$10 Volume XVI, Number 5 January 1995 February

FORTH DIMENSIONS

euroForth '94

Forth2LaTeX

Zeller's Congruence

Forth-Oriented Compiler Compiler

Forth in 32-bit Protected Mode

Contents

Features

Departments

Editorial

Challenge and Opportunity

Please note that this issue contains an announcement of elections for the Forth Interest Group's Board of Directors (page 34). The job of being a Board member can be easy—or not. President John Hall oversees day-to-day operations, a not-inconsiderable job that entails telephones ringing at all hours, stacks of paperwork, and donating lots of personal time. Secretary Mike Elola, in addition to his column in this magazine, has devoted many hours to projects like the ANS Forth Quick Reference Card and press release, and to being program chairman of the last FORML conference.

In most organizations, it is demoralizing and disruptive when Board members take part in the operations. However, in our own case, there is no executive staff to whom Board members can delegate responsibilities and who can be held accountable for their jobs. Board members may assign certain tasks to responsible, tactful, persevering volunteers who are aligned with the group's goals and provenance, and who are willing to act in the spirit of teamwork; but even volunteers must be accountable to the organization's overall goals and spirit. The Board cannot just "wind up" various components and let them go their own way untended. The organization is like a sailboat, whose lines and sails must constantly be trimmed and tended to work efficiently.

There are many areas in which FIG could be more active, but relatively few staff hours are available. Specific objectives can be identified rather easily in the following areas: FIG Chapters, Forth-vendor relations, collaboration with other Forth organizations, public relations and marketing, membership services, international affairs, on-line resources, academic matters, and entry-level Forth training.

This is a time of both challenge and opportunity for the Forth community, and I would like each incoming Board member to have the opportunity to contribute tangibly and meaningfully. The fact that FIG has held its own during a time when many small publications and volunteer-based groups have fallen by the wayside is a credit to the organization and its members. Now I have asked the current Board's nominating committee to focus on unearthing candidates whose skills, temperament, and commitment qualify them to bring a renewed vitality to the Forth Interest Group.

—Marlin Ouverson
ouversonm@aol.com

Forth Dimensions
Volume XVI, Number 5
January 1995 February

Published by the
Forth Interest Group

Editor
Marlin Ouverson

Circulation/Order Desk
Frank Hall

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 $40 per year ($52 overseas air). For membership, change of address, and to submit items for publication, the address is: Forth Interest Group, P.O. Box 2154, Oakland, California 94621. Administrative offices: 510-89-FORTH. Fax: 510-535-1295. Advertising sales: 805-946-2272.

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

"*Forth Dimensions* (ISSN 0884-0822) is published bimonthly for $40/46/52 per year by the Forth Interest Group, 4800 Allendale Ave., Oakland, CA 94619. Second-class postage paid at Oakland, CA. POSTMASTER: Send address changes to *Forth Dimensions*, P.O. Box 2154, Oakland, CA 94621-0054."

Letters

Beginner's Perspective

Dear Mr. Ouverson,

I just rejoined FIG after a two-year absence. I never learned Forth the first time around for various reasons (work, learning C and Pascal, etc.), but I always liked what it seemed to promise, and now I'm ready to try it. I've noticed in volume 16 of *Forth Dimensions* a lot of talk about finding ways to increase the popularity of Forth, and I would like to add some input from a beginner's (my) perspective.

First is cost/support. I can get a first class C++ system that runs under Windows for about $80 (I can even choose from two vendors) that includes extensive printed documentation, toll-free factory phone support, and all the point-and-click ease of use that comes from running under Windows. I have seen Forth systems for around $50, but they are either shareware or were designed to run on 640K '286 systems. Nothing wrong with that, I guess, but it would be nice to have a language that made use of my system's resources ('486, etc.). The Forth systems with all the bells and whistles seem to start at $300 or more, and go up from there.

Second is ease of use. I'm currently working with two versions of Forth that I downloaded from Delphi, F-PC and UpperDeck Forth. They are okay, but it was difficult in both cases to find information in the included docs on compiling a file from disk. It would be nice if *all* versions of Forth had a common (and widely publicized) word for compiling a file from disk. By the way, the version of F-PC I'm working with will run a program fine when it's entered in direct mode, but crashes when the same program is compiled from disk. I plan on ordering the latest version (and user's manual) from FIG very soon; hopefully, it will fix this problem.

Third is the attitude toward other languages, especially C, that seems to be displayed on a regular basis in *Forth Dimensions*. Think about it... if you were a C programmer, and the first reason you heard from someone to convert to a new language is that *your* language was stupid (not said, but implied), would that give you "warm feelings" for the new language? If Forth can stand on its own merits (and I think it can), why do some people feel this need to berate other languages? I know FIG doesn't control what authors write, but perhaps it could encourage them to concentrate on Forth's strength instead of pointing out "weaknesses" in other languages.

Finally, something I would really like to see in *Forth Dimensions* is a relatively inexpensive hardware/software project. I think a good candidate for this would be interfacing the new MuP21 system to a PC. Also, I'd *really* appreciate it if you could put me in touch with anyone running Forth on the RCA 1802. Thanks!

Ken Deboy
glockr@delphi.com

Forth vendors have pointed out that the price/performance of commercial systems is related to the volume of their sales (and perhaps other factors). If every public-domain or shareware Forth used in a paying project were, instead, a commercial Forth (with good documentation and technical support), the vendors would be healthier, prices would be lower, and there would be more money for Forth marketing, for research and development, and for system enhancements.

I know you are on comp.lang.forth so, presumably, you will find expert assistance with F-PC there. Those folks are usually quite helpful with system-specific questions and anything else "beginners" need to know.

Your comment on the attitude toward other languages is well taken. The best approach I've heard lately is not "what is the best language," but rather, "which tool is right for the job." More rationale, less chauvinism, please.

FD *has long wanted to publish relatively simple "how-to" articles that teach hardware/software principles by example in Forth. Such things aren't the easiest to design and write, but would make an excellent contribution. Any takers?*

—Ed.

Structured Comment

1. In *FD* XVI/5, the article "HDTV Format Converter" by Philip Crosby shows the continuing power of Forth in the development community. I realized that this and other efforts, such as Elizabeth Rather's [of Forth, Inc.] work in delivering the Saudi airport control system in nine months, need to be publicized. A good approach would be for you to contact the *Wall Street Journal* to do a feature article on "The Unsung Force in Real-Time Project Developments."
2. The *Journal* has on the feature page specialized articles of innovation and interest. For example, they had a recent article on PGP, the encryption program and its developer. So the precedent is set.
3. You could direct the reporter to people like Chuck Moore, Elizabeth Rather, Larry Forsley, and, of course, Philip Crosby, and Doug Ross at NASA. There are so many successful applications of Forth, and the community has enough quirkiness that it would be a natural.

4. Pöial's article ["Algebraic Specification of Stack Effects," *FD* XVI/4] is deeply appreciated by those Forthers who work in logic. Please try to have more.

John C. Kotelly
Arlington, Massachusetts

Thanks, John.

1. *You're right. (Maybe the Denver airport should have tried Forth, Inc., too.) If you—or anyone else—convince the* Journal *to assign an independent reporter, we'll help them work out the angles and to make contacts in the Forth realm.*
3. *Quirkiness? You must have us confused with someone else...*
4. *Done—Jaanus Pöial has co-authored another article which is published in this issue. Enjoy!*

—Ed.

Backhanded Critique

Dear Forth community:

Recently I came across an article entitled, "The Nightmare of C++" in the November 1994 issue of *Advanced Systems* magazine. Having just finished a two-year project using C++, and also being a fan of Forth, my curiosity was piqued.

The article is an excerpt of a new book, *The UNIX-Hater's Handbook*, which is based on messages sent to an e-mailing list of people frustrated with UNIX and its associated tools. As many of the critiques in the article address issues important to the Forth community, I wanted to highlight some of them.

The first point is that object-oriented features were added to C because object orientation is supposed to simplify things for designers and programmers. However, as the article states, "Instead of simplifying things, C++ sets a new world record for complexity... It's just one big mess of afterthoughts."

The article describes the features of a high-level language:

- *Elegance:* there is a simple, easily understood relationship between the notation used by a high-level language and the concepts expressed.
- *Abstraction:* each expression in a high-level language describes one and only one concept. Concepts may be described independently and combined freely.
- *Power:* with a high-level language, any precise and complete description of the desired behavior of a program may be expressed straightforwardly in that language.

It seems to me that Forth addresses each of these points well. Forth's minimal syntax guarantees that there is an easily understood relationship between program notation and the concepts expressed. Each well-factored Forth word is a self-contained abstraction that can be combined freely with other words. And, of course, the extensibility of Forth means that an application-oriented vocabulary can be developed that allows precise behaviors to be expressed in a straightforward manner.

C++ fails to meet these requirements in a few important ways. In the area of dynamic memory management, the article points out that the C++ programmer is forced to do all of the garbage collection manually for objects after they are no longer needed. The program then has many lines of code devoted to the lower-level details of the memory management system. Maintaining references between objects (via pointers) becomes an inelegant chore. Other high-level languages spare the programmer these headaches by providing automatic garbage collection, which has many benefits in code size, readability, and correctness. However, many Forth systems simply avoid the issue by using the parameter stack to hold small "objects," the dictionary for larger ones, and block-I/O-based "virtual memory" (as Chuck Moore used to call it) for "persistent" or really large objects.

In the area of syntax and abstraction, there are many problems with C++ and so many rules that it is very difficult to learn them all, or even to implement them correctly in a compiler. For example, the C++ source line

```
//*****************************
```

is interpreted as different kinds of comments by different compilers. Some compilers will even crash when confronted with indeterminate syntax. Forth avoids the problems of complex syntax by using an extremely simple one.

The article makes a backhanded critique of Forth when it states, "The real power of C++'s operator overloading is that it lets you turn relatively straightforward code into a mess that can rival the worst of APL, Ada, or Forth code you might run across. Every C++ programmer can create their own dialect, which can be a complete obscurity to every other C++ programmer." This is really a critique of extensible languages, and the author does not seem to value extensibility, even as it provides abstraction and power. In Forth, operators are not overloaded, but are made more numerous (`+`, `D+`, etc.), which avoids some of the confusion, at the expense of having more words to learn about. It is true that bad Forth code is completely obscure, but good Forth code is a joy to read.

The article points out some problems with the header file/source file, used for most large programs, that discourage programmers from changing header files (forcing an entire system recompile), so they start doing funny things in the source code to compensate. In a Forth system, the user can quickly `FORGET` portions of a system and modify them. Reloading an entire system in Forth is fairly fast compared to recompiling and relinking an entire application in C++. The namespace system of C++ is a small improvement over C, but can still cause problems because, "There's no way to be sure you haven't taken a name used somewhere else in your program, with possibly catastrophic consequences." Forth provides vocabularies, so name collisions are avoided.

Finally, the article presents an anonymous and humorous piece entitled, "The Evolution of the Programmer."

My fear is that these shortcomings of C++ will be addressed by using even more complex tools and creating

a new heir to C++ with even more problems. Instead, people could be using a proven tool based on the common-sense principles of simplicity which we know as Forth.

Sincerely,
Michael A. Losh
Auburn Hills, Michigan

Communication Without BIOS

Dear Editor:

Mr. Wilson's problems with the COM ports on the PC (*FD* XVI/ 1) are easily explained. The device that is used in the PC for communications is one of the few that cannot be configured to recognize the CTS signal in hardware. Most communication chips will handle this; that is, when the CTS signal goes false, the transmitter is stopped. On the PC, this is not the case. So what happens is that the software checks that CTS is active, then stuffs one or more characters (depending on the model) into the transmit buffer. These characters are sent regardless of subsequent changes to the state of CTS.

The BIOS on the PC cannot be used if CTS handshaking is desired. However, since Mr. Wilson has already gone to the trouble of finding the I/O address of the chip, a solution is readily available. Check the Tx Shift Register Empty, and the Tx Holding Register Empty bits, as well as the CTS line. These are normally bits five and six of the Line Status Register (x3FD). If these bits are zero and CTS is zero, you can safely send one character. There is a slight possibility that the character will start out just as the CTS line goes inactive, but this has not proven to be a problem for me.

This is just one of the many reasons why few programs use the standard BIOS routines for serial I/O on the PC.

Sincerely,
Gene LeFave
Libertyville, Illinois

Branch Without Doubt

Dear Marlin,

I enjoyed Tom Napier's letter (*FD* XVI/1) because, after complaining of Forth code and comments in lower case, he advocated that the Forth branch statement should be:

```
IF{ do-this }ELSE{ do-that }IF
```

Lower case, curly brackets? The C disease is epidemic.

I, too, was a bit confused by the Forth branch syntax at first, and even tried the ENDIF route. Then I came across a great explanation of the Forth branch statement that solved the problem.

```
<exp> IF <true, do-this>
      ELSE <false, do-that>
      THEN <continue>
```

The idea of THEN <continue> actually makes the Forth statement scan better than that of BASIC or C. Sine all branch code is between the IF and THEN, there's also no doubt as to where the statement limits are.

Yours truly,
Walter J. Rottenkolber
Mariposa, California

dot-quote

The reports of Forth's death are premature, to paraphrase Twain.

A language can hardly be dead that lives on the motherboards of all SPARC workstations and has just been adopted as the implementation technology of the "plug and play" facility on the new generation of Power PCs in the form of IEEE1275 Open Firmware. Nor one in use by groups in most of the Fortune 500 firms.

There are a number of systems with full-functioned, object-oriented extensions, ranging from MOPS to our EXPRESS high-level process control product. And the latter is an example of application-oriented products finding their way on the market. Systems are available for the hot new platforms, from Win-NT to Power PC. Hot companies—from Federal Express to McCaw Cellular to Saturn, etc.—are using it. We just installed a Forth/EXPRESS control system on NASA's latest ground-based version of the big shuttle robot arm. There have been more than 30 space applications in the last five years. And the Forth-based Open Firmware is in more than one million SPARC workstations and will be in all Power PCs. Does this sound dead to you?

> >...the world hungered for code that was easy to
> >write/read/change and that could solve almost any
> >problem large or small acceptably.

...which is exactly what Forth was designed to be, and can be. This is, in fact, why Forth has survived. It hasn't thrived because C came with the authority of AT&T and academia behind it, and programmers and their managers often prefer to do what "everybody else" is doing rather than hold out for the best technology available.

Forth's limited penetration isn't a technical problem, it's a marketing problem.

—Elizabeth Rather on comp.lang.forth
Adapted with permission

Forth in 32-bit Protected Mode

Richard Astle
La Jolla, California

Perhaps since the beginning of the Intel 80x86 architecture, segment registers have been maligned, I think unfairly, but with the advent of the 80386 they have come into their own as a protected mode memory protection mechanism in an otherwise flat 32-bit memory space. The fact remains, however, that, in real mode, they make programs larger than 256K difficult to write, and (with the exception of F-PC's creative misuse) make all attempts at 32-bit Forths inefficient. Access to 32-bit protected mode, on the other hand, has been made difficult by the existence of DOS, and the overgrown 32-bit operating systems here and on the horizon, while promising great riches, do so at the cost of large learning curves and dependence on other people's (committees of other people) ideas of what constitutes the truth and the good of programming tools. There is nothing minimalist about them, and minimalism is one of the things Forth has always seemed to me to be about. So I thought that now, before DOS and the Intel architecture slip over the horizon, while real programming is still possible, it would be worthwhile to explore the possibility of creating a simple 32-bit protected mode Forth that runs on DOS on any 386SX or higher PC. This may be an idea whose time has passed, but it is also one that to my knowledge hasn't been sufficiently explored. And besides, I thought it would be fun.

I thought it would be interesting to create tools for integrating the 16-bit and 32-bit Forths, so that each could execute code in the other.

One way to go is to code Forth in assembler and use a DOS Extender. Dr. Ting's 32-bit eForth takes this approach. A straightforward translation of 16-bit eForth, it runs on DOS with the aid of a DOS extender called PROT, written by Al Williams and included in source form in his *DOS 5: A Developer's Guide*. PROT provides a layer of software that arbitrates between the 32-bit protected mode eForth code and 16-bit real mode DOS. The details of this arbitration are interesting, of course, but using PROT, or any other DOS extender, adds another level of dependence, with mysterious black-box features and varying license fees and restrictions. PROT is fairly open, since source code is provided for the cost of the book, but rights are granted only for non-commercial use, which one would like to think could be a problem.

I wanted to try a different tack, partly to try a different tack, but also to see how simply (elegance, not development time) I could get from DOS to a protected mode 32-bit Forth that I could be comfortable with. Using an assembler (or C or Pascal or BASIC compiler) does not seem to me to be simpler (though it may be easier) than the meta- or target compiler approach that stays within Forth. On the other hand, I'm in favor of taking what the operating system (whatever it happens or has to be) offers, making DOS calls to open files rather than programming the hardware to read disk blocks directly, for example.

The DOS Protected Mode Interface seems to me to provide the right level of simplicity and ease for access to protected mode. Programming with DPMI is a lot like programming with DOS: just as DOS provides functions to read and write files, to allocate (real mode) memory, etc., DPMI provides functions, mainly interrupt based, to switch to protected mode and to allocate and characterize (as 16- or 32-bit, code or data, etc.) protected mode memory. One can do these things oneself, and some prefer to, but, given the availability of DPMI, one doesn't have to.

A few years ago, the only readily available DPMI host (they call it a "host," not a "server") was in Microsoft Windows 3.0. This is not a very cheerful place to work, particularly if one is interested in simplicity, but it apparently was the basis of Ray Duncan's explorations in chapter nine of *Extending DOS*. Fortunately, memory managers such as QEMM, 386Max, and Novell DOS 7's EMM386.SYS now provide DPMI host services. (I mention only the DPMI environments I've actually used. There are also some DPMI services in Microsoft Windows 3.1, but I have been unable to launch a 32-bit Forth from a DOS box: perhaps the Windows DPMI will only host 16-bit clients.)

There are currently two versions of the DPMI specification. Version 0.9 is the most common, but 1.0 is available at least in 386Max. 386 machines with DOS need some kind of memory management anyway, and with these products we can get DPMI without getting GUI.

I should add at this point that, though Ray Duncan's mini-DOS extender in *Extending DOS* (perhaps that was the only interesting thing to do with DPMI in a Windows 3.0 DOS box) is enlightening, and a successful implementation of a protected mode INT 21h handler would no doubt improve performance, it is not necessary, not for the present project. I refrain from the story of my discovery of this fact.

Thumbnail Protected Mode

The switch to protected mode involves some initialization: mainly setting up descriptor tables and loading some special registers to point to these tables. Next, setting a bit in a control register, CR0, puts the processor in protected mode, at which point a FAR jump instruction is used to replace the segment value in the CS register (which points to a real mode paragraph address) with a selector. The other segment registers also need to be loaded with selectors before they can be used. Al Williams's hand-rolled DOS extender PROT spends a lot of code on this process, but it can all be handled with one DPMI call.

Selectors are indexes into tables of eight-byte memory descriptors. For now, we do not need to know that it is actually the top thirteen bits of a selector that form the index into a table of descriptors, the bottom two bits correspond to the privilege level (with three being the lowest, where normal programs live), and the remaining bit, bit two, indicates whether the selector references the global descriptor table or the current local descriptor table.

There are actually three kinds of descriptor tables: interrupt, global, and local. Each task has, or can have, its own LDT and IDT (details differ between DPMI versions), but there is only one GDT. DPMI provides functions to manipulate entries in the LDT and IDT, and keeps the system-wide GDT to itself.

A descriptor describes a memory block. It contains the base physical address and size of the segment and a variety of flags. For us the most important mark the distinction between code and data descriptors and, for code descriptors, the distinction between 16-bit and 32-bit code.

Part of the protection scheme in protected mode is that memory referenced by data selectors cannot be executed, and that referenced by code selectors cannot be written. In practice, this means that a selector pointing to a descriptor marked as describing a data segment cannot be loaded into CS, and, similarly, a selector for a code segment cannot be loaded into DS. Fortunately, the same memory can be referenced by multiple selectors, so the same physical memory can, through this "aliasing," be both written and read. The other distinction, between 16-bit and 32-bit descriptors, is more interesting. A single bit in a code descriptor determines whether instructions in that memory block are interpreted as 16- or 32-bit instructions—for example, whether B8h is interpreted by the processor to load AX with a two-byte literal or EAX with four bytes, whether PUSH puts two or four bytes on the stack, and whether relative jumps have 16- or 32-bit offsets.

The DPMI API

Programming the DPMI API is a lot like programming DOS functions: for the most part, you put a function number in AX, set various other registers, and call an interrupt, 2Fh or 31h. For the most part, the success of a call is indicated by clearing the carry flag, which I translate to a zero on the stack, the inverse of a Forth flag.

My code accesses a dozen DPMI functions, calling them in assembler and wrapping the results in high-level words with simple error handling. The functions I call and the words that wrap them are listed below. (`FREE-PROT-MEM` is included for completeness. It is not called, though it should be.) All of these functions are in DPMI versions 0.9 and 1.0.

In addition to these interrupt calls, three of these functions return addresses to jump to or call during mode switches. `GET-DPMI-ENTRY` returns an address to call to enter protected mode and how many paragraphs the DPMI host needs for its private data. (Other information is available: I get and display the DPMI version number also.) The DOS call in `MALLOC` allocates the requested memory and the entry point gets shoved into a variable referenced by `>PROTECTED-MODE`. The three words are compiled together into `(ACTIVATE-DPMI)`. Surrounding `(ACTIVATE-DPMI)` in `ACTIVATE-DPMI` is some other activity which I'll get to later.

Astle's function calls.

INT 2Fh:	
function 1686h - Get CPU Mode	`MODE?`
function 1687h - Get DPMI Entry Point	`GET-DPMI-ENTRY`
INT 31h:	
function 0000h - Allocate LDT Descriptor	`ALLOC-DESCRIPTOR`
function 0007h - Set Segment Base Address	`SET-SEG-BASE`
function 0008h - Set Segment Limit	`SET-SEG-LIMIT`
function 000Ah - Alias LDT Descriptor	`ALIAS-DESCRIPTOR`
function 000Bh - Get LDT Descriptor	`GET-DESCRIPTOR`
function 000Ch - Set LDT Descriptor	`SET-DESCRIPTOR`
function 0305h - Get State Save/Restore Addresses	`GET-SS-ADDRS`
function 0306h - Get Raw Mode Switch Addresses	`GET-RAW-MODES`
function 0501h - Allocate Protected Memory	`ALLOC-PROT-MEM`
function 0502h - Free Protected Memory	`FREE-PROT-MEM`

Before the mode switch, the segments contain familiar real-mode paragraph addresses, all the same because this is a single-segment Forth; afterwards, they contain selectors which reference the same physical memory but only indirectly, through entries in a descriptor table kept somewhere in memory. The values of CS and DS are guaranteed to be different from each other; ES would contain a selector referencing the program's PSP if I didn't reload it with the selector from DS, and SS is either the same as or different than DS depending on the particular implementation of DPMI.

The fact that the selector in DS references the same physical memory as that in CS (the former an "alias" of the latter) means that we can still compile Forth relative to DS and execute it relative to CS and proceed pretty much as we're used to. The code executed is the same, in the same place in physical memory, nothing gets moved. Some things, however, innocuous or clever in real mode, won't work: using CS to reload DS after a long CMOVE, for example. The words I had to redefine in the Forth I began with (Guy Kelly's Forth-83) all have an "L" in them: C@L, C!L, @L, !L, CMOVEL, and DUMPL. Segment arithmetic—adding a value to a segment register to get to another physical address—won't work either, so forget about using F-PC in protected mode.

Switching Back And Forth

In the case of the Forth I used to build this project, the editor does not work in protected mode. One way to overcome this would be to rewrite the editor and metacompile, or to write a new one, but that is an old exercise, and one I've never indulged in. For me, the more interesting, if slightly more clumsy, approach is to switch back to real mode to use the editor I'm familiar with.

Buried in ACTIVATE-DPMI, (ACTIVATE-DPMI) handles the activation of DPMI and the switch to protected mode. The rest of the baggage in ACTIVATE-DPMI is involved with setting up words to switch between real and protected mode after DPMI is activated. (The long call address obtained from GET-DPMI-ENTRY is only good for the initial leap into protected mode, since

```
Screen 0
DPMTOOLS.SCR -  DPMI FUNCTIONS FOR PROTECTED MODE FORTH
Richard Astle
POBox 8023
La Jolla, CA 92038
619 456-2253

In stack diagrams in the following screens I follow the usage
of the "Forth-83 Handy Reference, " adding
    id    for Intel double number (reverse word order)
    -flag for a flag where 0 indicates success
I tend to use u d ud id etc as prefixes.
Otherwise undesignated stack items (seg off etc.) are 16-bits.

Screen 1
\ USEFUL WORDS                                          RA 04SEP93
: 0!    OFF ;           : KD    KEY DROP ;      : ?     @ . ;
: 4DUP  2OVER 2OVER ;   : 4DROP 2DROP 2DROP ;   : 4+  2+ 2+ ;
: 1+! ( addr --- ) 1 SWAP +! ;
: PLUCK ( n1 n2 n3 --- n1 n2 n3 n1 ) 2 PICK ;
: H.R   BASE @ >R HEX .R R> BASE ! ;
HEX CODE DS! 1F C, NEXT, END-CODE
    CODE ES! 07 C, NEXT, END-CODE   DECIMAL
: FLAG  ( n --- flag ) 0= 0= ; \ normalizes flag, thanks bg

\ special for 32-bit distinctions
\ : S>D DUP 0< ;    \ sign extension of word to dword, not used
: US>D  ( u --- ud ) 0 ;      \ no sign extension, Forth double
: US>32 ( u --- id ) US>D SWAP ;  \ as above, but  Intel double
: D! -ROT SWAP ROT 2! ;           \ 2! for Intel double
-->


Screen 2
\ DATA ITEMS FOR THE FOLLOWING

CREATE (FREGS)     8 ALLOT   \ FORTH REGISTERS SAVE BUFFER
CREATE DESCRIPTOR 8 ALLOT    \ DESCRIPTOR BUFFER

VARIABLE modesw \ call address for dpmi mode switch

\ storage for segments/selectors for
\ 16bit Forth in real and protected mode
\ we only need one segment in real mode
VARIABLE REAL-SEG
\ we need to save all four in protected mode
VARIABLE PROT-CS VARIABLE PROT-DS
VARIABLE PROT-ES VARIABLE PROT-SS
-->


Screen 3
\ DATA ITEMS FOR THE FOLLOWING

2VARIABLE F-MEM-HAND     \ memory handle for 32-bit mem block
0 CONSTANT CODE-SEL      \ 32bit selectors
0 CONSTANT DATA-SEL      \ to be set later

HEX
E000 CONSTANT SP0-32     \ sp0 for 32-bit Forth - change?
0A00 CONSTANT &MSTART    \ where code will start in 32-bit seg
DECIMAL                  \ above 2 block buffers and TIB
-->


Screen 4
\ SHOW-SEGS  SHOW-ALL-SEGS  .STACK-INFO          RA 19JUN94
\ these words are useful for displaying state information
\ so we know where we are, obsessively
: SHOW-SEGS  CR
    ." CS@ = " CS@ H. ." DS@ = " DS@ H.
    ." ES@ = " ES@ H. ." SS@ = " SS@ H. ;

: SHOW-ALL-SEGS
    SHOW-SEGS
    CR ." CODE-SEL= " CODE-SEL H. ."  DATA-SEL= " DATA-SEL H. ;
```

```
: .STACK-INFO
    ." SP0 = " SP0 @ H. ." RP0 = " RP0 @ H.
    ." SP@ = " SP@   H. ." RP@ = " RP@   H. ;
-->


Screen 5
\ REDEFINED FOR PROTECTED MODE: C@L C!L          RA 14SEP93

HEX CODE C@L      ( seg off --- byte )
    8C C, D9 C,    5B C,   1F C,   B8 C, 0 ,
    \ MOV CX,DS    POP BX  POP DS  MOV AX,0
    8A C, 07 C,    50 C,    8E C, D9 C,     NEXT, END-CODE
    \ MOV AL,[BX]  PUSH AX  MOV DS,CX

HEX CODE C!L      ( byte seg off --- )
    8C C, D9 C,    5B C,   1F C,   58 C,
    \ MOV CX,DS    POP BX  POP DS  POP AX
    88 C, 07 C,    8E C, D9 C,             NEXT, END-CODE
    \ MOV [BX],AL  MOV DS,CX
-->


Screen 6
\ REDEFINED FOR PROTECTED MODE: !L  @L           RA 14SEP93

HEX CODE !L      ( n seg off --- )
    8C C, D9 C,   5B C,   1F C,    58 C,    89 C, 07 C,
    \ MOV CX,DS   POP BX  POP DS   POP AX   MOV [BX],AX
    8E C, D9 C,   NEXT,   END-CODE
    \ MOV DS,CX

HEX CODE @L      ( seg off --- n )
    8C C, D9 C,   5B C,   1F C,    FF C, 37 C,
    \ MOV CX,DS   POP BX  POP DS   PUSH [BX]
    8E C, D9 C,   NEXT,   END-CODE
    \ MOV DS,CX
-->


Screen 7
\ REDEFINED FOR PROTECTED MODE: DUMPL CMOVEL     RA 16JAN93
DECIMAL
: DUMPL ( seg adr cnt --- )
    BASE @ >R HEX
    0 DO CR OVER 5 U.R DUP 5 U.R 2 SPACES
        16 0 DO 2DUP C@L 3 U.R 1+ LOOP
        16 - 2 SPACES
        16 0 DO 2DUP C@L DUP BL < OVER ASCII ~ > OR
            IF DROP ASCII . EMIT ELSE EMIT THEN 1+
        LOOP
        KEY? ?LEAVE
    16 +LOOP 2DROP R> BASE ! ;
: CMOVEL ( seg off seg off len --- )
    0 ?DO 2OVER I + 2OVER I + 2SWAP C@L -ROT C!L  LOOP
    4DROP ;
-->


Screen 8
\ MINI ASSEMBLER MACROS                          RA 04SEP93
.IFNDF REL-TARG,
  : REL-TARG, ( addr --- )
  \ calculate and compile relative 16-bit jmp/call offset
    HERE 2+ - , ;
.ENDIF
: CREL-TARG, ( addr --- )
  \ calculate and compile relative 8-bit jmp/call offset
    HERE 1+ -   DUP -127 127 WITHIN
    IF C, ELSE CR ." 8-bit offset too far " KEY QUIT THEN ;
HEX
: INT,  ( int# --- ) CD C, C, ;
: INT21, 21 INT, ;  : INT31, 31 INT, ;  : INT2F, 2F INT, ;
: CALL,  E8 C, ;
: IRET,  CF C, ;    : RET,   C3 C, ;
-->
```

each invocation creates a new DPMI client, walled off by modes of protection from previous and later creations.) GET-RAW-MODES obtains two addresses, one for a far jump in protected mode to switch to real mode, and the other a real mode address for switching to protected mode. To make the switch, you load the segment or selector values, stack pointer, and instruction pointer for the other mode into general-purpose registers and make a far jump to the appropriate address. For details, see my code or the *DPMI Specification*. It is to enable these mode switches that ACTIVATE-DPMI stores register values in the variables REAL-SEG, PROT-CS, PROT-DS, etc. The stack is the same in both real and protected modes, so the value of SP is simply stored in BX before the call to preserve it. Finally, through a little stack manipulation during the compiling of the words REAL->16PROT and PROT->16REAL, DI (which holds the new IP) is loaded with the address of the location just after the long jump.

GET-SAVE-STATE-ADDRESSES returns a pair of addresses to call to preserve the state of the machine before switching modes, along with the size of the buffer needed. Not all DPMI hosts require this state-saving: I remained happily ignorant of it using QEMM and 386Max, but Novell's EMM386 (with Novell DOS 7) woke me up. (In fact, I suspect Novell's strict implementation of DPMI in this area helps reveal incompatibilities due to careless programming in DPMI client applications which run in those more forgiving environments. Borland's command-line compiler for C++ 3.1 won't run, for example, though it works fine, if slowly, in a DOS box in Windows.)

The result of the call to GET-SAVE-STATE-ADDRESSES is four functions, saving and restoring state in protected and real mode. These and the functions created from the addresses returned by GET-RAW-MODES are hidden in >PROT and >REAL. Each of these words calls MODE? to find out if it is already in the requested mode. If so, it does nothing; if not, it saves the state of the current mode, makes the switch, and restores the state of the destination mode. Of course, the first invocation of >REAL calls RESTORE-REAL-STATE before SAVE-REAL-STATE has ever been

called, but this is an inevitable chicken-and-egg thing that the DPMI host has to know how to handle if the raw mode switches are to work at all.

32-Bits

Since at this point I have the same Forth in the same place in physical memory, and some things that used to work don't anymore, I seem not to have gained much (the normal Forth address space hasn't even gotten bigger), but what I do have now is the ability to create a 32-bit memory segment and target compile a Forth into it. The word GRAB-32BIT-MEMORY, which calls for most of its work GET-32SEG, coordinates the setting up of the 32-bit address space where we can target compile a 32-bit Forth. To do so, it makes a series of DPMI calls to seven different DPMI functions in the words ALLOC-PROT-MEM, ALLOC-DESCRIPTOR, SET-SEG-BASE, SET-SEG-LIMIT, SET-SEG-32, SET-SEG-CODE, and ALIAS-DESCRIPTOR. For the most part, these words are simple DPMI calls, and their naming and sequence make their function fairly clear: First we allocate some memory and a selector/descriptor to reference it, and set the base and limit of the descriptor to the base address and size of the segment we've allocated. The words SET-SEG-32 and SET-SEG-CODE are a little different: they call GET-DESCRIPTOR to get the descriptor associated with the selector into an eight-byte buffer we can modify (and display), set a bit, and call SET-DESCRIPTOR to put it back. (As with most things there's another way to do this, without the buffer, through function 0009h, "Set Descriptor Access Rights." But that method requires setting all access-right bits at once, while the method I use requires no assumptions about or even knowlege of the other 14 bits.) After these calls, we have a 32-bit code segment referenced through the allocated selector. The final call in this sequence, ALIAS-DESCRIPTOR, creates a data descriptor with the same physical base address and limit as the code descriptor it aliases. Between these two selectors, placing the first in CS and the second in DS, I can write and run code in the 32-bit memory space.

The Target

I base my 32-bit target compilation on

```
Screen 9
\ SAVE AND RESTORE FORTH REGISTERS      19JUN94 RA 03JAN93
HEX
HERE
    89 C, 2E C, (FREGS) 2+  , \ SAVE BP = FORTH RP
    89 C, 1E C, (FREGS) 4 + , \ SAVE BX = FORTH W
    89 C, 36 C, (FREGS) 6 + , \ SAVE SI = FORTH IP
    RET,
    CONSTANT (SAVE-FREGS)
HERE
    8B C, 2E C, (FREGS) 2+  , \ RESTORE BP
    8B C, 1E C, (FREGS) 4 + , \ RESTORE BX
    8B C, 36 C, (FREGS) 6 + , \ RESTORE SI
    RET,
    CONSTANT (RESTORE-FREGS)
: SAVE-FREGS,    CALL, (SAVE-FREGS)    REL-TARG, ;
: REST-FREGS,    CALL, (RESTORE-FREGS) REL-TARG, ; -->


Screen 10
\ MORE MINI ASSEMBLER MACROS            12SEP93 RA 04SEP93
HEX
: ifc,  ( --- addr ) 72 C, HERE  0 C, ;
: then, ( addr --- )
    HERE OVER 1+  -      \ adr len
    DUP FF > IF CR ." 8-BIT FWD JUMP TOO LONG " KEY QUIT THEN
    SWAP C! ;
: begin,  HERE ;
: zuntil, 75 C, CREL-TARG, ;
: FLAG-NEXT,
    \ extended NEXT, which converts CF to a Forth flag on the
    \ stack, and restores saved values of Forth registers
    B8 C, FFFF ,                    \ MOV AX,-1
    ifc,  B8 C, 0000 ,  then,       \ MOV AX,0
    50 C, REST-FREGS,  NEXT, ;      \ PUSH AX   NEXT
DECIMAL                                        -->


Screen 11
\ DOS CALL: MALLOC MEMORY ALLOCATION            RA 07SEP93
HEX
CODE MALLOC?   ( #pars --- #pars alloc-seg/code  -flag )
    SAVE-FREGS,
    5B C,                       \ POP BX    \ FIX 12/6/93
    B4 C, 48 C, INT21,          \ MOV AH,48 INT 21
    53 C,                       \ PUSH BX = MAXPARS
    50 C,   FLAG-NEXT,          \ PUSH AX = 8 IF INSUFF
END-CODE
DECIMAL

: MALLOC
    MALLOC?                     \ #pars seg/code flag
    IF CR ." MALLOC FAILURE " H. QUIT THEN ;
-->


Screen 12
\ DPMI FUNCTIONS: MODE?                         RA 07SEP93
HEX
CODE MODE?   ( --- mode-indicator )
    SAVE-FREGS,
    B8 C, 1686 ,  INT2F,  50 C,
    \ MOV AX,1686 INT 2F  PUSH AX
    REST-FREGS,
    NEXT,
END-CODE
DECIMAL
-->
mode-indicator = 0 when processor in protected mode


Screen 13
FUNCTIONS: GET ENTRY POINT ETC.          RA 07SEP93
HEX
CODE GET-DPMI-ENTRY? ( --- #pars  version  dAddr  -flag )
    SAVE-FREGS,
    B8 C, 1687 , INT2F,         \ MOV AX,1687 INT 2F
```

```
    56 C,                          \ PUSH SI = paragraphs needed
    52 C,                          \ PUSH DX = DPMI VERSION
    57 C, 06 C,                    \ PUSH DI PUSH ES = ENTRY POINT
    50 C,                          \ PUSH AX = 0 if successful
    REST-FREGS,
    NEXT,
END-CODE
DECIMAL
: GET-DPMI-ENTRY
    GET-DPMI-ENTRY?
    IF CR ." GET-DPMI-ENTRY FAILURE " H. QUIT THEN ;     -->


Screen 14
\ DPMI FUNCTIONS: CALL DPMI ENTRY POINT          RA 07SEP93

HEX CODE >PROTECTED-MODE?  ( data-seg --- -flag )
    SAVE-FREGS,
    07 C,                      \ POP ES
    B8 C, 0000 ,               \ MOV AX,0000
    FF C, 1E C, modesw ,       \ to protected mode
    1E C, 07 C,                \ PUSH DS POP ES
    FLAG-NEXT,
END-CODE DECIMAL

: >PROTECTED-MODE
    >PROTECTED-MODE?
    IF CR ." failure to get into protected mode " QUIT THEN ;
-->


Screen 15
\ DPMI FUNCTIONS: GET- SET-DESCRIPTOR            RA 07SEP93
HEX CODE (GET/SET-DESC?)  ( selector fn --- code -flag )
    SAVE-FREGS,
    58 C,                          \ POP AX=FUNC
    5B C,                          \ POP BX
    BF C, DESCRIPTOR ,             \ MOV DI,DESCRIPTOR
    INT31,                         \ INT 31
    50 C,                          \ PUSH AX=CODE
FLAG-NEXT,  END-CODE
: GET-DESCRIPTOR? ( sel --- code -flag ) 000B (GET/SET-DESC?) ;
: SET-DESCRIPTOR? ( sel --- code -flag ) 000C (GET/SET-DESC?) ;
DECIMAL -->


Screen 16
\ DPMI FUNCTIONS: ALLOC- FREE-PROT-MEM           RA 14SEP93
HEX CODE ALLOC-PROT-MEM?   ( dBytes --- dAddr dHand code -flag )
    SAVE-FREGS,
    5B C, 59 C,                    \ POP BX POP CX ( BX;CX )
    B8 C, 0501 , INT31,            \ MOV AX,0501 INT 31
    51 C, 53 C,                    \ PUSH CX PUSH BX ( BX:CX )
    57 C, 56 C,                    \ PUSH DI PUSH SI ( SI:DI )
    50 C,
FLAG-NEXT,  END-CODE
HEX CODE FREE-PROT-MEM?   ( dHand --- code -flag )
    SAVE-FREGS,
    5E C, 5F C,                    \ POP SI POP DI ( SI;DI )
    B8 C, 0502 , INT31,            \ MOV AX,0502 INT 31
    50 C,
FLAG-NEXT,  END-CODE  DECIMAL -->


Screen 17
\ DPMI FUNCTIONS: ALLOC- ALIAS-DESCRIPTOR        RA 14SEP93

HEX CODE ALLOC-DESCRIPTOR?  ( --- selector -flag )
    SAVE-FREGS,
    B9 C, 0001 ,                   \ MOV CX,1
    B8 C, 0000 , INT31,            \ MOV AX,0000 INT 31
    50 C,                          \ PUSH SELECTOR
FLAG-NEXT,  END-CODE

HEX CODE ALIAS-DESCRIPTOR?  ( selector --- selector' -flag )
    SAVE-FREGS,
    5B C,                          \ POP BX ( SELECTOR )
```

Robert Illyes' riFORTH, an elegant, spare, subroutine-threaded Forth that has the particular virtue of tininess. Not only is there relatively little source to convert, but I was able, without too many pangs, to dump the memory image and trace the machine code by hand, more than once. Guy Kelly used riFORTH as the basis for an exploration of Forth timings in his paper "Various Forths," which appears in the *1991 FORML Conference Proceedings* and in *Forth Dimensions* [XIII/6]. Mr. Kelly kept the riFORTH wordset and changed its underpinnings to explore different kinds of nestings and threadings; I keep the subroutine-threaded model, but change word names, definitions, and other features at will, so much so that I should probably call it raFORTH, to indicate, by name, both the closeness and distance of my implementation.

All Forths have at least two memory spaces, since the stack is always logically, if not physically, separate from code and data. On the other hand, despite schemes which separate code, data, lists, and headers, for the most part Forth is interactive, we execute what we write, and these distinctions are only conveniences in a segmented world. 16-bit riFORTH, like most Forths, sets all segment registers the same. For my 32-bit version, I set DS and CS to point to the same memory space, using the selectors returned by `GET-32SEG`, and I give ES the same selector as DS, but I leave SS alone, so it points to the same memory space as the 16-bit Forth stack. Leaving the stack segment alone made switching code segments easier, and allows passing data between the 16- and 32-bit Forths. The separate stack space enforces a certain discipline and the rewriting of a few code words (`R>`, `>R`, `R@`, `OVER`) that depended on indexing into the stack relative to DS, but it does not cause the obvious problem: though the stack in the 32-bit Forth is 32-bits wide, the size of `PUSH` and `POP` is determined by the characteristics of the code selector/descriptor, not the stack descriptor. The only issue is alignment, since a 32-bit `PUSH` or `POP` is more efficient when `SP` is on a double-word boundary.

The target compilation is straightforward. Perhaps the hardest part was figuring out how to code some of the 32-bit instructions: bit patterns twist one's brain.

For the most part, I keep Illyes' mini-assembler, extending register names to their 32-bit forms and adding a few instructions I found I needed. The assembler is not complete, but it's relatively easy to extend. I like Illyes' use of the character | for CODE. Illyes' assembler syntax is <source> <destination> <opcode>, so that "ebx eax mov" stores the contents of ebx in eax, not the other way around, as in many assemblers, Forth or not.

The target assembler C-commas code into the 32-bit segment, and M], instead of merely setting MSTATE, actually performs the target compilation, combining target words to make other target words. Since IMMEDIATE words in the target cannot be executed, their functionality is included in words beginning (for the most part) with "M" and included in a VOCABULARY called META in the host Forth. M] is a factor of M:, and runs as long as MSTATE is ON, parsing the input stream a word at a time, searching first in the target dictionary and then in the META vocabulary in the 16-bit host before giving up and trying number conversion. M[in META, and a factor of M:, turns MSTATE OFF and stops metacompilation until the next M[, M:, MCONSTANT, or MVARIABLE. Perhaps because riFORTH is so small, I was able to avoid forward references almost entirely. COLD, as in 16-bit metacompiled riFORTH, is referenced by a jump at the beginning of the code image, but this is to make it easier to find the entry point. Only QUIT needs to be used in a definition before it can be defined.

Among changes I made in riFORTH are the following: I made BLOCK zero-based, rather than one-based, meaning that the first block in a file is block zero, as in most Forths. I prefer to keep the first screen (block zero) for a documentation and title screen. I moved the two block buffers from the top of the memory map to the bottom, where they won't get in the way of seemingly infinite code expansion; I put TIB down there too. I changed the LINK field in the header to be a 16-bit relative offset to the previous link rather than an absolute (segment relative) address. This saves space, but a 32-bit relative offset might be better in some situations. I renamed Illyes' word \ to POSTPONE, its ANS equivalent. I prefer to use the backslash as a comment

```
    B8 C, 000A , INT31,          \ MOV AX,000A INT 31
    50 C,                        \ PUSH AX =- NEW SELECTOR
FLAG-NEXT,  END-CODE    DECIMAL -->


Screen 18
\ DPMI FUNCTIONS: SET-SEG-BASE -LIMIT           RA 14SEP93
HEX CODE (SET-SEG-SIZE?)     ( d s fn --- code -flag )
    SAVE-FREGS,
    58 C,                        \ POP AX=FUNC
    5B C,                        \ POP BX=SELECTOR
    59 C, 5A C,                  \ POP CX POP DX ( CX:DX )
    INT31,                       \ INT 31
    50 C,                        \ PUSH AX=CODE
FLAG-NEXT,  END-CODE
: SET-SEG-BASE?     ( dBase  sel --- code -flag )
    0007 (SET-SEG-SIZE?) ;
: SET-SEG-LIMIT?    ( dLimit sel --- code -flag )
    0008 (SET-SEG-SIZE?) ;
DECIMAL -->


Screen 19
\ GET-RAW-MODES                              27MAY94 RA 12NOV93
\ NOTE: these functions never fail, according to DPMI spec
HEX CODE (GET-ADDRS) ( fn --- r>p.off r>p.seg p>r.off p>r.seg )
    SAVE-FREGS,
    58 C,               \ POP AX=FUNC
    INT31,
    51 C, 53 C,         \ PUSH CX PUSH BX \ BX:CX = REAL->PROT
    57 C, 56 C,         \ PUSH DI PUSH SI \ SI:DI = PROT->REAL
    50 C,               \ PUSH AX
    REST-FREGS,  NEXT,  END-CODE
: GET-RAW-MODES ( --- r>p.off r>p.seg p>r.off p>r.seg )
    0306 (GET-ADDRS) DROP ;
: GET-SS-ADDRS  ( --- r>p.off r>p.seg p>r.off p>r.seg bufsz )
    0305 (GET-ADDRS) ;
DECIMAL -->


Screen 20
\ DPMI CALLS: ERROR WRAPPED VERSIONS    03OCT93 RA 18SEP93

: GET-DESCRIPTOR    ( selector --- ) \ puts descriptor in buffer
    GET-DESCRIPTOR?
    IF CR ." GET DESCRIPTOR FAILURE " H. QUIT THEN DROP ;
: SET-DESCRIPTOR    ( selector --- ) \ puts buffer in descriptor
    SET-DESCRIPTOR?
    IF CR ." SET DESCRIPTOR FAILURE " H. QUIT THEN DROP ;

: SET-SEG-LIMIT     ( dLimit selector --- )
    SET-SEG-LIMIT?
    IF CR ." SEG-SEG-LIMIT FAILURE "  H. QUIT THEN DROP ;
: SET-SEG-BASE      ( dBase selector --- )
    SET-SEG-BASE?
    IF CR ." SET-SEG-BASE FAILURE "   H. QUIT THEN DROP ;
-->


Screen 21
\ DPMI CALLS: ERROR WRAPPED VERSIONS    03OCT93 RA 18SEP93

: ALLOC-PROT-MEM    ( dBytes --- dAddress dHandle )
    ALLOC-PROT-MEM?
    IF CR ." ALLOT-PROT-MEM FAILURE "    H. QUIT THEN DROP ;
: FREE-PROT-MEM     ( dHandle --- )
    FREE-PROT-MEM?
    IF CR ." FREE PROT MEM FAILURE "     H. QUIT THEN DROP ;

: ALLOC-DESCRIPTOR  ( --- selector )
    ALLOC-DESCRIPTOR?
    IF CR ." ALLOC DESCRIPTOR FAILURE "  H. QUIT THEN ;
: ALIAS-DESCRIPTOR  ( selector --- selector' )
    ALIAS-DESCRIPTOR?
    IF CR ." ALIAS DESCRIPTOR FAILURE "  H. QUIT THEN ;  -->
```

```
Screen 22
\ DIAGNOSTIC: .DESC .DESCRIPTORS        04DEC93 RA 06OCT93
: <##> 0 <# # # #> TYPE ;
: .DESC ( selector --- )
    BASE @ HEX SWAP DUP CR <##> ." : "
    GET-DESCRIPTOR
    8 0 DO DESCRIPTOR I + C@ <##> SPACE LOOP
    CR ." BASE = " DESCRIPTOR 2+ @ \ low word
    DESCRIPTOR 4+ C@ DESCRIPTOR 7 + C@ FLIP +            UD.
    ."   LIMIT = " DESCRIPTOR @ DESCRIPTOR 6 + C@ 15 AND UD.
    BINARY
    CR ." ACCESS RIGHTS " DESCRIPTOR 5 + C@ 8 U.R
    ."   GDOAVL = " DESCRIPTOR 6 + C@ 16 / 15 AND 4 U.R
    BASE ! ;
: .DESCRIPTORS
    CR ." CS DESC " CS@ .DESC CR ." DS DESC " DS@ .DESC
    CR ." ES DESC " ES@ .DESC CR ." SS DESC " SS@ .DESC ; -->


Screen 23
\ SET SELECTOR TYPE AND SIZE                        RA 18SEP93
HEX : (SET-SELECTOR-TYPE)    ( mask selector --- )
    TUCK GET-DESCRIPTOR      \ sel priv
    DESCRIPTOR 5 + DUP C@    \ sel priv adr b
    F1 AND ROT 2* OR         \ sel adr b mask
    SWAP C! SET-DESCRIPTOR  ;
: SET-SEG-DATA ( selector --- ) 1 SWAP (SET-SELECTOR-TYPE) ;
: SET-SEG-CODE ( selector --- ) 5 SWAP (SET-SELECTOR-TYPE) ;
: (SET-SELECTOR-SIZE)        ( mask selector --- )
    TUCK GET-DESCRIPTOR      \ sel mask
    DESCRIPTOR 6 + DUP C@    \ sel mask adr b
    BF AND ROT OR
    SWAP C! SET-DESCRIPTOR  ;
: SET-SEG-32 ( selector --- ) 40 SWAP (SET-SELECTOR-SIZE) ;
: SET-SEG-16 ( selector --- )  0 SWAP (SET-SELECTOR-SIZE) ;
DECIMAL     -->


Screen 24
\ GET-MEM-INFO                                      RA 11SEP94
HEX CODE GET-MEM-INFO ( buffer --- ) \ always succeeds
    SAVE-FREGS,
    5F C,      B8 C, 0500 ,   INT31,
    \ POP DI   MOV AX,0500
    REST-FREGS, NEXT,
END-CODE  DECIMAL
-->


Screen  25
\ GET-MEM-INFO                                      RA 11SEP94
: ?.VAL
    2@ SWAP 2DUP AND -1 = IF ." unavailable " EXIT THEN D. ;
: .MEM-INFO CR ." MEMORY INFO via FUNC 0500: "
    HERE GET-MEM-INFO \ HERE 48 DUMP
    CR ." MAX MEM BLOCK:                " HERE 2@ SWAP D.
    CR ." MAX UNLOCKED PAGES:           " HERE  4 + ?.VAL
    CR ." MAX LOCKED PAGES:             " HERE  8 + ?.VAL
    CR ." LINEAR ADDR PAGES:            " HERE 12 + ?.VAL
    CR ." TOTAL UNLOCKED PAGES:         " HERE 16 + ?.VAL
    CR ." TOTAL FREE PAGES:             " HERE 20 + ?.VAL
    CR ." TOTAL PHYSICAL PAGES:         " HERE 24 + ?.VAL
    CR ." FREE LINEAR PAGES:            " HERE 28 + ?.VAL
    CR ." PAGING FILE/PARTIITION PAGES:" HERE 32 + ?.VAL ; -->


Screen  26
\ GET-32SEG                                         RA 14NOV93
: GET-32SEG ( dSize --- code.sel data.sel dMemhand )
    2DUP ALLOC-PROT-MEM          \ dSize dBase dMemhand
    >R >R ALLOC-DESCRIPTOR       \ dSize dBase sel ; R: dHand
    DUP >R SET-SEG-BASE          \ dSize     ; R: dHand sel
    R@     SET-SEG-LIMIT         \             R: dHand sel
    R> DUP SET-SEG-32                \ sel
    DUP    SET-SEG-CODE          \ code-sel
    DUP ALIAS-DESCRIPTOR  R> R> \ code-sel data-sel dHand
; -->
```

character, and POSTPONE is easier to see when you're scanning the code. I also added a control structure I've become used to in my other Forth life, ((...)), which is functionally equivalent to 0 DO ... LOOP.

Ways In

Once the target is compiled, the next step is to go there and run things. The 32-bit Forth is a real Forth, extensible from the keyboard, and, although it has no editor, it can LOAD source from block files edited elsewhere. It would be easy for those who enjoy such things to add an editor. Illyes' word SAVE creates a DOS file and saves the 32-bit Forth image in it. The syntax is SAVE <filename>, but the file created is not directly executable. Under DOS, every 32-bit program needs a 16-bit loader, and here that loader is in the 16-bit Forth, in the command LOAD-IMAGE <filename>. LOAD-IMAGE first makes sure that DPMI has been activated and the processor is in protected mode, then opens the file and copies its contents into the 32-bit segment. Some of the words that do this are peculiar to this particular 16-bit Forth, but any Forth that can load files into memory can do this. To be more like a "real" 32-bit program, the 16-bit Forth could be configured to switch to protected mode and load and jump to the 32-bit Forth at startup, without pausing at or announcing its stages.

Multi-Modalism

Most 32-bit programs running on DOS want to get to 32-bit protected mode as quickly as possible and stay there until the program terminates. But these programs aren't built by Forth programmers or, if they are, they have less Forth in them than they could. Rather than disguise its 16-bit origins, I thought it would be interesting to create tools for integrating the 16-bit and 32-bit Forths, so that each could execute code in the other, and perhaps so that I could feel better about not writing a 32-bit editor. This way, a potential hybrid application could be written in a combination of 16-bit and 32-bit Forths, with each portion performing the parts it is best suited for, whatever they are.

The first step in this cooperativeness is to be able to switch among modes. I've already discussed words to switch be-

tween 16-bit protected and real modes. The switch between 16-bit to 32-bit protected modes is accomplished by self-modifying code on both sides of the line, in the words (CALL-F32) on the 16-bit side and PRE-COLD and BACK on the 32-bit side. The heart of this process is the code word (CALL-F32), which initializes the 32-bit Forth's RP, DS, and ES, then makes the long call that loads CS and the processor's IP (also Forth's IP, since it is a subroutine-threaded Forth we're going to). The definition of (CALL-32) uses the same trick as the mode-switching word definitions, assigning constants to the addresses of spots within the definition that can be filled with the values of the selectors returned by (ACTIVATE-DPMI), which is done by (GO-32), the word that encloses (CALL-32). (GO-32) also puts the value of SP0 (extended to 32-bits in Intel order by US>32) on the stack so the 32-bit Forth can reference it.

On the 32-bit side, the entry point is redirected to PRE-COLD, which handles some initialization even before COLD gets hold of things. (In fact, COLD itself is bypassed in this version and replaced by DISPATCHER, which I'll discuss later.) Since the 32-bit Forth has the top stack element in a register, the first thing PRE-COLD does is execute DROP (a 32-bit DROP, of course) to fill that register (EBX) with the value the 16-bit Forth left for it. The next thing PRE-COLD does is to take the 16-bit return address and selector left on the stack by the long call and store them in variables so they can be used to return to the 16-bit side. In other languages this return would be accomplished by some version of the RET instruction, but in Forth that would require unreasonably perfect stack management. (Sometimes variables are just better.)

The word to return to the 16-bit Forth is called BACK. It begins (after the diagnostic .STACKS) with a DUP to get the top-of-stack back on the stack, and then makes a long jump to the address and selector stored by PRE-COLD. The simplicity of the code, as it stands, hides a lot of experimentation to figure out which stack (in riFORTH, ESP references both stacks, with the other pointer stored in SI and swapped appropriately) to look at, just where on the stack, and what to do with the stack pointer before the return.

```
given a segment size, this word allocates memory and
creates 2 selectors (code and data) pointing to it
--the selectors and a memory segment handle are returned
--the handle is required for freeing the segment later


Screen 27
\ MODE SWITCHING: 16PROT->REAL                    RA 12NOV93
HEX CODE 16PROT->REAL   SAVE-FREGS,
    A1 C, REAL-SEG ,     \ rds           \ MOV AX,[REAL-SEG]
    8B C, C8 C,                          \ MOV CX,AX (REAL ES)
    8B C, D0 C,                          \ MOV DX,AX (REAL SS)
    8B C, F0 C,                          \ MOV SI,AX (REAL CS)
    8B C, DC C,                          \ MOV BX,SP (R&PROT SP)
    BF C, HERE 0 ,       \ rds rip       \ MOV DI,0000 (REAL IP)
    EA C, HERE 0 , 0 ,   \ rds rip &sw   \ JMP MODESW
    SWAP HERE SWAP !     \ rds &sw
    REST-FREGS,
NEXT, END-CODE
CONSTANT P->R-SWITCH
-->


Screen 28
\ MODE SWITCHING: REAL->16PROT                    RA 12NOV93
HEX CODE REAL->16PROT   SAVE-FREGS,
    A1 C, PROT-CS , 8B C, F0 C,  \ MOV AX,[PROT-CS] MOV SI,AX
    A1 C, PROT-ES , 8B C, C8 C,  \ MOV AX,[PROT-ES] MOV CX,AX
    A1 C, PROT-SS , 8B C, D0 C,  \ MOV AX,[PROT-SS] MOV DX,AX
    A1 C, PROT-DS ,              \ MOV AX,[PROT-DS]
    8B C, DC C,                  \ MOV BX,SP (R&PROT SP)
    BF C, HERE 0 ,       \ ...   ip      \ MOV DI,0000 (PROT IP)
    EA C, HERE 0 , 0 ,   \ ...   ip &sw  \ JMP MODESW
    SWAP HERE SWAP !     \ ...   &sw
    REST-FREGS,
NEXT, END-CODE
CONSTANT R->P-SWITCH
-->


Screen 29
\ MODE SWITCHING: STATE SAVE FUNCTIONS  28MAY84 RA 27MAY94
HEX CODE (REAL-SAVE/RESTORE)
    58 C,                \ POP AX (0 or 1)
    BF C, HERE 0 ,       \ MOV DI,bufadr -- ES = DS already
    9A C, HERE 0 , 0 ,   \ FAR CALL
NEXT, END-CODE
CONSTANT REAL-SAVE-ADDR     CONSTANT REAL-SAVE-BUFF-ADDR
CODE (PROT-SAVE/RESTORE)
    58 C, BF C, HERE 0 ,
    9A C, HERE 0 , 0 , NEXT, END-CODE   \ as above
CONSTANT PROT-SAVE-ADDR     CONSTANT PROT-SAVE-BUFF-ADDR
: SAVE-REAL-STATE    0 (REAL-SAVE/RESTORE) ;
: SAVE-PROT-STATE    0 (PROT-SAVE/RESTORE) ;
: RESTORE-REAL-STATE 1 (REAL-SAVE/RESTORE) ;
: RESTORE-PROT-STATE 1 (PROT-SAVE/RESTORE) ;
DECIMAL                  -->


Screen 30
\ SETUP-STATE-SAVING                    28MAY94 RA 13NOV93
: SETUP-STATE-SAVING
    \ get and save state save addresses:
    \ the DPMI server with Novell DOS 7 needs this
    \ though QEMM and 386Max don't
    GET-SS-ADDRS
    HERE REAL-SAVE-BUFF-ADDR !  DUP ALLOT
    HERE PROT-SAVE-BUFF-ADDR !      ALLOT
    PROT-SAVE-ADDR D!
    REAL-SAVE-ADDR D!
;           -->
also getting state save/restore addresses which Novell's DPMI
server (in Novell DOS 7) needs but QEMM's and 386-Max's don't
```

```
Screen 31
\ MODE SWITCHING: >PROT >REAL                          RA 13NOV93
: >PROT
    SHOW-SEGS   MODE?
    IF  SAVE-REAL-STATE  REAL->16PROT  RESTORE-PROT-STATE
    ELSE CR ." ALREADY IN PROTECTED MODE "
    THEN SHOW-SEGS ;
: >REAL
    SHOW-SEGS   MODE?
    IF   CR ." NOT IN PROTECTED MODE "
    ELSE  SAVE-PROT-STATE  16PROT->REAL  RESTORE-REAL-STATE
    THEN SHOW-SEGS ;
-->
these words switch between real and protected mode after
DPMI is installed


Screen 32
\ ACTIVATE-DPMI
: .DPMI.VERSION ( DPMIversion --- )
    CR ." DPMI VERSION " DUP FLIP "FF AND U. ." ." "FF AND U. ;
          ( FLIP swaps bytes in a word, 12ABh -> AB12h e.g.)
: (ACTIVATE-DPMI)
    SHOW-SEGS
    \ get and save the entry point
    GET-DPMI-ENTRY modesw D!     \ #pars version
    \ announce DPMI version
    .DPMI.VERSION                \ #pars
    \ allocate memory requested by DPMI server
    MALLOC NIP                   \ seg
    \ and make the switch
    CR ." switching to protected mode "  modesw 2@ H. H.
    >PROTECTED-MODE
    SHOW-SEGS ;     -->


Screen 33
\ ACTIVATE-DPMI                                        RA 13NOV93

: ACTIVATE-DPMI
    \ save real mode segment for >REAL
    DS@ REAL-SEG !
    \ activate DPMI
    (ACTIVATE-DPMI)
    \ save protected mode segments for >PROT
    CS@ PROT-CS ! DS@ PROT-DS !
    SS@ PROT-SS ! ES@ PROT-ES !
    \ get and save raw mode switch addresses
    GET-RAW-MODES   P->R-SWITCH D!  R->P-SWITCH D!
    \ get and save state save addresses
    SETUP-STATE-SAVING
;           -->


Screen 34
\ SETUP WORD FOR PROTECTED 32-BIT SEGMENT       RA 03OCT93
HEX
: GRAB-32BIT-MEMORY
    \ get a chunk of 32-bit memory (requested in bytes)
    0000.F000   GET-32SEG   \ -- code-sel data-sel dHand
    \ save handle to the allocated memory for release later
    F-MEM-HAND 2!
    \ save data selector for memory block
    IS DATA-SEL
    \ and the code selector
    IS CODE-SEL
    \ and announce them
    CR ." CODE SELECTOR: " CODE-SEL .DESC
    CR ." DATA SELECTOR: " DATA-SEL .DESC
;
DECIMAL -->
```

One of the interesting features of this project is that the two Forths I used were so different from each other: on the one hand, a fairly traditional indirect-threaded 16-bit Forth-83 with the top-of-stack on the stack; on the other a quirky, compact, self-optimizing subroutine-threaded Forth with top-of-stack in a register. Interfacing these two created some interesting problems. I should add that the Forth I am most familiar with, the one I get paid to rewrite and use, is a four-segment (code, lists, data, headers) direct-threaded 16-bit model distantly based on MVP Forth, a hybrid of Forth-79 and Forth-83, where PICK and ROLL are zero-based, but where TRUE is 1 and LEAVE doesn't immediately.

One interfacing problem involves the differing stack widths for parameter passing between the two Forths. This problem cannot be ignored, but it mostly requires attention to the distinction between Intel and Forth word order in 32-bit values: when the 32-bit Forth places a 32-bit value on the stack, it appears in the 16-bit Forth with the low word on top, the reverse of Forth order; and when the 16-bit Forth passes a 32-bit value to the 32-bit Forth, it has to perform the equivalent of a 0 SWAP (for unsigned values) or S>D SWAP (for signed values). This is one of those "endian" things.

The interfacing is implemented by launching the 32-bit Forth with a value (a token) on the stack and having the start-up word, instead of just executing COLD, execute a case statement (called DISPATCHER and coded for now as nested IFs) to branch based on the token it receives. The tokens are coded as named constants in both Forths, to minimize confusion.

The simplest case, GO-32, is coded to pass a token signifying "do-nothing" to the DISPATCHER, which then just prints a sign-on message and executes (QUIT): in other words, this case is just the normal launch of the 32-bit Forth, with the ability to call BACK to get back to the 16-bit side.

FIND-32 parses the input stream and passes the token for "get-cfa" on top of the string address (selector and offset) of the word it parses. Note that both the selector and the offset are extended to 32-bit values, even though the selector is "really" only 16 bits, so that each one occupies a separate stack word from the

perspective of the 32-bit Forth. DISPATCHER passes this string address to WORDFINDER, which moves the string from the 16-bit segment to the 32-bit segment and searches for it in the 32-bit dictionary. After WORDFINDER, DISPATCHER executes BACK to return to the 16-bit Forth with cfa on the stack. Thus, FIND-32 WORDS will return the cfa of the word that prints the dictionary in the 32-bit segment.

GO-32 does FIND-32 one better. It not only finds the cfa, but also executes it, so that GO-32 WORDS will print the 32-bit Forth dictionary.

GO-32 is actually composed of two words: FIND-32 and EXECUTE-32. For words you want to call more than once from the 16-bit side, you can call FIND-32 once, save the cfa, and later call EXECUTE-32 with that cfa on the stack. EXECUTE-32 passes the 32-bit cfa to the 32-bit segment, where DISPATCHER executes it and returns.

For passing data between the segments, DATA-PAD passes an address in the 16-bit segment that the 32-bit segment stores. (I used to have other names for these words—RABBIT, FERRET, FOX, COYOTE, CAMEL—but here I've opted for clarity over color.)

The final word in this suite of interface words is SET-CALLBACK, which, whatever its usefulness, was the most interesting to code. SET-CALLBACK passes two addresses in the 16-bit segment to the 32-bit Forth. One of these addresses is the cfa of a word in the 16-bit Forth and the other is the address of a "callback handler." DISPATCHER passes these addresses to REG-CALLBACK, which stores them. DO-CALLBACK is very much like BACK, in that it makes a long jump to an address in the 16-bit Forth segment, but the address it jumps to is the address of the callback handler, and it jumps there with the callback cfa on the stack.

Since the 16-bit Forth is indirect threaded, handling the callback on the 16-bit side is not straightforward. We have to execute the cfa, which is probably a colon definition, and then jump back to the 32-bit segment. With respect to normal Forth execution, the situation is inside-out: we have to start in code, execute a colon word, and return to code. The other order is normal: all colon definitions eventually get down to

```
Screen 35
\ SETUP WORD FOR PROTECTED 32-BIT SEGMENT       RA 030CT93

: SET-PROT
    \ do the basic DPMI switch
    ACTIVATE-DPMI
    .MEM-INFO
    \ grab memory block for our specific needs
    GRAB-32BIT-MEMORY
;
-->


Screen 36
\ CALL 32-BIT FORTH SEGMENT             19JUN94 RA 29OCT93
HEX HERE    \ routine to restore 16bit prot seg registers
    2E C, 8E C, 1E C, PROT-DS , \ restore 16bit prot DS selector
    8C C, D8 C, 8E C, C0 C, \ MOV AX,DS MOV ES,AX - restore ES
    REST-FREGS, C3 C,       \ SS same for 16 & 32 bit sides
CONSTANT RESTORE-PROT-REGS
: RESTORE-PROT-REGS, CALL, RESTORE-PROT-REGS REL-TARG, ;
CODE (CALL-F32)     SAVE-FREGS,
    BE C, SP0-32 ,          \ MOV SI,E000
    B8 C, HERE 0 ,          \ MOV AX,D-SEL  \ -- dsel
    8E C, C0 C, 8E C, D8 C, \ MOV ES,AX  MOV DS,AX
    9A C,                   \ CALL LONG     \ -- dsel
    HERE  0 , 0 ,           \ CALL ADDRESS  \ -- dsel call
    RESTORE-PROT-REGS,
NEXT, END-CODE DECIMAL
CONSTANT F32-CALL-ADDR          CONSTANT DATA-SEL-ADDR      -->


Screen 37
\ TARGET SYSTEM LOCATION AND ACCESS     08JUN94 RA 030CT93
DECIMAL

: (GO-32)   \ how to get there ( ... act-code --- ... )
    US>32
    &MSTART  CODE-SEL   F32-CALL-ADDR D!
    DATA-SEL            DATA-SEL-ADDR !
    CR ." GOING TO raFORTH: "   CR .STACK-INFO
    SP0 @ 0 SWAP \ on stack for 32-bit forth to use 6/5/94
    (CALL-F32)
    CR ." BACK FROM raFORTH: "  CR .STACK-INFO ;

DECIMAL       -->


Screen 38
\ MENAGERIE OF ACTION WORDS                         RA 08JUN94
  0 CONSTANT do_nothing
  1 CONSTANT get_cfa
  2 CONSTANT execute
  3 CONSTANT data
  4 CONSTANT callback
: GO-32         \ just go there
    do_nothing (GO-32) ;
: FIND-32       \ get 32-bit cfa        ( --- idCFA T | d F )
    DS@ US>32  BL WORD US>32    get_cfa (GO-32) OR FLAG ;
: EXECUTE-32    \ execute 32-bit cfa    ( idCFA --- )
    execute (GO-32) ;
: DO-32         \ both of the above
    FIND-32 IF EXECUTE-32  ELSE CR ." FIND-32 FAILURE " THEN ;
: DATA-PAD      \ pass data buffer addr ( address --- )
    DS@ US>32 ROT US>32 data (GO-32) ;     -->


Screen 39
\ FAKE FORTH WORD FOR CALLBACK USE              RA 08JUN94

: GO-BACK-UP
    R> DROP R> DROP R> DROP         \ "normalize" return stack
    GO-32 ;
HERE ' NOOP , ' GO-BACK-UP , CONSTANT WORM
-->
```

```
Screen 40
\ CALLBACK                                          RA 15JUN94
HEX ( id16bit-cfa --- )
\ coming from a 32-bit to a 16-bit view of the stack, the low
\ word of the 32-bit double is on top
HERE \ address of callback handler
    RESTORE-PROT-REGS,
    58 C,               \ POP AX=cfa callback (lower 16 bits)
    A3 C, WORM ,        \ MOV [mock-pfa],AX   (mock Forth word)
    58 C,               \ POP AX=0 (high 16 bits of callback)
    BE C, WORM ,        \ MOV SI,mock-pfa     (Forth IP)
    NEXT,
DECIMAL   CONSTANT (CALLBACK)

: SET-CALLBACK ( cfa --- )
    US>32  (CALLBACK)  US>32 callback (GO-32) ;     -->
-->


Screen 41
\ ASSURE-PROT   REDEFINED BYE           15MAR94 RA 16NOV93
ONLY FORTH ALSO DOS ALSO FORTH DEFINITIONS      DECIMAL
: ASSURE-PROT
    \ if modesw has been set DPMI has been called
    modesw @ FLAG
    \ so we just make sure we're in protected mode
    IF >PROT
    \ otherwise we activate DPMI which puts us in protected mode
    ELSE SET-PROT
    THEN ;
RE- : BYE
    \ if DPMI has been activated we have to execute BYE
    \ in protected mode for proper cleanup, AND
    \ BYE must execute INT 21h function 4Ch
    modesw @ IF >PROT THEN BYE ;           -->


Screen 42
\ LOAD-IMAGE                                    15MAR94 RA 16NOV93
ONLY FORTH ALSO DOS ALSO FORTH DEFINITIONS      DECIMAL
: (LOAD-IMAGE)  ( fbuf buffer offset --- )
    >R                          \ fbuf buf
    BEGIN
        2DUP 1024 READ?         \ fbuf buf flag
        DS@ PLUCK DATA-SEL R@   \ fbuf buf flag ds buf dsel off
        1024 CMOVEL             \ fbuf buf flag
        R> 1024 + >R
    0=  UNTIL
    R> DROP 2DROP ;
: LOAD-IMAGE
    ASSURE-PROT
    FLUSH OFBUF [COMPILE] FILENAME OFBUF FREOPEN 0. OFBUF SEEK?
    OFBUF 0 BUFFER 0 (LOAD-IMAGE) OFBUF FCLOSE ;
```

RI32.SCR

```
Screen 60
\ DISPATCHER: WORDFINDER DISPATCH                RA 09JUN94
do_nothing US>D      MCONSTANT do_nothing
get_cfa    US>D      MCONSTANT get_cfa
execute    US>D      MCONSTANT execute
data       US>D      MCONSTANT data \ data blackboard
callback   US>D      MCONSTANT callback

M: WORDFINDER ( seg off --- cfa flag | x 0 )
    2DUP C@L 1+ DS@ HERE ROT
    HEX
    M." ABOUT TO CMOVEL " \ .S
    CMOVEL HERE FIND M;

M: DISPATCH ( cfa --- )
    M." ABOUT TO DISPATCH " HEX .DEPTH  KEY DROP EXECUTE M;
-->
```

lists that include code words. The solution I found was to create a two-word sequence consisting of NOOP and a word called GO-BACK-UP, which pops some things off the return stack and returns to the 32-bit segment. The callback handler, then, takes the 16-bit cfa it finds on the stack, shoves it into memory in place of the NOOP, points the Forth IP at it, and executes NEXT. The Forth indirect-threaded virtual machine then just executes the list fragment, which is now the callback word, and the word to return to the 32-bit Forth—and that's it.

As written, SET-CALLBACK provides for registering only one callback address at a time, but it would be easy to extend this to multiple callbacks through an array on the 32-bit side or even by passing names and creating actual Forth words in the 32-bit side that call 16-bit code.

...it allows the use of familiar tools without having to rewrite or port them to the 32-bit level.

Finally

What I have, then, at this stage, is a 16-bit Forth (Guy Kelly's), some words to enter protected mode via DPMI and to call a number of DPMI functions, and a target compiler which creates a 32-bit, subroutine-threaded Forth (based on Robert Illyes' riFORTH): something like off-the-shelf components. The 32-bit Forth, while still primitive, is capable of compiling from source files as well as the command line, and of saving its executable image to disk. If this seems a bit ornate, it's really not: every 32-bit program that runs on DOS has to have a 16-bit component to get it started; in this case, the 16-bit component happens also to be a Forth which runs in both protected and real modes. The ability to switch among these three modes, difficult to imagine in other languages but almost unavoidable in Forth, is one of the most interesting features of all this, particularly since it allows the use of familiar tools without having to rewrite or port them to the 32-bit level.

What I hope I've been able to demon-

strate, however, is not the creation/porting of a particular Forth in/to 32-bit protected mode, but a straightforward technique for getting there. Since all of it is written in one Forth or another, you don't have to use an assembler or a linker or a protected mode program loader to do it. All you need, though this is not nothing, is a 386 or higher DOS machine and a memory manager that supports DPMI.

Bibliography

Ray Duncan, editor. *Extending DOS: A Programmers Guide to Protected-Mode DOS.*

Robert L. Hummel. *PC Magazine Programmer's Technical Reference: The Processor and Coprocessor.*

Al Williams. *DOS 5: A Developer's Guide.* (A later edition, following MS-DOS major revision numbers, is current, but this is the edition I own.)

Al Williams. *DOS and Windows Protected Mode: Programming with DOS Extenders in C.*

DOS Protected Mode Interface (DPMI) Specification. This is available for versions 0.9 and 1.0 from Intel, order numbers 240763-001 and 240977-001, respectively. Windows 3.1 partially implements version 0.9, QEMM currently implements version 0.9, and 386MAX currently implements version 1.0. I use functions from version 0.9 that are also available in 1.0.

This may be an idea whose time has passed, but it is one that to my knowledge hasn't been sufficiently explored.

```
Screen 61
\ DISPATCHER: REG-CALLBACK                          RA 15JUN94

MVARIABLE CALLBACK-HANDLER
MVARIABLE CALLBACK-CFA \ later they will get names &C

M: REG-CALLBACK ( cfa callback-handler --- )
    CALLBACK-HANDLER !
    CALLBACK-CFA !
    CR M." CALLBACK HANDLER AT " CALLBACK-HANDLER @ H.
    CR M." CALLBACK CFA =      " CALLBACK-CFA     @ H.
M;
-->


Screen 62
\ FINAL WORDS: BACK DO-CALLBACK          15JUN94 RA 02OCT93
HEX | BACK. \ long jump back to calling 16-bit segment
    EA MC, MHERE 00.00 MD,  MHERE 00    MW, ret
    S>D MCONSTANT RETSEG  S>D MCONSTANT RETOFF
M: BACK    .STACKS DUP BACK. M;  \ push TOS to hardware stack
HEX | CALLBACK. \ long jump back to calling 16-bit segment
    EA MC, MHERE 00.00 MD,  MHERE 00    MW, ret
    S>D MCONSTANT CBSEG  S>D MCONSTANT CBOFF
M: SETUP-CALLBACK-ADDRESSES
    CALLBACK-HANDLER @ CBOFF ! RETSEG @ CBSEG ! M;
M: DO-CALLBACK
    SETUP-CALLBACK-ADDRESSES
    CALLBACK-HANDLER @ CBOFF ! RETSEG @ CBSEG !
    CALLBACK-CFA @   0  CALLBACK.   M; -->


Screen  63
\ DISPATCHER: IN PLACE OF COLD ON ENTRY 30OCT93 RA 02OCT93
M: DISPATCHER
    CR  DECIMAL   EMPTY-BUFFERS  \ ZERO removed 6/7/94
    CR M." ENTRY CODE IS: " DUP .     .STACKS   SHOW-SEGS
    DUP do_nothing =
    MIF DROP CR M." 32-BIT FORTH DISPATCHER " CR  (QUIT) MELSE
    DUP get_cfa =
    MIF DROP CR M." WORDFINDER " WORDFINDER BACK MELSE
    DUP execute =
    MIF DROP CR M." DISPATCHER " DISPATCH   BACK MELSE
    DUP data    =
    MIF DROP CR M." DATA PAD " H.  BACK MELSE \ not done yet
    DUP callback =
    MIF DROP CR M." CALLBACK REG " REG-CALLBACK (QUIT) MELSE
            CR M." UNKNOWN CODE "          BACK
    MTHEN MTHEN MTHEN MTHEN MTHEN M;  -->


Screen 64
\ FINAL WORDS: PRE-COLD                  07JUN94 RA 02OCT93
HEX
| PRE-COLD M]   \ careful to maintain same param stack as 16bit
    DROP \ get top stack element into TOS register
    DUP RETOFF W! HIWORD RETSEG W!  \ store return addr in BACK
        4 + SP0 ! \ value passed on stack from 16-bit side
    RP@ RP0 ! \ SP@ SP0 !
    .STACKS CR .REGISTERS CR
    NORM-SP
    .STACKS \ .DEPTH
    \ either DISPATCHER or COLD called here
    DISPATCHER M;
 \  COLD M;
DECIMAL -->
```


Richard Astle has a Ph.D. in English Literature and nine years of database programming in Forth. He can be reached via e-mail at rastle@aol.com and 76450.16@compuserve.com and will be glad to mail the complete source code to anyone who sends a disk and a self-addressed, stamped mailer to him at P.O. Box 8023, La Jolla, California 92038-8023.

A Forth-Oriented Compiler Compiler and its Applications

Mati Tombak, Viljo Soo, Jaanus Pöial
Tartu University, Estonia

The stack-oriented languages (Forth, PostScript, etc.) are often used as intermediate or target languages in software systems because of their portability, flexibility, compactness, and simplicity. In the field of compiler compilers, the concept of a "virtual stack machine" is often used to describe the source language semantics and program interpretation. Unfortunately, every author uses his/her own stack machine. The main idea of our approach is to use the real, widely known, and standard language (Forth-83) in the role of intermediate code in the compilers. The extensibility of Forth allows one to build a virtual machine with a problem-oriented instruction set as the source language.

The compiler compiler TARTU is fully Forth oriented (i.e., Forth is the implementation language of the system and the target language of the compilers written in TARTU). This choice determines formalism for the description of the source language semantics. In the TARTU system, the syntax of the source language must be described conventionally by means of context-free grammar. The (1,1)-DMSP parsing method is used (a generalization of mixed strategy of precedence). In principle, we may use any bottom-up method.

The description of the semantics consists of:

1. Translation from the source language into Forth, which must be described by a special kind of syntax-directed translation scheme.
2. Semantical analysis (context checking) and "threaded code" generation, which must be described in Forth, using the fact that Forth is a one-pass macroprocessor for itself (IMMEDIATE words).
3. Proper semantics (run-time semantics), which must be described as an extension of Forth.

The following example demonstrates the metalanguage of the TARTU system for writing the translation scheme. We use the "standard" lexical analyzer of ALGOL-like languages with predefined classes of tokens.

[I] — class of tokens "identifier,"
[C] — class of tokens "unsigned integer,"
[<*delimiter*>S<*delimiter*>] — class of tokens "string," e.g., `[{S}]`.

Instances of these classes are translated one to one (textually). Other Forth words are generated using the so-called translation components which are connected with the grammar rules (non-terminal nodes of the syntax tree). *[See Figure One.]*

This translation scheme generates the Forth text in the following way:

Figure One.

```
#REG xxx#          regime of translation xxx (starts the text generation
                   from the subtree of the syntax tree),
#PERM k1...kn #    permutation of subtrees,
#POST ...#         postfix component of the text generation,
#PRE ...#          prefix component of the text generation.

=DEF= prog
=COMMENT= [{S}]
     prog ==> stms  #REG TRANS# ;
     stms ==> stm /
              stms stm ;
     stm  ==> 'VAR' [I]  #PRE VARIABLE# /
              'PRINT' exp  #POST .# /
              id ':=' exp  #PERM 2 1 #  #POST !# ;
     id   ==> [I] ;
     exp  ==> mon /
              exp '+' mon  #POST +# ;
     mon  ==> term /
              mon '*' term  #POST *# ;
     term ==> [C] /
              '(' exp ')' /
              id  #POST @# ;
=END=
```

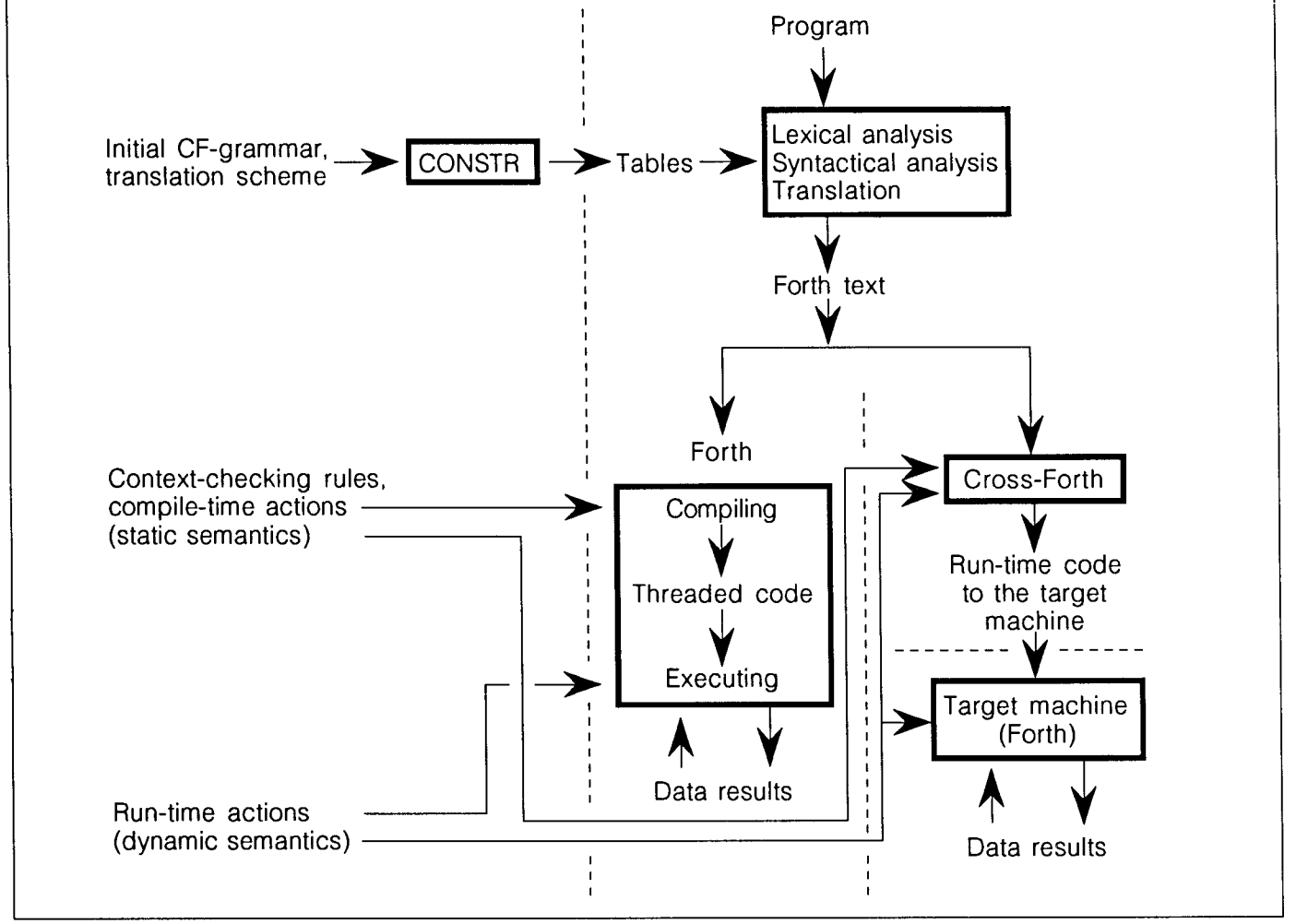
Source text	*Forth text*
`VAR C`	**`VARIABLE C`**
`C:=1+1`	**`1 1 + C !`**
`PRINT C+2`	**`C @ 2 + .`**
`PRINT 65536*2856`	**`65536 2856 * .`**
`PRINT (2+3)*4`	**`2 3 + 4 * .`**

The Forth text may be directed to the standard Forth system or to a cross-system for another target machine. The relative simplicity of cross-compiling in Forth allows one to use the TARTU system for real applications. The authors of this paper have implemented Fortran-IV, Modula-2, and a few special languages for different machines.

It is a most inconvenient task to express the context checking/code generation in terms of `IMMEDIATE` words in real translators. The example given above was too simple to reflect these problems. *[See Figure One.]*

The compiler compiler TARTU now works on IBM PC/XT/AT-clone computers under our own 32-bit implementation of Forth (Forth-83/32).

Compiler-compiler TARTU



Prof. Mati Tombak (1942): Professor of Theoretical Computer Science, Department of Computer Science, University of Tartu, Estonia. Graduated from University of Tartu as a mathematician in 1969. Ph.D. in mathematics from St. Peterburg (Russia, former Soviet Union) in 1981. E-mail: mati@cs.ut.ee

Viljo Soo (1958): Senior Research Fellow in EENet (Estonian Education and Research Network). Graduated from University of Tartu as a mathematician in 1982. E-mail: viljo@eenet.ee

Jaanus Põial (1959): Associate Professor, Department of Computer Science, University of Tartu. Graduated from University of Tartu as a mathematician in 1982. Ph.D. in computer science from Tallinn in 1987. E-mail: jaanus@cs.ut.ee

Compiler-compiler project in Forth: 1983–1991.

Forth2LaTeX— a Pretty-Printer

Ronald T. Kneusel
Milwaukee, Wisconsin

Forth code is beautiful. A listing of Forth reads like a poem, according to Chris Heilman, author of Pocket Forth. Sadly, Forth's appearance is drab. It resembles a newspaper editorial, a Final Demand notice, or COBOL code. I decided that Forth deserved better. Its beauty should shine through even a printout, I thought. So I wrote Forth2LaTeX. It wasn't all altruism: I needed to dress up another project of mine. But it was a fine by-product.

Forth2LaTeX brings out the beauty of Forth code by transforming it into LaTeX, a variation of Donald Knuth's famous TeX typesetting system. I've seen other pretty-print programs, but none for Forth, and I haven't searched for one. My hobby is re-inventing the wheel.

Forth2LaTeX is meant for anyone who would like to create eye-catching source-code listings, though professionals and academics will be its major users. The program:

- sets code and comments in different fonts,
- numbers lines,
- highlights programmer-defined words,
- generates an alphabetical index of word definitions and associated line numbers,
- makes use of LaTeX's sectioning abilities, to create either an article (for shorter programs) or a report with chapter headings,
- outputs a table of contents, and
- lets the programmer plant LaTeX code within the Forth code.

Additionally, Forth2LaTeX permits straightforward text within the Forth source code. A programmer can write source code that runs and generates its own formal report when finished.

The Forth2LaTeX prototype is a Modula-2 application for the Apple Macintosh. Why not Forth? Modula-2 was faster to start with, and friendlier with the Macintosh GUI interface. However, for maximum portability, final versions will be in C.

A nice feature of Forth2LaTeX is that it knows nothing of Forth other than what a comment is, and how words and other structures are created via the words : (colon), VARIABLE, CONSTANT, and CREATE. This makes it virtually universal, untied to any particular Forth dialect.

Run the program, and it makes two passes through the source code. One gathers Forth2LaTeX settings from comment statements, the other does the actual translation. Choose your settings with command-line switches, or menu selections on the Mac. Settings within the source code as Forth comments are also supported.

Most lines in the source code are simply copied to the output file, character by character, after passing through a mapping that converts lowercase to uppercase (if enabled), and converts special LaTeX characters to their equivalents. For example, the $ character is used by LaTeX to begin inline mathematics mode. So each occurrence of $ in the input code must be replaced by \$ to make LaTeX generate the correct character. LaTeX removes extra spaces unless they are preceded by a \ character. Forth2LaTeX clutters the output code with unpoetic \ characters, but they preserve the indentations set by the programmer. Code itself is set in LaTeX's monospaced typewriter font, to ensure correct formatting.

Lines of source text that mark specific commands or sections are treated differently from lines of pure Forth code. The first pass through the code finds all parameter settings. These lines are either copied as-is, or are removed from the output. For example, to credit the original programmer, you write a comment line of the form:

```
\ Author: Joe Schlabotnik
```

This provides information for the title page and is echoed in the output. However, a line to mark a section of code, such as

```
\ Section: Calculate the Eigenvalues
```

won't appear in the output.

If the line contains a definition of some kind, it's placed into a binary tree, along with the line number for later output in the index. If boldface is enabled, the name is copied to the output within a {\bf }.

Like LaTeX, Forth2LaTeX can chop a document into chapters (reports only), sections, subsections, and subsubsections. It is this sectioning ability, plus the capacity to make comments of the form \ .[text] typeset as normal, that inspired Forth2LaTeX.

Figure One is a Forth program which estimates the square root of an integer. The code contains the Forth2LaTeX settings within comments. The only comment that Forth2LaTeX itself notices is the \ variety,

Figure One. Forth code with Forth2LaTeX settings.

```
\ Program: Square Root                                                    (a)
\ Author: Ronald T. Kneusel
\ Started: 10/24/94
\ Modified: 10/29/94
\ Modify By: RTK
\ Summary: Estimates the square root of an integer using integer arithmetic
\ Comments: FORTH                                                         (b)
\ (this is the default, the other option is LATEX)
\ Style: REPORT
\ (switch from ARTICLE to REPORT so we can use chapters)
\ Index: ON
\ (create an index of programmer defined words)
\ Bold: ON
\ (bold word names when defined)
\ Left Margin: -0.5in
\ (default is 1in margin, this adds a negative offset to get 0.5in margins)
\ Chapter: Estimate the Square Root of an Integer                         (c)
\ .
\ .  This text will be typeset as normal and will make it appear that the
\ .  source-code is pasted into the text that makes up the article or report.
\ .  This program will estimate the square root of an integer, usually to
\ .  within 5 percent of the actual value.
\ .
\ Section: The SQRT function
0 variable n
: sqrt  ( n -- frac sqrt )                                                (d)
  \ Subsection: Initialize count
  dup n ! 1  \ initial count
  \ Subsection: Subtract odd integers while positive
  32750 1 do
    swap i - dup 1 < if
     i + 10000 n @ */ swap \ estimate decimal
     dup dup * n @ = if
      swap drop 0 swap  \ no estimate if exact
     else
      1-  \ correct inexact result
     then
     32750  \ done
    else
     swap 1+  2  \ increase count
    then
  +loop ;
\ Section: The MAIN function
: main  ( -- )  \ table of square roots from 500 to 1000
  ."  X   Square root X" cr
  1001 500 do
    i dup .  space sqrt .  46 emit .  cr
  25 +loop ;
```

reserving (...) delimited comments for stack effects.

The comments in part (a) of Figure One give Forth2LaTeX information that will be printed on the listing's title page. Since the program makes two passes, it does not matter where these or any other settings are placed. Put them at the end, if you wish, or omit them.

Part (b) shows how settings are placed within the file to control Forth2LaTeX. The `Comments` setting determines how Forth2LaTeX treats comments within the code. If set to `LATEX` there will be no mapping of characters within comments, allowing the programmer to place actual LaTeX commands in the code. The `Left Margin` setting may seem odd, but LaTeX defaults to one inch down and one inch to the right, so changes to these settings must be made as offsets. Thus, to set the left margin at 0.5 inch, add an offset of -0.5 inch.

Part (c) marks the beginning of a new chapter. Because we switched Style to `REPORT`, chapters are permitted; they are ignored in `ARTICLE` format. The lines below this are comments of the form `\ .[text]`, typeset as normal, with no mapping of characters: you can write in pure LaTeX. Also, enclose part of a comment in backquotes (`) and no mapping will occur.

Part (d) is the actual code for the square root program with sectioning added as an illustration.

Figure Two is the LaTeX code generated from Figure

Figure Two. LaTeX code generated from Figure One.

```
\documentstyle[12pt]{report}                                          (a)
...  page setup statements skipped ...
\begin{document}
\pagestyle{plain}
\pagenumbering{roman}
\title{{\bf  Square Root}}                                            (b)
\author{{\small  Ronald T.  Kneusel}}
\date{{\small \today}}
\maketitle
\vspace{4in} \hfil \break {\large{\bf Program Information:}} \hfil \break
\hfil \break
{\bf Summary:}\ \  Estimates the square root of an integer using integer arithmetic\hfil \break
{\bf Author:}\ \  Ronald T.  Kneusel\hfil \break                      (c)
{\bf Modified:}\ \  10/29/94\hfil \break
{\bf Modify by:}\ \  RTK\hfil \break
{\bf Lines:}\ \  46\hfil \break \clearpage
\tableofcontents \clearpage
\pagestyle{plain}
\pagenumbering{arabic}
\setcounter{page}{1}
\makeatletter
\def\@evenfoot{ }
\def\@evenhead{\hfil\thepage\hfil}
\def\@oddhead{\@evenhead}
\def\@oddfoot{\@evenfoot}
\makeatother
\small                                                                (d)
\leftline{{\tt0000\ -\ }{\it$\backslash$\ Program:\ Square\ Root}}
\leftline{{\tt0001\ -\ }{\it$\backslash$\ Author:\ Ronald\ T.\ Kneusel}}
\leftline{{\tt0002\ -\ }{\it$\backslash$\ Started:\ 10/24/94}}
\leftline{{\tt0003\ -\ }{\it$\backslash$\ Modified:\ 10/29/94}}
\leftline{{\tt0004\ -\ }{\it$\backslash$\ Modify\ By:\ RTK}}
\leftline{{\tt0005\ -\ }{\it$\backslash$\ Summary:\ Estimates\ the\ square\ root\ of\ an\ integer\
using\ integer\ arithmetic}}
\leftline{{\tt0006\ -\ }{\it$\backslash$\ Comments:\ FORTH}}
\leftline{{\tt0007\ -\ }{\it$\backslash$\ (this\ is\ the\ default,\ the\ other\ option\ is\ LATEX)}}
\leftline{{\tt0008\ -\ }{\it$\backslash$\ Style:\ REPORT}}
\leftline{{\tt0009\ -\ }{\it$\backslash$\ (switch\ from\ ARTICLE\ to\ REPORT\ so\ we\ can\ use\
chapters)}}
\leftline{{\tt0010\ -\ }{\it$\backslash$\ Index:\ ON}}
\leftline{{\tt0011\ -\ }{\it$\backslash$\ (create\ an\ index\ of\ programmer\ defined\ words)}}
\leftline{{\tt0012\ -\ }{\it$\backslash$\ Bold:\ ON}}
\leftline{{\tt0013\ -\ }{\it$\backslash$\ (bold\ word\ names\ when\ defined)}}
\leftline{{\tt0014\ -\ }{\it$\backslash$\ Left\ Margin:\ -0.5in}}
\leftline{{\tt0015\ -\ }{\it$\backslash$\ (default\ is\ 1in\ margin,\ this\ adds\ a\ negative\ offset\
to\ get\ 0.5in\ margins)}}
\chapter{ Estimate the Square Root of an Integer}                     (e)

 This text will be typeset as normal and will make it appear that the
 source-code is pasted into the text that makes up the article or report.
 This program will estimate the square root of an integer, usually to
 within 5 percent of the actual value.
```

(Figure Two continues on next page.)

One after it's been through Forth2LaTeX. The `table of contents` and `force uppercase` options were set via the application, not the code example. It's cluttered, but the final product is elegant (see Figure Three) and may be made more elegant by adding LaTeX code in strategic places. I've marked some key areas that are created by Forth2LaTeX and labeled them (a) through (f).

For example, Part (a) of Figure Two defines the document type, sets up the page, and starts the document. Part (b) contains the information that is used in the title page and part (c) has the program information. Part (d) begins the source code for the program. It continues after the text in part (e). LaTeX's natural unit is the paragraph, but it's not wanted here. To make LaTeX treat the code on a line-by-line basis the `\leftline` command is used. Initially, I used an `\hfil \break` combination, but this caused TeX's memory to be exceeded for all but the smallest programs. All code between the braces is set on a single line. For example,

```
\leftline{{\tt0027\ -\ \ \ \ \ \ SWAP\
I\ -\ DUP\ 1\ <\ IF\ \ }}
```

This line contains only Forth code and is set in typewriter font `\tt` with the line number indicated first. Note the many \ characters to force spaces to appear. This line:

```
\leftline{{\tt0025\ -\ \ DUP\ N\ !\ 1\ \
}{\it$\backslash$\ initial\ count}}
```

(Figure Two, continued.)

```
\section{ The SQRT function}
\leftline{{\tt0023\ -\ 0\ VARIABLE\ N}}
\leftline{{\tt0024\ -\ :\ {\bf SQRT}\ \ (\ N\ --\ FRAC\ SQRT\ )}}
\subsection{ Initialize count}
\leftline{{\tt0025\ -\ \ \ \ DUP\ N\ !\ 1\ \ }{\it$\backslash$\ initial\ count}}
\subsection{ Subtract odd integers while positive}
\leftline{{\tt0026\ -\ \ \ \ 32750\ 1\ DO}}
\leftline{{\tt0027\ -\ \ \ \ \ \ SWAP\ I\ -\ DUP\ 1\ <\ IF\ \ }}
\leftline{{\tt0028\ -\ \ \ \ \ \ \ \ I\ +\ 10000\ N\ @\ */\ SWAP\ }{\it$\backslash$\ estimate\
decimal\ }}
\leftline{{\tt0029\ -\ \ \ \ \ \ \ \ DUP\ DUP\ *\ N\ @\ =\ IF}}
\leftline{{\tt0030\ -\ \ \ \ \ \ \ \ \ \ SWAP\ DROP\ 0\ SWAP\ \ }{\it$\backslash$\ no\ estimate\
if\ exact}}
\leftline{{\tt0031\ -\ \ \ \ \ \ \ \ ELSE}}
\leftline{{\tt0032\ -\ \ \ \ \ \ \ \ \ \ 1-\ \ \ }{\it$\backslash$\ correct\ inexact\ result}}
\leftline{{\tt0033\ -\ \ \ \ \ \ \ \ THEN}}
\leftline{{\tt0034\ -\ \ \ \ \ \ \ \ 32750\ \ }{\it$\backslash$\ done\ }}
\leftline{{\tt0035\ -\ \ \ \ \ \ ELSE}}
\leftline{{\tt0036\ -\ \ \ \ \ \ \ \ SWAP\ 1+\ \ 2\ \ }{\it$\backslash$\ increase\ count}}
\leftline{{\tt0037\ -\ \ \ \ \ \ THEN}}
\leftline{{\tt0038\ -\ \ \ \ +LOOP\ ;}}
\section{ The MAIN function}
\leftline{{\tt0039\ -\ :\ {\bf MAIN}\ \ (\ --\ )\ \ }{\it$\backslash$\ table\ of\ square\ roots\
from\ 500\ to\ 1000}}
\leftline{{\tt0040\ -\ \ \ \ ."\ \ X\ \ \ \ Square\ root\ X"\ CR}}
\leftline{{\tt0041\ -\ \ \ \ 1001\ 500\ DO}}
\leftline{{\tt0042\ -\ \ \ \ \ \ I\ DUP\ .\ SPACE\ SQRT\ .\ 46\ EMIT\ .\ CR}}
\leftline{{\tt0043\ -\ \ \ \ 25\ +LOOP\ ;}}
\leftline{{\tt0044\ -\ }}
\appendix                                                    (f)
\chapter{Index of User-Defined Names}
\begin{tabular}{lllll}
{\rm MAIN}\ {\tt (0039)} & {\rm SQRT}\ {\tt (0024)} &  \\
\end{tabular}
\end{document}
```

contains a comment. Comments are set in italics `\it` so you must use the awkward `$\backslash$` string of characters to create the \ character: the $ starts LaTeX's math mode and makes the `\backslash` command available to print a \ character, and the last $ turns off the math mode. Spaces are escaped, since the comments setting is FORTH and not LATEX.

Part (e) shows what happens to `\ .` comments in the Forth source code. The text on these lines will be set as a paragraph in normal Roman font, which looks pleasing when printed. Part (f) sets the index of defined words as an appendix. The table is created in alphabetical order by the Forth2LaTeX program and uses the standard LaTeX tabular environment. `\end{document}` completes the translation.

Forth2LaTeX has no error-handling routine because it's not necessary. Most pretty-printers require syntactic correctness, but Forth2LaTeX just scans for special characters or predetermined strings. There is an upper limit on the size of the input and output string, but most programmers write code that fits in 80- to 132-character lines anyway.

Though the prototype of Forth2LaTeX is for Macintosh in Modula-2, I will port it to C for Unix, VAX/VMS, and MS-DOS to make it as universal as possible. There are many features that could, and likely will, be added, including alternate fonts for code and comments, highlighting of programmer-defined words, and the ability to include other source-code files. The most recent version of the program is available from me via e-mail.

Forth2LaTeX acts like a compiler for LaTeX, itself a language for typesetting documents. I've had a Forth2Postscript translator suggested to me, but the Forth2LaTeX translator is already the "front end" of a Forth2Postscript translator with LaTeX as the intermediate language, and LaTeX itself as the "back end."

The ability to include LaTeX in the code creates some interesting possibilities. For example, inclusion of figures and graphics through the `\special` command, available on some LaTeX implementations like OzTeX for the Macintosh, or the automatic generation of footnotes, or a bibliography. It might even be possible to reverse Forth expressions in `postfix` and output them in LaTeX's math mode in `infix` notation for easier reading. The addition of structured code "filters" to impose a form of structured coding style on the output is another possibility...

As someone once might have said, "Of the writing of Forth code, and Forth tools, there will be no end..."

Ronald T. Kneusel usually works in Pocket Forth for the Macintosh, but is in the midst of a computer algebra project in Forth using Yerk. He is trying to keep the code fairly portable, and promises to put it on the Internet when completed. He has used Forth to write a small IFS fractal generator and a microcomputer simulator. Both are available via FTP from mac.archive.umich.edu as /mac/graphics/fractal/fractallabkit and /mac/development/languages/mdp80v1.3.sit.hqx. Kneusel is just starting a Ph.D. program in mathematics and computer science in at Marquette University. To obtain information about how to acquire the latest version of Forth2LaTeX, he can be contacted by mail at 8725 West Burdick Avenue, Milwaukee, Wisconsin 53227 U.S.A., and by e-mail at kneusel@studsys.mscs.mu.edu.

Figure Three. Final LaTeX output.

```
00000 - \ Program: Square Root
00001 - \ Author: Ronald T. Kneusel
00002 - \ Started: 10/24/94
00003 - \ Modified: 10/29/94
00004 - \ Modify By: RTK
00005 - \ Summary: Estimates the square root of an integer using integer arithmetic
00006 - \ Comments: FORTH
00007 - \ (this is the default, the other option is LATEX)
00008 - \ Style: REPORT
00009 - \ (switch from ARTICLE to REPORT so we can use chapters)
00010 - \ Index: ON
00011 - \ (create an index of programmer defined words)
00012 - \ Bold: ON
00013 - \ (bold word names when defined)
00014 - \ Left Margin: -0.5in
00015 - \ (default is 1in margin, this adds a negative offset to get 0.5in margins)
00016 - \ Chapter: Estimate the Square Root of an Integer
```

This text will be typeset as normal and will make it appear that the source code is pasted into the text that makes up the article or report.

This program will estimate the square root of an integer, usually to within 5 percent of the actual value.

1 The SQRT function

```
00024 - 0 VARIABLE N
00025 - : SQRT  ( N -- FRAC SQRT )
```

1.1 Initialize count

```
00026 -    DUP N ! 1  \ initial count
```

1.2 Subtract odd integers while positive

```
00027 -    32750 1 DO
00028 -      SWAP I - DUP 1 < IF
00029 -        I + 10000 N @ */ SWAP \ estimate decimal
00030 -        DUP DUP * N @ = IF
00031 -          SWAP DROP 0 SWAP  \ no estimate if exact
00032 -        ELSE
00033 -          1-   \ correct inexact result
00034 -        THEN
00035 -        32750  \ done
00036 -      ELSE
00037 -        SWAP 1+  2  \ increase count
00038 -      THEN
00039 -    +LOOP ;
```

2 The MAIN function

```
00040 - : MAIN  ( -- )  \ table of square roots from 500 to 1000
```

Using Zeller's Congruence

to Calculate the Weekday from a Date

Walter J. Rottenkolber
Mariposa, California

If you are like me, you might remember the date of an important event, but do you know what day of the week it was? If you need help, calculate the day of the week with Zeller's Congruence.

The formula is:

DW = (⌊2.6 * m# - 0.2⌋ + d + h + ⌊h/4⌋ + ⌊c/4⌋ - 2*c) MOD 7

where

DW = day of week (Sunday = 0...Saturday = 7)
m# = month number (March = 1...February = 12)
d = day of month
c = century
h = hundreds in year

Note: ⌊x⌋ = truncated integer

The month number begins with March as 1, and ends with February as 12. The word M# does the conversion. For the calculation, January and February are included with the previous year to simplify dealing with leap years. The first line of CALCDW does this automatically.

This calculation is for the Gregorian calendar only. It will be invalid for other calendars, such as the Julian. The problem is compounded because, although officially the Julian calendar ended on Thursday, October 4, 1582, and the Gregorian calendar began on the next day, Friday, as October 15, 1582, it did so only in the Catholic countries. The non-Catholic countries converted it gradually, at irregular intervals. In England and the colonies, it did not occur until 1752.

DW takes the date and prints the day of the week. The date must be entered as: month day year. The year must be the full four numbers, e.g., 1994, not an abbreviation.

Example:

```
4 19 1994 DW <cr> = Tuesday  ok
```

DD prints a list of days for a range of dates in a month and year set by SETMY.

Example:

```
3 10 DD --> list.
```

F13 prints those months in a given year containing a Friday the 13th.

I did spot checks for accuracy and the formula seems to work, but a lot of days have passed through the centuries, so I can't guarantee the results. Even so, when I found discrepancies, it was the reference dates that were in error.

Walter J. Rottenkolber bought his first computer in 1983. Early on, he experimented with fig-Forth and other languages, but gravitated to assembler until reintroduced to Forth in 1988. He notes that Forth provides the same close-to-the-silicon feeling as assembler, but without the pain. Interests include small embedded systems, programming, and computer history, about which he enjoys writing.

Example: for `4 19 1994 DW <cr>`

```
DW = ((2.6*2-0.2) + 19 + 19 + 94/4 + 19/4 - 2*19) MOD 7
     (    5       + 19 + 94 +  23  +  4   -  38 ) MOD 7
                                      107   MOD 7
DW = 2
DW = Tuesday
```

```
      1
0 \ Day of Week
1 : .DW$  ( n)  \ n= Sun=0...Sat=6
2     " Sun   Mon   Tues  WednesThurs Fri   Satur "
3     DROP SWAP  6 * +  6  -TRAILING  TYPE ." day" ;
4 \S
5       Day of Week uses Zeller's Congruence
6
7   DW = (|2.6*m# - 0.2| + d + h + |h/4| + |c/4| - 2c) MOD 7
8
9   where
10    DW = Day of Week  (Sun=0..Sat=6)
11    m# = month number (Mar=1..Feb=12)
12     d = day of month
13     c = century
14     h = hundreds in year
15  Note: |x| = truncated integer

      2
 0 \ Day of Week
 1 : M#  ( n - n)  2- DUP 1 ( IF 12 + THEN ;
 2 : CALCDW  ( m d y - n)
 3     2 PICK M# 10 ) IF 1- THEN
 4     100 /MOD            ( y= h c )
 5     DUP 2*        )R    ( 2c )
 6     4 /           )R    ( c/4 )
 7     DUP 4 /       )R    ( h/4 )
 8                   )R    ( h )
 9                   )R    ( d )
10     M#  26 * 2 - 10 /
11     R) + R) + R) + R) + R) - 7 MOD ;
12 : DW  ( m d y)  CALCDW  ." = " .DW$ CR ;
13 \S Gregorian Calendar: m= month, d= day, y= year (4 numbers)
14
15


      3
   \ Month's With Friday the 13th in Year.
   VARIABLE MO      1 MO !
   VARIABLE YR   1809 YR !
   : SETMY  ( mo yr)  YR ! MO ! ;
   : .DATE  ( mo da yr)  SWAP ROT . . . ;
   : .MO$  ( n)  \ n= Jan=0..Dec=11
      DUP 11 U) ABORT" Month number error."
      " January  February March    April    May      June     "
      " July     August   SeptemberOctober  November December "
      DROP NIP ROT DUP 6 ( IF NIP
      ELSE 6 - ROT DROP THEN  9 * + 9 -TRAILING TYPE ;
   \S


      4
   \ Day of Week
   : DD  ( d1 d2)
      CR MO @ .MO$ ." , " YR @ .
      CR 1+ SWAP DO
      MO @  I YR @  I 2 .R DW LOOP ;
   \ DD generates a Date = Day list from d1 to d2.
   \   Use SETMY to set the month and year.

   : F13  ( yr)
      YR !  CR ." Months with Friday the 13th in " YR @ .
      13 1 DO
      I 13 YR @ CALCDW 5 = IF
      CR 8 SPACES I 1- .MO$ THEN LOOP CR ;
```

euroForth '94

Gordon Charlton
Hayes, Middlesex, England

The euroForth '94 conference kicked off with an "in at the deep end" session. Anton Ertl, the first speaker and my traveling partner, continues to make a comprehensive evaluation of stack caching techniques for virtual stack machine interpreters, looking at both dynamic and static stack caching. Static stack caching is the technique of optimising away stack operators at compile time. Dynamic caching is the idea that a map of the top few elements of the stack could be kept at run time, each change in the map represented by a transition from one inner interpreter to another, each optimised for that particular mapping. Anton's research is ongoing. He hopes to set up a system to get some empirical results to back up his theoretical findings.

Michael Gassanenko, from the St. Petersburg Institute for Information and Automatics, followed with a description of a novel and ingenious way of resolving the addressing problems that 8086 paged memory gives, which allows 32-bit addressing to be used and arithmetic

Winchester, the ancient capital of England. A medieval city of old picturesque buildings and narrow streets...

applied to addresses at a cost of only six additional clock cycles in the overwhelming majority of cases. Very neat.

Chris Bailey, a canny Newcastle lad from the University of Teeside, finished the session by summarising his work studying the effects of local variable optimisation on a C-based, stack-processor environment. The question he asks is, how should the architecture of a stack machine be arranged to best allow compiled C programs to run on it, given the assumption that the compiler will attempt to optimise local variables by treating them as stack entities? A valid question, if stack processors are to gain significant market penetration. Currently, he has devised some neat optimisation strategies to investigate, and initial results suggest that a larger stack cache is indicated than for applications written in Forth.

After coffee and Diwali sweetmeats (it being that time of year), Graeme Dunbar from the Robert Gordon University in Scotland described his investigations into using Register Transfer Language (RTL) for coding a Forth kernel. He summarised previous efforts to give an unambiguous and simple description of a Forth kernel. Current work is on a FIG kernel, but in the Forth tradition of throwing out the first attempt and doing over, he is giving serious consideration of starting again using ANS Forth as a basis.

Anton Ertl returned with a well-argued case for releasing some of the restrictions ANS places on local variables, such as not declaring them within control structures, in his paper "Automatic Scoping of Local Variables."

The morning sessions were rounded off by Michael Gassanenko, whose second paper, "BacForth: an Approach to New Control Structures" demonstrated not only how simply Forth could be extended to include backtracking, but how the technique could simplify a large class of programming problems very neatly. He also suggested a stack notation for a backtracking Forth that impressed me particularly, as I had been unable to find a neat solution when I investigated backtracking in the area of pattern matching a few years ago. Michael's two papers, taken together, were one of the prize winners for "best paper" this year.

Lunch was a welcome respite (for this reporter) from the mental workout, but we were soon back into the highbrow stuff with the Formal Methods session, and Jaanus Pöial (associate professor at the University of Tartu in Estonia) telling us that stack comments are a good idea! Actually he said rather more than that; it went along the lines of "Stack comments can be shown to be a valid algebra and are mathematically rigorous, so may form the basis for a formal description of Forth." I do not claim to understand the demonstration, but can see the significance of the outcome.

This was taken a little further by Bill Stoddart, from the University of Teeside in Middlesbrough (coincidentally

my home town), who has been working in similar areas to Jaanus but with a different notation, who gave a work-in-progress paper explaining the whys, and some of the hows, of producing a formal equivalent of the ANS standard. The whys are simple; given that English is both vague and ambiguous, it is rather difficult to determine if a standard written in English is consistent, which is a virtue one may reasonably demand of a standard. The how is using "Abstract Machine Notation," a relatively unpublicised development from Jean Raymond Abrial, who Bill assures us is a genius at that sort of thing. AMN is related to B, which is a successor to Z.

After dinner (more standard English hotel fare; edible and really not bad, but embarrassing to those of us who take pride in English cuisine), Roy Goddard from MPE (our hosts), honorary clown prince of proceedings, took to the stage, ably assisted by his assistant, The Lovely Gary (as one wag from the audience dubbed him), to present the traditional "silly competition." As Roy has a style of humour that does not bear repeating (Roy: I mean it's inimitable, not unrepeatable), I won't attempt to. The silly competition was to devise a Forth program that hinted at a film, play, or book, very much as one may in the game of charades. The example Roy gave (the one I remember) was:

```
: BOOK  22 THROW ;
```

And for the illiterate amongst us, the tome in question was *Catch-22!*

I am sorry to report that Saturday morning was a little blurred for this reporter, one of the few lacking the sense to grab an early night, yet still, inexplicably, being the first to rise. As Vivian Stanshall says in the role of Sir Henry at Rawlinson End, "I always sleep *tight*."

Nonetheless, Ben Campbell gave a bouncingly enthusiastic report of how to use Forth to gain dominance in niche markets without, as he candidly notes in his abstract, understanding `CREATE DOES>`. The basic premise was, find a market small enough to generate personal relationships with all your customers, let them tell you what they want, and demonstrate that you can produce results. And pick an area that you find interesting, otherwise you won't have the enthusiasm required. Don't waste time on poor leads, culture good ones. Sort of "`CREATE opportunity DOES> $$$ @ ;`".

Stephen Pelc of MPE was pleased to report that, at a recent meeting with IBM, Seimens, and ARM, amongst others, the general consensus was that traditional in-circuit emulation was ceasing to be feasible, basically because of the raw speed of modern processors, so interactive debugging environments would become imperative. We all know what language is the perfect choice.

Next, Howard Oakford from Inventio Software proposed a method for quality control in Forth development. By then, I was drifting off, so do not recall too much, apart from a very humorous diatribe in which Howard claimed to like C very much, as it allowed him to blame everything on the compiler writer without fear of comeback.

Larry Forsley (Dash, Find Associates, and The Forth Institute) spoke lucidly about the philosophical aspects of the Forth paradigm in "Rhyme, Reason and the Tao of Forth." Whether you agree with his vision of Forth or not, he is a speaker worth hearing.

Session five was Embedded Applications, an area well outside of my field, so I will not try to do justice to the speakers here.

Briefly, Malcom Bulgar described his frustrating experiences with Echelon—a basically neat idea for a product crippled by the compromises made in the implementation. Naturally Forth comes to the rescue!

Tim Hendtlass (from Australia—whoever said that euroForth was parochial!) was a delight with his vivid imagery of elderly folk encountering technology, and how Forth could be used to maximise the usefulness of technology in providing a safer home environment whilst minimising intrusiveness, if only the State cared enough to fund it. Some problems are universal. This was another prize-winning talk.

Sergei Baranoff (also from the St. Petersburg Institute for Information and Automatics) delivered a paper entitled, "A Model of a Real-time Executive on Forth," but your foolish reporter forgot to take any notes, and his paper did not arrive in time to make it into the proceedings. Sorry.

Session six was MPE's and Forth Inc.'s opportunity to showcase their new products in the GUI and Professional Products part of the conference. I was pleased to discover that, in person, Elizabeth Rather showed the same clarity and sureness that she displays in her postings to comp.lang.forth; and EXPRESS, Forth Inc.'s GUI Process Control Software, is impressive.

Not to be outdone by Elizabeth's walk-through of a demo running under EXPRESS, Roy and the lovely Gary from MPE elected to demonstrate proForth for Windows by using GUIDE (stands for GUI DEsigner, or something like that) to create a window, complete with menus, submenus, and a button that printed a "hello world" message graphically before our very eyes, and then modify the automatically generated source code a little by hand, and then leap back into the graphic designer to change the hand-modified window a little more, and then run the resulting code. All without crashing or making a single error. Again, most impressive, if not a little foolhardy!

After dinner, we all dressed in warm clothes to join Winchester's torch-lit procession to the fireworks display. There were thousands thronging the streets, and it was a genuinely moving experience. The bonfire was the largest I have seen, being about the size of a house. The discussion on the way centred around what was actually being celebrated, Guy Fawkes' attempt to blow up parliament, or his being prevented from doing so. I believe we celebrate his failure. In typical British fashion. Not that this is unique; Liz Rather conceded that the Alamo was celebrated for similar reasons, and suggested that the confederacy was a larger example of the same.

After the display, we retired to one of the quieter pubs (quieter in the sense that the bouncer was still admitting customers, not in the sense that it was either quiet or not

euroForth: the View From Down Under

Tim Hendtlass
Hawthorne, Victoria, Australia

Winchester, the ancient capital of England. A medieval city of old picturesque buildings and narrow streets with history pressing in from every side. The Royal Hotel, warm and cozy, almost 500 years old and with no piece of level floor anywhere. The Darials Room, upstairs with a low, beamed ceiling and, from the 4th to the 6th of November, earnest Forthers from all over the globe. People from Austria, Estonia, Germany, Russia, Sweden, Switzerland, United Kingdom, United States. Oh yes, and one Australian, me. Taken together, euroForth '94.

It would be unreasonable to single any one presentation out (and space does not allow all to be discussed), but the scope of the discussion and the range of topics was most impressive. Two sessions were devoted to compilation techniques, one to formal methods, one to competition and exploration of commercial Forth. A further session to embedded applications, one to GUI and professional products, and the final session to Forth techniques. From the very practical to the theoretical and philosophical. Everyone who attended will no doubt have different special memories, but for me the overwhelming impression I got was that people are solving real problems using Forth and, while this is so, Forth is well. Seeing Forth GUIs and other *big* machine packages alongside embedded systems, hearing people sharing experiences of using different approaches to similar problems reminded me how flexible and versatile Forth is, how many different forms it can assume, and how diverse the Forth community is.

The organization and time keeping was excellent. All sessions started promptly, and an excellent way of keeping speakers to their allotted time that was new to me was used, apparently attributed to Lawrence Forsley. The speaker was given five minutes, two minutes, and one-minute-to-go warnings as is usual at conferences, but if they were then still talking the session chair shuffled across to the speaker, put their arm round them and, still shuffling, gently but firmly removed them from the stage. Very effective—I know I was very conscious of time when my turn came!

The single strongest impression is how friendly Forth users are and how they share an interest that spans language. It is people who make Forth live and people who make conferences work. To Stephen and Linda Pelc and all their helpers, a very big *thank you* for your efforts. To those who couldn't be there, bad luck, you missed a rich experience. The titles of the talks presented are listed below, but they only give a dry idea of what went on and none of the scope of the discussion. I believe the conference proceedings are to be made available, contact MicroProcessor Engineering Ltd., 133 Hill Lane, Southampton, UK SO15 5AF, so that you can at least read the full papers. But if there is a chance you can contribute or go to euroForth '95 don't hesitate—do it.

The papers in order of presentation:

Stack caching for interpreters. M. Anton Ertl

Combined addressing model for 8086 processor. M.L. Gassanenko

The effect of local variable optimization in a C-based processor environment. C. Bailey

Forth and register transfer language. G. Dunbar

Automatic scoping of local variables. M. Anton Ertl

BacFORTH: an approach to new control structures. M.L. Gassanenko

Forth and formal language theory. J. Pöial

Towards a formal specification of the Forth ANSI standard. W. Stoddart

How to turn Forth into a competitive advantage, the experiences of EDSL. B. Campbell

Embedded systems debugging: the return of software. S. Pelc

Validating Forth source text—a design concept. H. Oakford

Rhyme, reason and the Tao of Forth. L. Forsley

Going Forth with Echelon. M. Bugler

A distributed data acquisition and decision-making system. T. Hendtlass

A model of a minimal real-time executive on Forth. S. Baranoff

A GUI toolkit for polyFORTH. E. Rather

EXPRESS: A Forth-based process control software product. E. Rather

Commercial exploitation of the interactive advantage. R. Goddard

Tool interactively for rapid GUI application development. G. Ellis

An ANS heap. G. Charlton

Interactive remote target compilation and the PIC16CXX. A. Robertson

Handling source code and images in natOOF. M. Dahm

A library versus a kernel. A. Robertson

A taste of direct programming. W. Wijgaard

Breakthrough in knowledge management. Murray 2000—the desktop super computer. L. Forsley and B. Gruenwald

crowded) and sampled the ale.

Sunday brought the last session, which I kicked off with a brief advert for my implementation of the Memory Allocation Word Set. I have the ignominious honour of being the only speaker to run out of talk before running out of time.

Alan Robertson described working with the PIC16XX family of processors, which have 2K of code space, an eight-deep return stack, and a massive 35 bytes of RAM. Yes, 35 bytes! Astonishingly, he has developed techniques to work interactively on these devices by transferring virtually all of the functionality required in development to a host Forth. This was a prize-winning talk.

Markus Dahm from Aachen University brought us up to date with developments in natOOF, particularly in handing source file management over to the computer, which now logs what is loaded and defined during development, and creates a loadable source file from its logs when required.

Larry Forsley assisted Bjorn Gruenwald in describing the Murry 2000, a concept for a desktop supercomputer based on Bjorn's unique style of programming, where the fundamental object is not the word but the process, or task. Typically, Bjorn writes applications that have hundreds or thousands of tasks working cooperatively. He likens it to the techniques used by living creatures to store and process information. The Murry 2000 uses cellular automata linked in a tree structure to facilitate this. Code is generated to drive this from an ordinary procedural language by an automatic algorithm that he describes as a stochastic compiler. If the Murry 2000 lives up to its promise, it will herald a sea change in computing.

Wolf Wejgaard brought us up to date with Holon, his unique vision of the future of Forth which, as virtually all who have seen it agree, is an incredible system. He has given up with files and blocks, and works with a database, structured like a Smalltalk browser. Holon only does target compilation, but completely seamlessly. It uses recursive-descent compilation, so all compiled code is of minimum size. Both high level and code definitions are single-steppable, and now it is written in Holon. Wolf claims that the process of rewriting Holon in Holon has allowed him to improve all the rough edges, because it is an ideal development tool. I see no reason to doubt him.

The conference was rounded off over lunch by the presentation of the prizes for best talk, as indicated above, and also for the silly competition. There were three prizes awarded for this also, of which I forget one. Again, sorry. The other two went to Liz Rather, for;

```
: QUEEN  0 ' >HEAD ! ;
```

and myself, for

```
: PLAY  MILK WOOD SWAP ;
```

(Respectively, *Alice in Wonderland*, from the Queen's cry of "off with his head!" and Dylan Thomas' *Under Milk Wood*).

Then came the surprise of the year!

Stephen Pelc and Elizabeth Rather stood together to announce a new era of cooperation between Forth vendors, pointing out that the real competition was with the 99% of the world that does not use Forth. From now on, Forth Inc. will carry and use MPE's proForth for Windows in the States. (I also understand that some changes will be made to the internal structure of MPE products to satisfy the expectations of polyFORTH users—specifically the inclusion of a locate field). Steve and Liz did not actually hold hands, but it would not have been inappropriate. The applause was loud and long.

The survivors party was highlighted by a lesson from the American contingent in Shakespeare and political correctness, Bjorn Gruenwald's skilled but all-too-brief piano playing, and Larry Forsley's pizzas. Again, too short lived. We especially enjoyed the one topped with a sausage provided by Jaanus Pöial. His English did not extend to naming it, but his mime indicated a horned beast, and a question-and-answer session established it was probably venison.

It was with regret that I left before the party finished.

Nominations for FIG Directors Commence

The nominating process for the selection of new directors for the FIG Board of Directors is under way. Elected directors serve on a volunteer basis (no monetary remuneration).

The current Directors are John D. Hall, Director, President; Jack Woehr, Director, Vice-president; Dennis Ruffer, Director, Treasurer; Mike Elola, Director, Secretary; C.H. Ting, Director; Nicholas Solntseff, Director; and David Petty, Director.

At the last Board of Directors meeting, held at FORML in November of last year, Jack Woehr and Mike Elola were appointed as a Nominating Committee to solicit candidates.

In accordance with FIG bylaws, it is the duty of the Nominating Committee to nominate at least one individual for each vacancy on the Board. Currently, there are seven vacancies to be filled, each for terms of three years, with the possibility of reelection thereafter.

Nominations for vacancies may be made by petition, signed by 25 members of the Forth Interest Group. Petitions should be submitted with a brief note from the nominee stating qualifications, background, and position on issues.

The nomination and subsequent election processes take place as prescribed by our bylaws. As the following extract from Article VIII, Section 1 of the bylaws indicates, open elections are made possible by the timely completion of steps stretching over at least a five-month time period. The first step has now been taken.

From the Bylaws...

(a) Nominating Committee. The Board of Directors shall appoint a Nominating Committee composed of at least two Directors to select qualified candidates for election to vacancies on the Board of Directors at least 120 days before the election is to take place. The Nominating Committee shall make its report at least 90 days before the date of the election, and the Secretary shall provide to each voting member ... a list of candidates nominated [at least 60 days before the close of elections].

(b) Nominations by members. Any 25 members may nominate candidates for directorships at any time before the 90th day preceding such an election. On timely receipt of a petition signed by the required number of members, the Secretary shall cause the names of the candidates named on it to be placed on the ballot along with those candidates named by the Nominating Committee.

(c) If the Corporation publishes, owns, or controls a magazine, newsletter, or other publication, and publishes material in the publication soliciting votes for any nominee for director, it shall make available to other nominees, in the same issue of the publication, an equal amount of space, to be used by the nominee for a purpose reasonably related to the election.

(d) Should a petition be received, a ballot process will be provided to the voting membership. Otherwise, the Secretary shall cast a unanimous ballot for the candidates as proposed by the Nominating Committee.

Obtaining a Nomination

The Nominating Committee selects candidates for the ballot. FIG members who wish to become candidates this way should submit a letter requesting consideration by the Nominating Committee (c/o FIG office) before the deadline.

Alternately, 25 FIG members can nominate you as a candidate by petition. Send this petition to the FIG Secretary (c/o FIG office) before the following deadline.

The deadline for submitting either nominating petitions or letters requesting consideration by the Nominating Committee is February 1, 1995. Send these items to the FIG office at P.O. Box 2154, Oakland, California 94621. The Nominating Committee's selection will be reviewed by the Board and printed in the March/April 1995 issue of *Forth Dimensions* along with the candidates' statements.

After notice of the selected Board candidates, members wishing to be, but not selected by the Nominating Committee can submit a petition signed by 25 FIG members. Send this petition to the FIG Secretary (c/o FIG office) before the deadline. The names of the qualifying candidates will be placed directly on the voting ballot.

The deadline for submitting these nominating petitions is April 1, 1995. Send these items to the FIG office at P.O. Box 2154, Oakland, California 94621. The Nominating Committee's selected candidates' and petition candidates' statements will be printed in the May/June 1995 issue of *Forth Dimensions.*

Elections

Elections will not be conducted by mail ballot when there is only one nominee for each Board position.

If a mail ballot is required, the final deadline for candidates' statements will be April 1, 1995. Those statements received at the FIG office by that time will be included in the May/June 1995 issue of *Forth Dimensions.* The statement can be 500 words or less.

The voting ballots must be returned to the FIG office by July 31, 1995. The newly elected directors assume their duties at the next meeting of the Board of Directors.

Member Name (Please Print)	**Member Signature**	**Member Number**
<name1>	<name1>	<number1>
<name2>	<name2>	<number2>
<name3>	<name3>	<number3>
.	.	.
.	.	.
.	.	.
<name25>	<name25>	<number25>

Nominating Petitions should also include a candidate statement.

(Fast Forthward, from page 39.)

Design after Design

The core layer of functionality provided by my multi-purpose FIND routine could serve as a framework, or template, for other Forth refinements. Any usage context that involved a dictionary search represented an opportunity. I had a means to fix a number of quirky problems throughout Forth, and I was determined to look in all the right places.

A Forth system typically has a group of routines that help implement the runtime. The data-handling words that precede inlined data are good examples of such routines. It's an error to execute those words in interpretation contexts—and in compilation contexts, as well.

(The compiling system is removed from the runtime system where such routines serve their intended roles. In reality, compiling has much in common with interpreting. Our preoccupation upon two Forth states predisposes us to think contrary to these facts.)

The appropriate execution of such routines belongs with the function dispatch inside the inner (colon-routine) interpreter. So I embellished INTERPRET to call ?FIND along with a word-rejecting parameter that causes it to ignore inline data-handling words.

Even though the text interpreter has been changed to overlook such words, they remain executable when and where they achieve productive results.

Well, okay. There is context in which we want such handlers to be treated legitimately. POSTPONE should be able to start a dictionary search that finds them. So this is how visibility comes to these words: they must be preceded in the input stream by POSTPONE. (POSTPONE is ANSI Forth's single replacement for both COMPILE and [COMPILE]).

However, such code is rarely specified. It is only required when you are creating a compiler-extending routine that handles new kinds of inline data. In all other contexts, the visibility of the dictionary entries for these handlers is counter-productive.

Because this usage context is so rare, and because it is always moderated by POSTPONE, the search that POSTPONE launches can be made special somehow. Can you guess how?

The visibility of inline data handlers needs to be enabled or disabled depending on the search context: POSTPONE will always find them. INTERPRET will always ignore them. There is no need to conditionally find or ignore these words within either context.

Observe how Forth's factoring conspires significantly in the implementation of a safer Forth.

Compiler extensions sometimes usurp parsing and dictionary-searching responsibilities from INTERPRET, as exemplified by POSTPONE. Such compiler extensions can be considered standalone compilers. They consume one word from the input stream, the same way that INTERPRET would in one iteration of its outermost loop.

It's easy to imagine that the same embellishment for another language would have required very thoughtful conditional logic that would be nested several conditions deep. For a monolithic interpretive language, such a change would probably be beyond my reach.

Thanks to Forth's brilliant design, I was done with the change in a matter of minutes. What a relief.

As it turns out, I did not fine-tune POSTPONE because it had never been refitted with my more flexible FIND routine in the first place. It could continue to find any word in the dictionary.

The real effort involved adding a new word flag (HERALDS_INLINE_DATA), adding a new routine (ONLY_POSTPONE_FINDS) to set it, and calling that routine for each inline data handler. Besides those additions, a tiny change to the text interpreter was required.

Even when these changes are taken as a whole, this Forth tune-up is pleasing in terms of its minimalism.

Because a similar change regimen has already occurred on the morning of my imagined work day, this round of changes would naturally proceed much faster. In its aftermath, I should have plenty of time left in my day to scout for other tune-up opportunities.

For many systems, a number of inner interpreter routines can be compiled by associated word-defining words. If these inner interpreter routines are given dictionary entries, they will probably appear to be executable routines with the same status as other executable routines. However, a system crash is the likely result if these routines are executed by name reference within the text input stream.

Dictionary searches launched from the text interpreter should not find such routines, regardless of a compiling or interpreting system state. They should not even be found as part of a dictionary search launched by POSTPONE. To safeguard Forth, a new word attribute can render these words invisible in all dictionary searches. The name I chose for this new attribute was EXECUTE_IFF_AT_CF.

(POSTPONE is used to help compile bodies of routines, not code field values. Which brings up...)

We should arrange for a very special kind of dictionary search to exclusively find such inner interpreters so that they are less subject to accidental execution. The tick routine can be considered a safe moderator for such searches. Words such as CREATE might rely on the tick word-interpreter to obtain an execution token corresponding to an inner interpreter. (More likely, word-defining words will be hard-coded to "know" the correct values to compile, eliminating the need for special handling of inner interpreters.)

By way of the tick operation, the normal text interpreter can be put to good use managing substitute inner interpreters. Inner interpreters must be native-code routines. When such a routine has been defined, it can be safely used to patch the code fields of compiled words. This offers a way to change the behavior of compiled words without recompiling them. As an example, the Forth system I am developing defines a tracing version of the colon-word (inner) interpreter. I occasionally use tick to obtain its associated value, which I use to patch any colon routines that I want to trace. If the patched word is not the

source of some problem, I patch it again to reference the normal colon-word interpreter, and continue the investigation elsewhere.

To implement protection against ad-hoc execution of words such as TRACING-DOCOL, I set the flag that I dedicated for their use on each of them. Then I altered the text interpreter so that the word-rejection parameter for each ?FIND has the EXECUTE_IFF_AT_CF bit set. This prevents Forth from interpreting or compiling them when they are referenced by name in the text input stream.

An unconstrained dictionary search within the tick routine remained permissible, so it did not receive any changes. Besides the text interpreter, POSTPONE required changes to further quarantine the inner interpreter routines (which also should not appear in the "body" portion of a colon word).

Let's say that I still have time in the balance of my imagined afternoon to stretch my legs and contemplate other Forth refinements. What would I come up with next?

Without any new word attributes, a tune-up is possible that still involves the dictionary, but not a dictionary search specifically. Is it productive for you to see words such as 0BRANCH, <LIT>, and DODOES listed by the Forth WORDS routine? As has been discussed, these are not names you normally want to place in the input stream (ignoring the rare compiler extension). Other words take care of their compilation for you. Their names are of dubious value inside the input stream. So why not hide these questionable words, at least as far as the word-listing utility is concerned?

Removing such words from the dictionary is unwise. The names of these routines are needed for decompiling and tracing purposes. They can also be useful as text parameters to be parsed by the tick or POSTPONE routines.

To appear to rid the dictionary of words that are counterproductive most of the time, Forth's WORDS routine can be embellished to check each word for attributes like EXECUTABLE_IFF_AT_CF and ONLY_POSTPONE_FINDS. When it finds one of these attributes set, WORDS can skip over the associated word and move immediately to the next entry in the dictionary.

Polymorphism for Minimalists

You might have seen this one coming. We already started to look beyond dictionary-searching contexts as candidates for change. That way, we can extend our growing safety net to encompass other areas of Forth.

Although the EXECUTE routine fails to consult the dictionary normally, that situation could change. We would duplicate the execution token on the stack and use the copy to query the attributes of the associated word. We could arrange for EXECUTE to check some of the already suggested word flags to determine if the current context is an appropriate one for the token atop the stack. If it is not an appropriate context, EXECUTE could report an error rather than run a routine that has the potential to crash the system.

Word attributes that EXECUTE might inspect include FIND_IFF_COMPILING and EXECUTE_IFF_AT_CF. Testing of the former attribute will require checking the accompanying system state too.

The extra overhead primarily affects the speed of the text interpreter which relies on EXECUTE. Almost all compiled code would bypass these safety checks. These checks would be performed for each word in the input stream "seen" by INTERPRET.

Such alterations to EXECUTE can supplant the equivalent upstream actions within the text interpreter. However, this is not a wise thing to do. When usage errors are detected upstream within the text interpreter, the ?FIND routine can continue to search for another word of the same name deeper in the dictionary that meets all the prevailing search criteria.

Placing such error checking in EXECUTE while trying to arrive at a similar solution would be much more awkward: EXECUTE would have to raise an exception to be handled by the text interpreter. The text interpreter would have to take control by resuming the original search at the word beyond the one that led to the exception, then pass control back to EXECUTE if a new word is found that at least has the correct name.

Given the complexity of this alternative, it is simpler to perform the error-checking twice, once at dictionary-search time, and again within EXECUTE itself. By creating an INTERPRET-only version of EXECUTE that skips such error-checking, the text interpreter's duplicate checks would be eliminated.

Notably, either implementation permits the (polymorphic) overloading of names such as dot-quote, so that the novice (and expert) can enjoy a more consistent and easier-to-learn user interface.

Such name overloading permits Forth to approximate the polymorphism exhibited by object-oriented programming languages.

As I have taken care to report, simple implementations are available for all of these Forth tune-ups. Therefore, Forth's minimal look-and-feel is not in the least endangered.

Making Forth Easier to Grasp

As I have tried to show, a number of Forth tune-ups can be implemented within an eight-hour stretch of time. But why should you spend eight hours this way?

In such a short time and with so little effort, you can fine-tune Forth's operation. That way you will provide better guidance to the Forth novice. As exemplified by reduced error susceptibility, the system you create will not tie the hands of the Forth expert—who can still access any Forth functionality that isn't counter-productive to begin with.

Who loses, if your changes preserve Forth's usual flexibility and minimalism? It's unreasonable for you to leave the next generation of Forth programmers prey to so many easily avoided errors. Your protestations that Forth's error susceptibility upholds a long-standing tradition will only make you look mean-spirited, not smug or clever.

(I am not claiming to have found ways to tune out all

the possible error pitfalls. However, the measures I have described make significant progress towards that goal.)

Through a system designed to help the novice, you will also help the expert.

When applied to your varied Forth kernel routines, a handful of descriptive word flags will help to correctly categorize many of them. This aids user understanding and provides your kernel with a form of self-documentation. Even if they could not produce a safer Forth, there might be an organizational need for such categorizations.

Cleaning House

The adoption of new word attributes helps clarify the role played by the varied routines that constitute a Forth kernel. Perhaps, with these aids to understanding, more Forth users will gain the knowledge to become system implementors, too.

Taken together, these tune-ups are a form of house cleaning for Forth. In a way, I am saying that Forth implementations have typically been eyesores, with bits and pieces of the kernel strewn all over the place. By tidying Forth, many of the difficult aspects of Forth's operation can be made clearer—and aspects that are susceptible to errors are effectively quarantined. Forth doesn't have to appear haphazard and be hazardous to prove its power and minimalism.

Furthermore, these tune-ups lay the foundation for a slimmer Forth namespace. One way to unclutter the namespace is to withdraw support for certain words that were introduced long ago to make Forth more error resistant and more "correct."

For one, the (fig-Forth-introduced) routine `?COMP` can be retired and all the words where it appeared can be streamlined.

Dot-paren may be supplanted as well. At least its namesake can be retired, because two state-oriented dot-quote routines can be discriminated by the enhanced text interpreter. Likewise, other names that we ought to be able to retire are `[CHAR]` and `[']`.

Summary

Treating word flags beyond "immediacy" as first-class citizens of a Forth system has proven worthwhile. I have shown how to use them to improve Forth's namespace management and to add substantial error detection capabilities to Forth.

The result is a Forth with a sensible user interface, as well as with sensible groupings of Forth kernel routines. The Forth command interface can become more consistent and Forth's operation can become more friendly.

Best of all, these enhancements can be implemented very simply. Because no Forth virtues were compromised, no hardships should befall anybody.

As the term "tune-up" implies, these benefits were always within Forth's close reach. I am proud to be the one to reveal them. My appreciation of Forth continues to deepen as I see how well it endures as a fountain of creativity.

Studying Forth is a tonic for your thought processes. If you let it, it will lead you to highly refined designs and implementations.

ADVERTISERS INDEX

Fast FORTHward

Fine-Tuning Forth

Mike Elola
San Jose, California

Coding and designing are good tasks to alternate between as part of an unhurried development process. On a really good day, you might return to design work once or twice before leaving the day's coding efforts behind.

Co-mingling the design and coding tasks is more likely to occur when the final design evolves gracefully out of earlier designs.

A rapidly evolving design seems to take on a life of its own, guiding us as we go. This can lead to the sublime experience of programming as an interactive experience, despite its solo performance. I'll try to convey a sense of this as I describe a dream of a work day. Of course, the work day I am about to describe is purely fictional.

In an earlier installment, I lingered on some of the problems with the vocabulary mechanism (see *Forth Dimensions* XVI/1). I was concerned about the lost productivity that could ensue.

While I haven't found any answers for Forth's vocabulary quirks, I have discovered various tune-ups related to Forth's namespace management. Error avoidance is the primary benefit. The tuning also helps organize a raft of Forth kernel functions. By clarifying and helping to crashproof Forth, a more enjoyable programming environment emerged.

Forth doesn't have to appear haphazard and be hazardous to prove its power and minimalism.

Error Avoidance Without Sacrificing Minimalism

The FIND routine walks the dictionary's linked list as part of the search process. In the last installment, I described how I created a variation of the FIND routine that is able to search the dictionary in accordance with various flags.

This version of FIND was called ?FIND. It accepts two new input parameters, one of which is a word-sanctioning parameter. The other is a word-rejecting parameter.

A dictionary search is normally performed once for each input word. We can build upon Forth's states by establishing different search behaviors for different search contexts, or system states. Through state-attuned search behaviors, we can cure many of the ill-mannered aspects of Forth's operation.

The system state can actually be checked twice by the text interpreter, once to obtain a suitably tuned dictionary search—and once more to trigger compiling or interpretation of a word. While compiling and interpreting actions still follow a successful search, the new logic will enable Forth to be more selective about the words available to be compiled or interpreted.

As shown following, the parameters for ?FIND help fine-tune INTERPRET to account for different word states within different system states:

```
: INTERPRET
  BEGIN
    ...
    STATE @ IF ( compiling-state search )
      <sanctioning-attribute>
      [ OR <sanctioning-attribute> ]...
      <rejecting-attribute>
      [ OR <rejecting attribute> ]...
      ?FIND
    ELSE  ( interpreting-state search )
      <sanctioning-attribute>
      [ OR <sanctioning-attribute> ]...
      <rejecting-attribute>
      [ OR <rejecting-attribute> ]...
      ?FIND
    THEN
    ...
```

Compiler-extending routines have compile-time behaviors that must be acted out at the appropriate place and time. The immediacy bit helps ensure that those compiling behaviors are exhibited at compilation time. However, no corresponding effort is taken to suppress those behaviors at other times. (Individual routines supply their own error checks in well-mannered Forth systems.)

The elimination of this error susceptibility was an

implied pursuit of the last two installments of Fast Forthward (through a new module for compiler words in the earlier installment, and through a new version of FIND in the most recent installment).

A very direct approach would be to just pass a parameter to the embellished INTERPRET routine telling it to neglect immediate words while interpreting. However, so direct an approach turns out to be an oversimplification.

A few immediate words can be used safely in interpretation mode, including "paren." Few other immediate words are likewise state-neutral. Still, they require a new word attribute independent of the immediate flag. The name I chose for this flag is FIND_IFF_COMPILING.

The new flag will identify words that have unpredictable and nonstandard consequences (possibly fatal) if executed in interpretation mode. Such a flag can be set independently of the immediacy flag. That way, non-immediate words such as >R and R> can also receive the new attribute without changing their treatment during compilation.

Such a provision may be difficult to justify when every Forth programmer is well trained. But if the overhead is extremely low, what does it hurt to make it a part of the system? As few as 250 bits are needed for a Forth system of that many words. Assuming we set aside two bytes per word to leave room for 13 other new word attributes, this feature adds only .5K of memory to such a system.

Because Forth's compiling system is tiny to begin with, no one should care. A turnkey-application compiler eliminates the compiling system from the final application, rendering this a non-issue anyway. (Forth applications usually do not need to be delivered with the compiling system left intact.)

To evaluate overhead costs more equitably, consider how many compiler words incorporate phrases such as ?COMP, which is defined something like:

```
STATE @ ABORT" Compilation Only" ;
```

Words containing such code can be streamlined by setting the new attribute instead. So those who agitate for minimal Forth systems should feel like they are also being heard.

Whereas some "system" source code can be supplanted, some new system code has to be added to be able to specify the new attribute. Because the vast majority of words that are immediate are intended for use only in compilation mode, the word IMMEDIATE can be changed to set both the immediate flag and the FIND_IFF_COMPILING flag. INTERPRET_FINDS can be the routine for clearing the new attribute for those rare immediate words that can be executed safely during interpretation. INTERPRET_MISSES can be the routine for setting the new attribute for non-immediate words that are also unsafe to interpret, such as >R.

To accomplish these changes, let's say that I will set aside all morning. That should be a liberal amount of time to change and then regenerate the kernel.

(Continues on page 35.)

Product Watch

OCTOBER 1994

Creative Solutions announced a Power Mac version of MacForth that takes as its name *Power MacForth*. It is a native-code Macintosh development environment that allows the creation of high-speed applications on a RISC platform. Additionally, access to the Mac Toolbox has been made easier. A special-offer price of $129 is extended to current MacForth owners.

Kudos to CSI for the distinction of creating a line of ANS Forth products

Power MacForth is an ANSI standard Forth dialect. Creative Solutions also expects to be shipping an ANSI upgrade for its 680X0-based MacForth around press time.

NOVEMBER 1994

From MicroProcessor Engineering, Ltd. comes *ProForth for Windows*, which is being sold in the USA and Canada by Forth, Inc. "Exceptional interactivity and ease of development" are offered besides technical support according to Elizabeth Rather, president of Forth, Inc. Rather and Stephen Pelc, the managing director of MPE, made the announcement of the agreement between the two companies at the annual EuroForth conference in England.

The 32-bit Forth system for Windows and Windows-NT includes ProForth GUIDE™, a code-generating tool that allows Windows-compliant user interfaces to be designed and modified in a purely graphical manner. Integration with the host operating environment is complete, including support for DDE and linking to third-party DLLs.

Forth, Inc. sells its own line of Forth products, including the EXPRESS™ software package. It sports a graphical user interface, too, but is outfitted for embedded systems development through its real-time capability and multitasking support.

COMPANIES MENTIONED

Creative Solutions, Inc.
4701 Randolph Road, Suite 12
Rockville, Maryland 20852
Phone: 301-984-0262 • Fax: 301-770-1675

Forth, Inc.
111 N. Sepulveda Blvd.
Manhattan Beach, California 90266
Phone: 800-55-FORTH • Fax: 310-318-7130

MicroProcessor Engineering Ltd.
133 Hill Lane
Southampton SO15 5AF
United Kingdom
Phone: 0703 631441 • Fax: 0303 339691
Telex: 474695 FORMAN G

Forth Modification Laboratory
FORML and EuroForth '92 and '93
Conference Proceedings

Conference papers from the fifteenth annual FORML Conference held November 1993 at the Asilomar Conference Center, Pacific Grove, California, U.S.A. and conference papers fro the EuroForth '92 Conference held October 1992 at Potters Heron Hotel, Ampfield. Nr Romsey, Hampshire, U.K. and the EuroForth '93 Conference in Marianske Lazne (Marienbad), Chexh Republic 509 pages.

Order from the Forth Interest Group.
FIG member discounts available.
See order form inside.

$45.00

1993 FORML Conference

- A Novel Approach to Forth Development Environments for Embedded Real-Time Control
- ExecDoc Scripting Tool
- F21 and F*F
- Forth and the rest of the (DOS) World
- Forth in 32-Bit Protected Mode
- Forths in the Design, Text and Extension of an HDTV Format Converter
- Graphing Functions in Forth
- Implementing Forth as a C/C++ Library
- MIPS eForth
- Multiple Entry Points
- Object Oriented Programming in One Definition
- Optimization and Macros in This Forth
- Plumbing Forth - Filters and Pipes
- Postscript File Filter and Hypertext Indexed ANS Forth Standard
- Simple Mouse and Button Words for DOS-Based Systems
- To Boldly Go Forth Where No One Has Gone Before: Talking Forth to Forth
- The Lion's Choice of Names
- The Simplest File Loader
- Umbilical Compilation
- Why HLLD Forth?
- Wild Bean and Blind Awe

euroForth '93 Conference

- Error-0 undefined" versus "C1000 Unknown Fatal Error..."
- A Forthable ATE - Implementing Inexpensive Automatic Test System using Forth
- The Common Sense Pattern Classifier - Application Studies
- A graphical user interface in natOOF based on the Model-View-Controller concept
- A Portable Forth Engine
- Context-Oriented Programming: Evolution of Vocabularies
- Multi-CFA>: Implementation via Self-Modifying Code
- Static Stack Effect Analysis
- A Look at Forth's Academic Standing
- Towards a Formal Forth
- Porting Forth through C on Workstations
- Increasing RTX-2001 Return Stack Depth
- Quick, Make a Weighing Machine!
- Some ideas on Formal Specifications of Forth Programs
- Interface X: The First Forth Single Chip Controller
- The Halting Problem in Forth
- Self Reference and Type Interference in Forth
- Interactive Cross Platform Development

euroForth '92 Conference

Session One - Communication - Good, Plus Good, double Plus Good
- WRITING Better Forth
- Unity and Communication for Exploration
- Holon - A New Way of Forth

Session Two - Exploration - Object Lessons in Hard Truths
- Toward Programming in Natural Language Style in the Object-Oriented Forth natOOF
- The Specification Language ETF-1
- Rule-Based Expert Shell for Real-Time Control
- Marketing Forth and Forth Products in Bulgaria: problems and perspective
- A DMA-based PC-AT Communication Architecture for DSP Applications using FIFO RAMs
- A 448 Byte Forth Multitasking Kernel
- FRP 1600 - 16-Bit real time processor

Session Three - Unity - Methods and Skills in Harmony
- The Hysteretic Heap
- A New Approach to Forth Native Code Generation
- Cellular Automata on a Personal Computer
- Multi-Target Program Product Stripping in Beta-Forth
- Implementation of the Forth Type Checker
- The (almost) Complete Theory of Forth Type Inference
- PROCIC - Process Computer for Computationally Intensive Control

Session Four - Communication - Application: The Employment Connection
- FOSM, A FOrth String Matcher, continued
- Application of the RTX processor as a high performance multi-channel data converter
- SMAN - From Greatness to Excellence
- Forth at the International Convention Centre, Birmingham, England
- The DBMS - Forth realization problems and solutions
- A Portable Implementation of Logo in Forth
- Using Forth to improve programming productivity on Programmable Controllers