

MT: That’s right: I’m a senior programmer at Forth, Inc. For the past year, I implemented the digital-signal-processing Forth for the Texas Instruments TMS 320-22. We’re selling that, and I’m working on other projects as they come up.

MH: What machines?

MT: Quite a few people want us to write software for them on the IBM, usually the AT computer. But it varies. We do quite a bit of work on the 68000. Process control people are turning to ruggedized IBM PCs; I wouldn’t say it’s quite a machine of choice yet, but it’s getting close to it.

MH: How did you get into Forth?

MT: I first encountered Forth working on programming a myoelectric artificial arm for a below-the-elbow amputee, which means there’s enough of a stump that you can fit the arm over the stump and still make contact with the remnants of the muscles. You teach the machine every morning when you put it on, by concentrating on an action — that activates the remnants of the muscle. The computer watches and learns what your intention is, and then moves the artificial arm the same way. It gives some crude control over the arm but, of course, there is no feeling.

The processor in the arm was an RCA 1802. It was programmed on a Decus Forth development system, so I started reading the Decus Forth manual. It was incomprehensible to me, and I gave up on Forth at that point. I stayed away from it for perhaps a year or two. Instead, I wrote a tiny Pascal compiler, which was sold through Programma International for several years. I started looking for the ideal programming language, which to me meant portability. That is, I would be able to move my tools to the different laboratory computers I was working with. Our laboratory desktop computer spoke only BASIC, our laboratory minicomputer spoke only Fortran, the statistical packages I worked with spoke only APL, and I was somewhat miffed by having to translate the tools around.

MH: What was your job when you were working on the arm?

MT: I was a full-time lecturer in the dance department at UCLA, teaching anatomy for dancers while completing a Ph.D. in bioengineering.

MH: And the arm was part of the bioengineering?

MT: Yes, in the UCLA Bioengineering Laboratories.

MH: So you stayed away from Forth and developed your design for the ideal language.

MT: It was basically a macro-assembler. Phil Wasson pointed out that I was developing a language very much like fig-FORTH. I looked at fig-FORTH and thought I could implement that model on an Apple computer. It was already on an Apple, but it wasn’t in the form I needed. I thought it would take me a month. Phil was a programmer with Programma International at that time, working with their version of Forth. And in fact I was able to convert the Forth in a short time, so we decided to form a company, MicroMotion, and sell Forth for the Apple computer. That’s how I first got into Forth.

At this time I was reaching my seven-year limit as a lecturer at UCLA. UCLA does not encourage lecturers to remain after seven years. They were interested mostly in tenure track positions. So I left the UCLA dance department and started Micromotion with the hope that the company be able to run itself when I periodically left to dance, which is what I did for the next several years.
MH: What kind of dance do you do?

MT: Classical ballet and character; I've retired.

MH: What's "character"?

MT: Character is what ugly ballet dancers used to do. It's where Drosslemeyer comes in, or Rothbart, the evil magician, Puss in Boots, the Bluebird, the Spanish dancer.

MH: So you set up the company to cover while you were not there...

MT: Yes, I wanted very much to get into computers with the Forth language, but I could only dance while I was young. So I chose to give dance priority. Of course, in a fast-moving technical field that wasn't a very good choice if your goal is to make a lot of money, but it was the right choice in that I did get to dance. Linda Kahn ran the company when I was away, and that's how MicroMotion got its start.

MH: Whence the MicroMotion logo of the little dance figure...

MT: That's right.

MH: Did you dance mostly on the West Coast?

MT: I danced mostly in the Orient, in Japan and Taiwan. I did dance a bit in Texas and New York City. I danced with the American Festival Ballet, Radio City Music Hall, West Side Story tours...

MH: Did you ever have any direct overlap of your Forth experience and dance? I'm thinking of Labanotation, for instance. [Labanotation, or the Laban system, is a somewhat reconsidered system for writing down the movements of a dance. —ed.]

MT: Yes; in fact I'm a Labanotation instructor. I taught that for several years. My master's thesis was a computer-assisted, movement-notation system.

MH: Based on Labanotation?

MT: I did bring Labanotation into it, but it turns out that the muscles and bones can be modeled fairly accurately mathematically, and even Labanotation has at its heart a model of the body easily transposed into computer terms. In fact, I held a panel on computer dance, around 1975 in Philadelphia, part of the conference for the Committee on Research in Dance. But that was peripheral to my interest; it turned out that I really wasn't interested in combining computers and dance.

MH: When you set up MicroMotion, you quickly released other versions of your Forth.

MT: MicroMotion still exists as a company, it's just that I am no longer associated with it. After the Apple, we went to the Z-80, then the Commodore 64, then the IBM PC, then Ray Talbot produced a Macintosh MasterForth for us.

MH: During that time, you wrote the introductory Forth text Mastering Forth.

MT: That's right. The very first version was a yellow book, when MicroMotion was still Forth-79. We produced what I believe is the first tutorial in Forth. Brodie's Starting Forth came out a year and a half after that little yellow book came out. The version known as Mastering Forth came out two years after Starting Forth. I am currently doing a second edition of Mastering Forth for Brady Books (Prentice-Hall).

MH: Is it still tuned to MasterForth?

MT: It's expanded. It has the same material — though revised — as Mastering Forth, but with chapters on topics that I feel have not been covered in Forth books, such as target compilation, graphics, and floating point.

MH: Will it be published in 1988?

MT: I'm committed to producing the book by the end of the March.

MH: How did you decide to leave MicroMotion?

MT: My fortieth birthday was approaching. I had decided to retire from dance, and also to purchase a home. To do that in southern California takes a lot of money, so I sold MicroMotion and started to work for Forth, Inc.

MH: When you worked in bioengineering, you already had programming knowledge. How did you get started in programming itself?

MT: The thing that has interested me for a long time is human beings in motion, people when they move. Part of this is my background in dance; part of it Labanotation and Effort-Shape, and other forms of movement notation.

MH: What is "Effort-Shape"?

MT: The quality of movement, as opposed to where the limbs go. For example, are you "bursting" — that is, are you letting one muscle carry the action without inhibiting it with the antagonistic muscle? I've taught several forms of movement notation, and I've also taught some aspects of nonverbal communication and anatomy for dancers.

MH: What was anatomy for dancers?

MT: It's the owners manual and operating guide for the body: what you're designed to do, what is a violation of that, how things work, how you keep them working. Towards the end, I tailored it specifically for dancers and martial artists. Dancing is primarily a world of women, and martial arts primarily a world of men, but the bodies are very much alike, so with the combination I could attract a fairly large number of students interested in either of those two.

MH: And they both have to know how to move.

MT: Yes, and the contrast is just as interesting as the similarity.

MH: Do you have a background in martial arts?

MT: I have had seventeen years of various Chinese styles.
MH: How did dance enter your life? We don’t have a culture that directs people toward dance, by and large.

MT: When I was around 15, a girl friend asked me to come and help her out in her dance recital. I knew nothing about it, but I went and helped her by dancing a bit and moving with her, and the director of the American Festival Ballet saw me there and indicated that I had some promise. So I started taking classes and within a year was doing my first professional work — not unusual for a male dancer. It’s unusual for a woman, but not for a man.

MH: Where were you then?

MT: I was in Providence, Rhode Island, did most of my dancing in New York City. From then on I’ve always been a dancer and something else — the “something else” changes from time to time.

So at UCLA, after I developed my course and started taking classes in kinesiology, the study of motion in the body. Eventually, as a teaching assistant, I taught lectures and classes in electromyography and biomechanics for the kinesiology department. To work with the body at that level, you do need to learn something about computers and mathematics. For instance, suppose I need to know what’s going on inside the hip. Let’s say I’m interested in why dancers who dance beyond the age of 30 often develop hip arthritis and may even need hip replacement. Well, I can’t put a transducer in the hip of a dancer. The only thing I can do is take a high-speed film of the outside, model the forces, and deduce what is going on in the hip to make that happen. So I need rather sophisticated tools immediately: high-speed photography, mechanics and mathematical modeling of moving objects, anatomy — all tools needed to solve that problem. When I had learned what I could from the kinesiology department, I moved on to the bioengineering department to learn more, and the more I focused on that, the closer I got to computers and had to learn more about that. And that’s how I got into programming.

MH: Do you see any body-movement projects at Forth, Inc.?
Forth Dimensions

MT: I don’t see anything like that coming through the door. The closest we get is robotics, but they lack the human nature: robots are too predictable for my taste.

MH: Do you mean too few degrees of freedom in the movement, or the lack of will?

MT: The lack of will. For instance, when you’re embarrassed you move very differently. But I don’t know how to embarrass a robot or a neural net.

MH: I would like to get your thoughts on FIG: where it’s come from and where it’s going.

MT: I think it has come from a hobbyist orientation. I would like to see it move to more of a professional-support organization. I don’t know whether it can or not, I think we’re trying it on right now to see.

MH: What kinds of activities do you see for a professional support organization? Is there a model for it, or is it something we have to create as we go?

MT: The model I have in mind is somewhere between a professional society and a public relations firm. Certainly one of the things FIG can do for Forth is to promote name recognition of the Forth language. To me, it seems strange that there is an organization for Forth. To me, a language is always the thing that gets in the way between you and the problem; some get in the way less than others. What I like about Forth is that it doesn’t get very much in the way of solving the problem. But it does get in the way. I mean, if I want to measure the temperature of a lamp and you tell me I have to learn to type, that’s a skill that I don’t need.

To me the language is interesting in its ability to let me solve my problem and leave me in peace as I do so. One of the things that attracts me to Forth is that it lets me do so. But I think if I were a manager and you said, “Come join the Forth Interest Group,” or “Come see Forth at the Forth convention,” I would ask, “Why?”

But if you said, “Come and see a good solution to solving real-world, real-time programs on existing hardware at the Forth convention,” or “Join the Forth Interest Group to learn more about solving such problems,” that would be appealing.

MH: So, in a sense, we’re handicapping ourselves by having a convention at which the theme seems to be Forth rather than solutions to particular problems.

MT: Yes, I think we should concentrate on what we’re good at. I really believe Forth is the best language for certain classes of problems. Unlike some of my peers, I do not believe it is good for everything.

MH: The Rochester group has been very successful by making their conference theme a particular problem — AI, robotics, and the like — and then getting people to attend who are interested in that problem. They inevitably get a lot of exposure to Forth, but they are drawn by the problem and its solutions. Do you see something like that as a possibility for future FIG conventions? A targeted problem area, perhaps?

MT: Yes. I have volunteered to run the next FIG convention, which will be in Anaheim. I would like to use exactly this line of argument to bring people in from the aero-space corridor. I want to reach people who don’t already use Forth. I will do more than that: I want to reach people for whom, if I said, “Come learn Forth,” it would be a strange request. I’m going to reach them by saying, “Come see working neural nets at the Forth convention,” or “Come see RISC or WISC language oriented processors at the Forth convention.” I want to give them a reason, with the Forth in small letters rather than capitals. I want something visible, audible — real-world, real-time problems. And that’s great, because those are fun to watch. But if Forth is that kind of language, why wave a book at me, or a piece of paper or a theory?

MH: Or a case statement?

MT: Right. Don Colburn had a wonderful idea, and I’m going to try to make it happen: a programming contest with at least a $1000 cash prize. I will arrange gizmos or widgets for each contestant, the same for each contestant. We will provide a room and tables and power. The contestants will bring anything to it: any computer, any software. They can bring a team if they want, whatever it takes to do the problem. And when the gun is fired, they solve the problem.

It will be a fun problem to watch. You get to see this happening, and once it’s solved, we will leave it running, so people can come by and watch it. I’m going to challenge Microsoft Quick BASIC, Turbo C, and others. I’m going to challenge them all to come.

MH: You have a pattern in which you do things on your own. Forth has grown up in an environment in which many programmers work on their own, but at Forth, Inc. you have a cluster of Forth programmers and they do team projects. Do you have any thoughts about Forth in a team atmosphere?

MT: One is that I think it needs to be managed differently than other languages. You break up large tasks differently. I don’t believe a simple Forth (without local variables or other tools), is very good at large projects, despite the fact that there have been many large projects done with Forth. I think the first thing that happens is that the Forth is extended in some way so that you can manage the large project, and then you work with that extension. But whether you are in Forth itself...

Here’s an example I often give: you write a C in Forth and now you write a program — are you writing a program in C or in Forth? As far as I am concerned, you are programming in C. It looks like C, acts like C. So the fact that Forth can do anything is a kind of cop-out; the real interesting question to me is what does it do naturally, as Forth, and not what you can bend it to be.

MH: So, as a natural thing, you see Forth as a one-person language, and for a large team you build a language suited to the task-and-team approach, with local variables and the like. Then it’s not Forth anymore.

MT: Right, but Forth, Inc. would not agree with me. One approach we take there is to break the problem into tasks that can be done at the same time, run at the same time. Programmers work on different tasks, then they are put together and run at the same
time to make the system.

MH: With lots of use of vocabularies to avoid collisions...

MT: No, actually we use vocabularies very little. We run tools at integration to detect name conflicts and change them.

MH: You’ve done a lot of Forth programming. Do you have any particular favorite, anything you’ve done in Forth that you like the best?

MT: Well, the LISP extensions I did for the Forth Model Library were quite interesting; the ones on Volume I of the library.

MH: You have a good ability to lay out an interesting and reasonably sized problem, and then do it completely.

MT: I have a definite sequence I go through when I solve a problem. The first thing I do is immersion. I get together everything I possibly can gather in a short amount of time. For example, searches of the Byte network BIX, trips to the library if I can get books — mostly books, in fact: my preferred source is books.

I collect as much information on the topic as I can, and read quite a bit of it without understanding very much of it. I’m just bringing the material in. Then I will let a little time go by, half a day or a day, when I am not concentrating on the problem. And then I’ll start to work on the problem at that point.

FUTURE
announces
Eight new products based on the NC4016

Future Series products:
CPU board (available 2nd quarter 1988)
• NC4016 (5 MHz standard)
• Stack and data RAM
• Full 128K bytes of paged main memory
• Power fail detect
• Automatic switching to on board battery backup at power fail
• Psuedo-serial port - full compatibility with CM-FORTH and SC-FORTH
• 16K bytes of EPROM (SC-FORTH, SC-C and CM-FORTH available)

Display/Debugger board (available 2nd quarter 1988)
useful for testing and debugging custom hardware
• Provides hexadecimal display of the data, address, and B-port
• Indicates status of reset, interrupt, WEB, WED, and X-port
• Provides for free running and single step clocking
• Provides the ability to independently drive (write to) the data, address, and B-port directly with user data

I/O board (available 2nd quarter 1988)
for serial communication, interrupt handling, event timing, time and date logging and saving system state parameters
• Two RS232 serial ports
• Eight level prioritized interrupt controller. Each interrupt line is individually maskable and resettable. Current pending interrupt status is readable.
• Real time clock with 2K of non-volatile RAM
• Three 16-bit timer/counters

Extended Memory board (available 3rd quarter 1988)
• Paged memory — 64 Kbytes segments, up to eight segments

Card Cage & Power Supply (available 3rd quarter 1988)
• Rack mountable card cage with face plates for each slot
• ±5 volts and ±12 volts supplied
• 72 Pin backplane

Disk Drive Controller board (available 3rd quarter 1988)
• 3-1/2 inch floppy and SCSI controllers (for hard disks)

Video board (available 4th quarter 1988)
• Will drive Apple Macintosh II high resolution (640 x 480) monochrome monitor and PC compatible monochrome monitors

A/D & D/A board (available 4th quarter 1988)
• 12 bit, 1 MHz A/D & D/A converters

Future, Inc. P.O. Box 10386 Blacksburg, VA 24062-0386 (703) 552 - 1347

Apple is a registered trademark of Apple Computer, Inc. Macintosh is a trademark of Apple Computer, Inc. SC-FORTH and SC-C are products of Silico Computer.
At last November’s National Forth Convention in San Jose, California, the Forth Interest Group celebrated its tenth anniversary. Mr. Charles Moore, the creator of Forth, contributed to the event in many ways, among them his annual “Fireside Chat” with attendees. Here, Scott Squires shares the notes he took as he listened to the informal session.

As usual, Chuck was full of unusual ideas, mixed with tongue-in-cheek, during his annual “fireside chat” at the 1987 Forth National Convention. I have tried to record these as accurately as possible, and hope that at least the concepts are correct.

There are two attitudes about Forth in the Forth community:
1. It’s about to die.
2. It’s all set to take off.

Chuck didn’t know which is true, but didn’t actually think it matters. He uses Forth; maybe it would be more useful if other people didn’t. (Chuck smiles.) He can’t conceive of a successful SDI (‘Star Wars’), given the complexities. It’s impossible to check out, and a problem could kill everyone. Ben Bova has written a book, Millenium, that covers a lot of this.

He hasn’t seen any new, compelling reasons that persuade people to use Forth. He and the Forth community have been providing reasons for a long time. He’s not sure any longer about what Forth is. Originally, he created it as an interface to the computer, so he could solve problems. Now he wants it as an interface to the problem, with the computer just being an incidental. He could make a new computer fairly easily now, so that’s almost as flexible as the software. This alters the tradeoffs profoundly.

With his new, three-key keyboard, Chuck has come up with some new ideas, some of them in the last few days. Forth doesn’t need an interpreter or compiler — it’s possible to use just a decompiler. To him, a disk is just a non-volatile backup of the object code. There would be no blocks or buffers; these are things he had always thought were a part of Forth. Now he’s busy removing more and more of Forth, and isn’t sure of what will be left. Somebody suggested it might be like the smile of the Cheshire cat from Alice in Wonderland. Chuck thinks it might be the illusion of Forth. He’s not worried about conflicts, as long as it’s fun.

“The difficulty with neural nets is training them.”

We should figure out how programmers will be doing it in 1000 years and start doing that now. Most people think there won’t be any programmers in 1000 years. He doesn’t think that’s true, especially since he’s heard the same thing for the past 20 years. A programmer is the one who understands the problem, not necessarily the one who does the coding.

Some people think the computers of the future will be neural nets. The most difficult thing about neural nets is training them, not programming them. You need to spend time coxing these machines, when you really want to just tell them what to do. People are going to want loyal and faithful machines — slaves, if you will, that do exactly what they are told. You never really know what a machine has learned. You can’t trust a machine like that. It’s thought that programming tools will be so powerful that programmers won’t be needed. That isn’t true.

It’s easier to write in Forth than in other languages, but not a magnitude easier. Instead of being difficult, as with other languages, it makes it possible. Computers will be put to more complex tasks in the future.

What will a programmer be doing 1000 years from now? What kind of interface? Probably brain waves. How many parallel channels? Well, it would be controlling a very high-resolution display with full 3D color and sound. Several channels would be modulating, but there would probably be three main channels. This makes it close to the three-key keyboard on his latest system. (Everyone laughs.) The programmer would be laying down — no, make that floating. Yeah, that’s it. (Chuck smiles to himself at the thought.) Now, will this programmer be dealing with files or screens? (Audience laughs.)

You won’t need to deal with source code — this notion just came to him in the last few days. You’ll just do a memory dump or decompile to see the code. Source code is bulky. In the past, he resisted saving the object code because he couldn’t see maintaining both object and source code. That would have been redundant. Instead, the source code was recompiled very quickly each time it was needed. He had completely overlooked the opposite idea of saving just the object code.
Chuck has never found a pretty printer he liked. They always seem to format the code differently than he would. His source-code format is inconsistent. Sometimes he wants an IF at the start of a line, sometimes at the end; or he wants something spaced differently. “Of course I’m always right,” he laughs. It will never decompile and indent as he’d like, but now that most of his definitions are only one line long, indenting doesn’t matter.

One feature of blocks is that it allows a specific grouping of words. Decompiling can’t do that. Typically, though, you’ll probably only need to decompile one word at a time.

Comments and stack effects won’t be in the object code, but they are necessary. He’ll probably put these in shadow blocks on the disk. Every word could have a pointer to related comments on the disk.

You would be able to walk up to any computer and see what program is running and how it works.

1. It doesn’t matter what computer it is; the process is the same, if there is a smart decompiler.
2. It doesn’t matter how it got there. If it were done in C, it would still decompile to Forth.
3. Forth could unify the representation of the computer.
4. You can look at a program even if the supplier hasn’t given you access. The concept of “proprietary” would have a new meaning. The Forth community is a bit like a terrorist group. Maybe each person could decompile a program. (More laughs.)
5. You could change a program while it’s running. His new machine writes directly to the CRT. It has a variable for the number of pixels per line and a variable for the number of lines. If he changes these while the program is running, he now has to go back to the source code and change it there. If there were no source code, decompiling would always show the latest version with the correct information.

Changes to a program would probably make it larger. To make changes, you might have to relocate words or remove words in the middle, thereby leaving holes. But most debugging is done at the end of the dictionary, so this may not be a problem.

The Novix was the first CPU for which Chuck seriously tried to write a full compiler. Forth and the Novix chip are not as ideal together as he’d like. To truly optimize, you need to look back three or four words. DUP is a prefix in Forth, but on the Novix it’s a suffix. All these problems go away with the compiler. Because you’re writing true, in-line Forth, all changes could be optimized; and that leads to more compact code.

To go from one machine to another, you would decompile the object code to produce the source code. The target machine would compile this source code in its own format. This is similar to the idea of metacompiling, but implies that compiling is only needed when moving between machines.

Someone from the audience mentions that RTL (a Forth-variant language) has flags in the object code to tell what type of data structure it is (i.e., IF, WHILE, BEGIN, etc.) Chuck thought that Wil Baden’s diagramming system, presented at last year’s FORML, might be used as part of the decompiling. This is a “pleasant flowchart,” where it doesn’t matter exactly what word is used to generate the structure, as long as the result of the structure is clear.

Other languages could be decompiled to Forth, and perhaps it could optimize the decompilation to produce good Forth code, not just a step-by-step decompilation of programs written without Forth in mind.

He would be willing to change his programming style to conform to the tools. At one time he pushed for the ‘[‘] word. Since the Novix, he hasn’t used it at all, and says that he’s changed mind.

His objection to the mouse is the coordination required.

About Chuck’s three-key keypad:

1. Color-coded keys (red, green, blue). Selects the word or item with that color.
2. It provides a limit or bound, so there isn’t any need to check for limits.
3. Seven choices are possible. Seven items is the limit the brain can store and refer to at one time.
4. A key always points to a Forth word.

Chuck has started using menus in his system. The menus started out as a tree structure, but that was restrictive. Now he uses cross-referencing and a web structure. Any menu can point to any other menu. You can go back all the way, anytime, because this uses the Novix chip (which has a circular return stack).
FIG CHAPTERS

USA.

* ALABAMA
  Huntsville FIG Chapter
  Tom Konantz (205) 881-6483

* ALASKA
  Kodiak Area Chapter
  Horace Simmons (907) 486-5049

* ARIZONA
  Phoenix Chapter
  4th Thurs., 7:30 p.m.
  Dennis L. Wilson (602) 956-7578
  Tucson Chapter
  2nd & 4th Sun., 2 p.m.
  Flexible Hybrid Systems
  2030 E. Broadway #206
  John C. Mead (602) 323-9763

* ARKANSAS
  Central Arkansas Chapter
  Little Rock
  2nd Sat., 2 p.m. & 4th Wed., 7 p.m.
  Jungkind Photo, 12th & Main
  Gary Smith (501) 227-7817

* CALIFORNIA
  Los Angeles Chapter
  4th Sat., 10 a.m.
  Hawthorne Public Library
  12700 S. Grevilleas Ave.
  Phillip Watson (213) 649-1428
  Monterey/Salinas Chapter
  Bud Devins (408) 633-3253
  Orange County Chapter
  4th Wed., 7 p.m.
  Fullerton Savings
  Huntington Beach
  Noshir Jessung (714) 842-3032
  San Diego Chapter
  Thursdays, 12 noon
  Guy Kelly (619) 450-0553
  Sacramento Chapter
  4th Wed., 7 p.m.
  1798-59th St., Room A
  Tom Ghormley (916) 444-7774
  Silicon Valley Chapter
  4th Sat., 10 a.m.
  H-P, Cupertino
  George Shaw (415) 276-5953
  Stockton Chapter
  Doug Dillon (209) 931-2448

* COLORADO
  Denver Chapter
  1st Mon., 7 p.m.
  Clifford King
  (303) 693-3413
  CONNECTICUT
  Central Connecticut Chapter
  Charles Krajewski (203) 344-9996

* FLORIDA
  Orlando Chapter
  Every other Wed., 8 p.m.
  Herman B. Gibson (305) 855-4790
  Southeast Florida Chapter
  Coconut Grove area
  John Forberg (305) 252-0108
  Tampa Bay Chapter
  1st Wed., 7:30 p.m.
  Terry Mcnay (813) 725-1245

* GEORGIA
  Atlanta Chapter
  3rd Tues., 6:30 p.m.
  Western Sizzlin', Doraville
  Nick Hennenfest (404) 393-3010

* ILLINOIS
  Cache Forth Chapter
  Oak Park
  Clyde W. Phillips, Jr.
  (312) 386-3147
  Central Illinois Chapter
  Urbana
  Sidney Bowhill (217) 333-4150
  Rockwell Chicago Chapter
  Gerard Kusiedek (312) 885-8092

* INDIANA
  Central Indiana Chapter
  3rd Sat., 10 a.m.
  John Glesby (317) 353-3929
  Fort Wayne Chapter
  2nd Tues., 7 p.m.
  IP Univ. Campus, R71 Neff Hall
  Blair MacDermid (219) 749-2042

* IOWA
  Iowa City Chapter
  4th Tues.
  Engineering Bldg., Rm. 2128
  University of Iowa
  Robert Benedict (319) 337-7853

Central Iowa FIG Chapter
1st Tues., 7:30 p.m.
Iowa State Univ., 214 Comp. Sci.
Rodrick Eldridge (515) 294-5659
Fairfield FIG Chapter
3rd Wed., 7 p.m.
Gurly Leete (515) 472-7077

* KANSAS
  Wichita Chapter (FIGPAC)
  3rd Wed., 7 p.m.
  Wilbur E. Walker Co.,
  532 Market
  Ame Flores (316) 267-8852

* MASSACHUSETTS
  Boston Chapter
  3rd Wed., 7 p.m.
  Honeywell
  300 Concord, Billerica
  Gary Chanson (617) 527-7206

* MICHIGAN
  Detroit/Ann Arbor area
  4th Thurs.
  Tom Chrapkiewicz (313) 322-7862

* MINNESOTA
  MNFIG Chapter
  Minneapolis
  Even Month, 1st Mon., 7:30 p.m.
  Odd Month, 1st Sat., 9:30 a.m.
  Vincent Hall, Univ. of MN
  Fred Olson (612) 588-9352

* MISSOURI
  Kansas City Chapter
  4th Tues., 7 p.m.
  Midwest Research Institute
  MAG Conference Center
  Linus Orth (913) 236-9189
  St. Louis Chapter
  1st Tues., 7 p.m.
  Thornhill Branch Library
  Contact Robert Watham
  91 West Dr.
  Ellisville, MO 63011

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

* NEW MEXICO
  Albuquerque Chapter
  1st Thurs., 7:30 p.m.
  Physics & Astronomy Bldg.
  Univ. of New Mexico
  Jon Bryan (505) 298-3292

* NEW YORK
  FIG, New York
  2nd Wed., 7:45 p.m.
  Manhattan
  Ron Martinez (212) 866-1157
  Rochester Chapter
  4th Sat., 1 p.m.
  Monroe Comm. College
  Bldg. 7, Rm. 102
  Frank Lanafame (716) 235-0168
  Syracuse Chapter
  3rd Wed., 7 p.m.
  Henry J. Fay (315) 546-4600

* NORTH CAROLINA
  Raleigh Chapter
  Frank Bridges (919) 552-1357

* OHIO
  Akron Chapter
  3rd Mon., 7 p.m.
  McDowell Library
  Thomas Franks (216) 336-3167
  Athens Chapter
  Joel Unelli (614) 594-3731
  Cleveland Chapter
  4th Tues., 7 p.m.
  Chagrin Falls Library
  Gary Bergstrom (216) 247-2492
  Dayton Chapter
  2nd Tues. & 4th Wed., 6:30 p.m.
  CPC. 11 W. Monument Ave.,
  #612
  Gary Ganger (513) 849-1483

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

* OREGON
  Greater Oregon Chapter
  Beaverton
  2nd Sat., 1 p.m.
<table>
<thead>
<tr>
<th>State/Territory</th>
<th>Chapter Name</th>
<th>Contact Person</th>
<th>Address</th>
<th>Phone Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXAS</td>
<td>Philadelphia Chapter</td>
<td>Melanie Hoag</td>
<td>2921 N. 35th East, Laramie, WY 82070</td>
<td>(307) 742-6268</td>
</tr>
<tr>
<td>TENNESSEE</td>
<td>East Tennessee Chapter</td>
<td>Joel McMurray</td>
<td>2010 E. 7th Ave, Johnson City, TN 37601</td>
<td>(423) 289-3000</td>
</tr>
<tr>
<td>PENNSYLVANIA</td>
<td>Pennsylvania Chapter</td>
<td>Richard Shirey</td>
<td>1500 Lititz Springs Rd, Lititz, PA 17543</td>
<td>(717) 652-3000</td>
</tr>
<tr>
<td>VIRGINIA</td>
<td>First Forth of Hampton Roads</td>
<td>William Edmonds</td>
<td>3000 South Capitol St, Suite 1200, Washington, DC 20001</td>
<td>(202) 974-3700</td>
</tr>
<tr>
<td>WISCONSIN</td>
<td>Lake Superior FIG Chapter</td>
<td>Donald Kimes</td>
<td>5260 S. 6th St. Apt. 4, Superior, WI 54880</td>
<td>(414) 377-0708</td>
</tr>
<tr>
<td>AUSTRALIA</td>
<td>Melbourne Chapter</td>
<td>C. R. Loomes</td>
<td>3535 Lincoln Rd, Mildura, VIC 3500</td>
<td>(03) 5022-2222</td>
</tr>
<tr>
<td>BELGIUM</td>
<td>Belgium Chapter</td>
<td>Christian DeCuyper</td>
<td>7 Rue de la Gare, 1100 Brussels, BE</td>
<td>(203) 999-3333</td>
</tr>
<tr>
<td>CANADA</td>
<td>Northern Alberta Chapter</td>
<td>Dr. N. Alta. Inst. of Tech.</td>
<td>9200 25th Ave S, Edina, MN 55435</td>
<td>(612) 920-4444</td>
</tr>
<tr>
<td>COLOMBIA</td>
<td>Colombia Chapter</td>
<td>Rodolfo Lara</td>
<td>PO Box 1234, Bogota, CO 32100</td>
<td>(503) 752-5113</td>
</tr>
<tr>
<td>DENMARK</td>
<td>Forth Interesse Group Denmark</td>
<td>Erik Oestergaard</td>
<td>1-520494</td>
<td></td>
</tr>
<tr>
<td>ENGLAND</td>
<td>Forth Interest Group UK</td>
<td>London</td>
<td>1st Tues., 7 p.m.</td>
<td>(01) 61-444-0306</td>
</tr>
<tr>
<td>FRANCE</td>
<td>French Language Chapter</td>
<td>Jean-Daniel Dodin</td>
<td>77 Rue du Caire, 75011 Paris, FR</td>
<td>(1) 75-22-2222</td>
</tr>
<tr>
<td>GERMANY</td>
<td>Hamburg FIG Chapter</td>
<td>John Horst-Gunter Lysace</td>
<td>2000 Hamburg 6</td>
<td>(01) 22-33-44</td>
</tr>
<tr>
<td>HOLLAND</td>
<td>Holland Chapter</td>
<td>Adriaan van Roomen</td>
<td>4817 Breda, NL 31730</td>
<td>(01) 31-31-31</td>
</tr>
<tr>
<td>IRELAND</td>
<td>Irish Chapter</td>
<td>Hugh Dobbs</td>
<td>05177575 or 05174124</td>
<td></td>
</tr>
<tr>
<td>ITALY</td>
<td>FIG Italia</td>
<td>Marco Tausel</td>
<td>20161 Milano</td>
<td>02/435249</td>
</tr>
<tr>
<td>JAPAN</td>
<td>Japan Chapter</td>
<td>Toshi Inoue</td>
<td>7-3-1 Hongo, Bunkyo, Tokyo 113</td>
<td>812-2111 ext. 7073</td>
</tr>
<tr>
<td>NORWAY</td>
<td>Bergen Chapter</td>
<td>Kjell Birger Faerass</td>
<td>47-518-7784</td>
<td>812-2111 ext. 7073</td>
</tr>
</tbody>
</table>
Australia $2550*

Forth Symposium—Sydney May 19-20
Organized by Forth users from industrial and academic organizations, the focus is Forth as a programming system for productivity. It will feature presented papers, demonstrations, and commercial exhibits. Charles Moore, Forth’s inventor, is the keynote speaker.

World Expo 88—Brisbane May 21-24
The highlight of Australia’s Bicentenary celebration, World Expo 88 features more than 30 nations and 20 corporations showcasing their achievements under the theme “Leisure in the Age of Technology”.

Great Barrier Reef—Hamilton Island May 25-27
You can fish, waterski, parasail, scuba dive, snorkel, play tennis or squash, cruise, sail, visit the Great Barrier Reef by helicopter or launch—or just soak up the sun around the Pacific’s largest fresh water swimming pool.

Group Tour
Departs San Francisco May 16, 1988 and returns May 28, 1988. Group travel includes air fare, ground transportation between airports and hotels, and hotel accommodations. Forth Symposium attendance is an additional $70 for each attendee and symposium banquet attendance is $20 per person. Local tours will be arranged for non-conference guests.

A brochure with a complete itinerary and additional information is available from the Forth Interest Group or Silicon Valley Travel, Inc.

* Payment must be received by March 31, 1988. After this date trans-Pacific fares will increase by $150 per person which will increase the tour price to $2700.

Make reservations directly with Silicon Valley Travel, Inc., 100-68 West El Camino Real, Mountain View, CA 94040. Telephone (415) 962-9820.

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