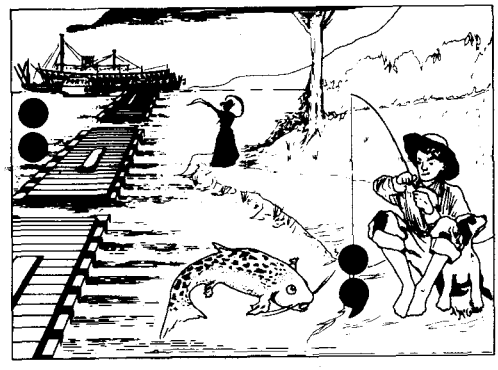
Volume 5, Number 2
July/August 1983
$2.50

FORTH Dimensions

Moore on FORTH



FEATURES

DEPARTMENTS

FORTH Dimensions

Published by FORTH Interest Group

Volume V, No. 2

July/August 1983

Editorial
Marlin Ouverson

Publisher
Roy C. Martens

Typesetting/Production
LARC Computing, Inc.

Subscription to FORTH Dimensions is free with membership in the FORTH Interest Group at $15.00 per year ($27.00 foreign air). For membership, change of address and/or to submit material, the address is: FORTH Interest Group, P.O. Box 1105, San Carlos, CA 94070.

Letters to the Editor

Imagine, If You Will...

Dear Editor:

Am I in the FORTH Dimension or am I in the "Twilight Zone"? I have implemented FIG-FORTH on my TI-99/4A. It is not completely debugged, but it runs. (Thanks, Mike O'Malley.) The disk is not linked in yet, though I hope to complete that portion soon. I would enjoy hearing from anyone who has implemented or is implementing FIG-FORTH on the TI-99/4A.

Maybe TI will be marketing a FORTH product superior to what I have put together but at least I can say I'm the first kid(?) on my block with FIG-FORTH on his TI-99/4A. It's a fun project. Don't keep me in the "Twilight Zone," guys. If you're out there coding FIG-FORTH on your TI, drop me a line. Maybe we can trade ideas.

John Forsberg
17740 S.W. 109th Place
Perrine, FL 33157

(Continued)

Cover Art

Artist Al McCahon pictures catfish taking the bait of a new definition, as a barge of new R65F11s heads downstream. That FORTH showboat in the distance has set sail for the annual FORTH convention... it's a long way from the Mississippi to San Jose, but well worth the trip for this year's event.

Editorial

Changing of the Guard

A little more than two years ago I was planning the first all-FORTH issue of *Dr. Dobb's Journal*. Making contact with the FORTH community and acquiring good FORTH code and articles took the better part of two months. It was a pleasure to meet the new people, and to learn new ways of finding software solutions. And it was a challenge to make sense of what often seemed like—and sometimes was—hieroglyphic code. It was such a satisfying and rewarding project that we made the *DDJ* FORTH issue an annual event.

As a result, when the FORTH Interest Group asked me to become the editor of *FORTH Dimensions*, it was a pleasure to accept. I look forward to renewing old friendships and to making many new ones in the course of publishing some of the most exciting and important material in the microcomputer industry. I would like to begin my tenure by expressing a collective vote of thanks to former editor Leo Brodie. He is continuing his work in the FORTH community and will undoubtedly be seen at various FORTH meetings (speaking of which, have you made plans to attend this fall's conference in San Jose?).

FORTH Dimensions has some exciting issues planned, and we are looking for authors and programmers to contribute code and articles. We need utilities, applications, some good tutorials, and articles about data acquisition, project management and many other topics. If you have an idea or a request, now is the time to write. You will receive a reasonably prompt reply. And remember, "Letters to the Editor" is your forum. Use it if you know a way to do something better or faster, or to express your gripes and praise.

Your contributions are always welcome, but especially now. We are on an accelerated publishing schedule in order to once again come out on time. In a few months we will be back on target. We are working towards installation of a new procedure that will speed up the entire publishing process. Our typesetting is done via interface to an HP-3000, and many articles are now edited on my own micro, then relayed by long-distance telephone to the mainframe. Once that part of the process functions smoothly, we plan to allow authors to upload their own material. It will then be reviewed online, edited and typeset without requiring anyone to re-keyboard the text. This will help maintain accuracy and hold down costs.

The obvious extension of all this wish-making will eventually be subscriptions to the electronic edition of *FORTH Dimensions*. Readers will someday have the option to download text and code directly to disk. *But please don't write to ask about these services yet!* I just wanted to give a sneak preview of a not-improbable future. In the meantime, be ready for anything—especially some great issues to come. And let us hear from you!

—Marlin Ouverson
Editor

DBMS, or,
Do Blanks Mean Something?

Dear FIG,

While participating in coding a data base application in FIG-FORTH recently, I discovered that the FIG word **NUMBER** apparently requires that the ASCII text string to be converted be followed by a 20H to work according to its definition. Even though this shouldn't pose a problem if the conversion is done at **HERE**, I believe the definition of **NUMBER** should state the requirement.

Sincerely,

James R. Schierenberg
119 S. Berry Avenue
Indianapolis, IN 46219

Addison-Wesley's Pocket Guide to FORTH *explains that the string to be converted must be followed by a blank (20H).—Ed.*

RAM Card Access

Dear FIG,

First, please send me a writer's kit, as I have a definite interest in submitting material to *FORTH Dimensions.*

Next, I am using MVP-FORTH on an Apple II and am interested in hearing from anyone who has the same configuration. In particular, I am interested in modifying the source to take advantage of the RAM card. If anyone has done this, I would like to know what their success was.

Thank you for publishing such fine work, and keep up the good job.

Sincerely,

G. Edw. Learned
1513 Woodbine Lane
Brooklyn Center, MN 55430

Wanted: Lexi-Con Artist

Dear Editor,

I am what one might describe as an intermediate-beginner where FORTH is concerned; for the past year or so I have been working with a FIG-FORTH on the Apple II and Apple IIe computers. I have been receiving *FORTH Dimensions* most of this time and generally enjoy the publication.

Occasionally, though, *FORTH Dimensions* has been the unknowing instigator of many frustrating moments. Why? The author of a particular article has used FORTH words in a definition which may be standard to his FORTH but not to mine.

Most recently, I have been struggling to bring up Michael Jesch's "Floating Point FORTH" (Vol. IV No. 1). Mr. Jesch uses the following non-FIG words:

```
< >
< ROT
I'
(D.)
M*/
```

How would these words be defined in "FIG-ese"?

For those of us who are still in the learning curve, it would be extremely helpful if *FORTH Dimensions* would adopt a standard and ask authors to stick to it. My preference, of course, would be FIG.

Thank you for your consideration of, and response to, this matter.

Sincerely,

William B. Judd
Sales Support Analyst
Tech. Representatives, Inc.
3100 N 14th St., Ste. 101
Lincoln, NE 68521

Dear FIG Folk,

My membership dues and subscription renewal to *FORTH Dimensions* Volume V are enclosed.

Please keep printing the check sum (as defined in Vol. IV No. 3). It really helps.

We need more tools (but I've seen enough editors, really). The data base design articles by Haydon and Watkins were very helpful. I've had trouble getting the **PICTURE** stuff by Fittery (Vol. IV No. 5) working, but it had good ideas. I could use more help and suggestions for output formatting (to screen and printer), input and checking of terminal entry, and defining and using data entry and update screens.

Friends, I need an article on FORTH debugging tools real bad. Debugging should be easy in an interpreter like FORTH, but I haven't been able to redefine **DOCOL** and **;S**.

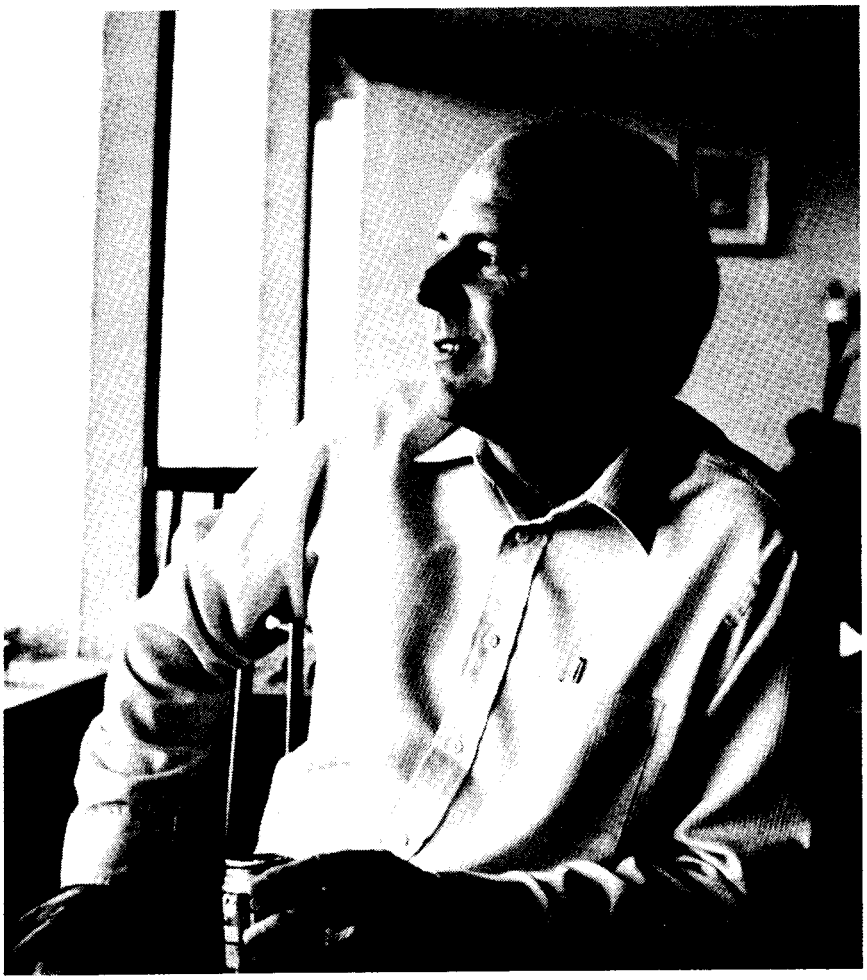
Please check the code you print for

(Continued on page 18)

Interview with Charles Moore

Marlin Ouverson
Forth Dimensions

The author found the Moore house by wandering through sunny beachfront streets in southern California until he met a woman with the kind, smiling face matching the friendly voice of "Min" Moore. Taken under wing, introductions were made and everyone settled down to a grand view of the Pacific Ocean and Catalina Island. The tapes of the interview are interspersed with thunder, laughter, and observations and speculation by the inventor of FORTH.



Forth Dimensions: Is it true that you developed FORTH originally as a productivity aid in your own programming?

Charles Moore: Productivity aid is a hard phrase—I don't think I was thinking of it as being able to write more programs. I was probably thinking of doing what I already did, but more easily. The hassles of compiling a Fortran program were what I was trying to avoid: loading card decks and loading more card decks, dropping them and picking them up, and then printing out listings in order to see what was going on.

I guess it's pretty obvious that having done that, having by-passed the operating system, it was much easier to do things and I could do much more; so I thought my productivity was increased. But at the time I kind of accepted the notion of "write one program a year."

I've heard a couple different stories about your first practical application of FORTH...

The first one was in carpet design, the pattern in carpets, high-resolution graphics. This was not raster, this was a vector graphics system. And even with the kind of FORTH I had then, I could do complex pictures that moved—not that that was relevant to carpets. I could do in 4K and a tenth of a second what otherwise would have taken several seconds. And you couldn't effect motion that way, and it would have been too large to fit in available memory.

So you went from pedestrian to extra-terrestrial by working next in astronomy and instrument control?

Well, after the graphics I got involved with a large data-base management system. There I was using FORTH as the operating system and interfacing with Cobol modules. All of the real transaction processing was to be done in Cobol. Fortunately, the operating system aspect of FORTH was emphasized there, and it did a very good job. Performance was dramatically greater than we could have achieved any other way. At that point I began to suspect that I could have coded the Cobol modules in FORTH more easily than in Cobol. I didn't dare push it, because that was asking people to believe too much. "Obviously that wasn't possible—if so, what was the point of Cobol?" The answer is (*laughing*), I was right and there wasn't any point to Cobol.

My next application was astronomical data acquisition, followed by telescope control. I worked in that field for many years; that was fun. I'd still like to work in that field. If I ever get the right equipment, I'd like to put together my own telescope. Small, maybe eight- or ten-inch, with the right kind of controls and some sort of vidicon so I can put the telescope on

the roof and sit in the comfort of my control room and observe just like the astronomers do.

Some say that FORTH either breeds or attracts fanatic interests...

We were talking about that yesterday, not among the FORTH people but among business people. Speculating on how the successful entrepreneur is usually weird. He doesn't fit the accepted patterns of career or lifestyle. The same kind of thing is true of FORTH programmers—probably of computer people in general. People who like to work with computers, either in the hardware or software sense, are not your average person. They may be introverted—a flock of psychiatric terms probably apply—but in particular they are the kind who would rather deal with things than with people.

That's quite a distinctive characteristic. People who like to deal with FORTH are like the hardware people who want to build their own machines. Not that they are necessarily unsatisfied with available machines; mostly they just want to understand how it works, everything about it, to be in control of the hardware. The only way you can really do that is to build it, put it together piece by piece.

FORTH addresses that person's preferences because those are my preferences. That's what I wanted in a language—something I could change any part of I wanted to. When that kind of person finds FORTH, he tends to like it because it does give him the power he wants. No other language does that; going to Assembler, you can do anything you want, but the difficulty associated with it is very high. And the ability to write good software is not a forgone conclusion.

Now I consider that FORTH has solved the software problem. You can just do it in FORTH and that's the end of it. But now the hardware problem. You cannot build or control your own hardware to the degree that is attractive. So I'm building a FORTH computer and I want to give myself the ability to go down there and twiddle the bits and define the registers and move the data around without someone else having gotten in the way and prevented me from doing what I want.

I think there is a lot to be done over the next few years: the building block approach to hardware, where you buy some simple pieces and put them together in some simple ways and have much greater capability than you can buy because you've customized the hardware to your own problem.

Like many FORTH programmers, you like to get in and enhance or change FORTH itself. How about the tension that exists—I suspect it is a healthy tension—between the standards team and the vendors, who want some kind of stability in the definition of the language?

That's a very hard issue. We got together with the standards team to formulate a transportability standard, so that one person's programs could run on another person's machine. I think we achieved that pretty well with FORTH-79. I think FORTH-83 addresses it as well. The question arises, how much does anyone actually run a program on someone else's machine? FORTH doesn't seem to be a language like BASIC, where you write lots of programs and distribute them widely and lots of people use the same programs. FORTH is much more the language in which you write programs than in which you run programs. I've kind of always argued that the degree to which we can afford to discuss transportability depends upon the amount of transportation that happens. But it's the chicken and the egg, because if you don't have a common language, you aren't going to get any programs transferred at all. So.

The standards team, though, is recently (and inevitably) talking, not about the words everyone is using, but the words everyone should use. They are becoming a promotional discussion instead of a concensus discussion. That is much more interesting, but it isn't quite the purpose that was originally intended: what words can we agree upon which, if used, you don't have to worry about portability. Unfortunately, a lot of the portability of a real program that you're going to run on your IBM PC isn't going to be interesting unless it does something with graphics, or interfaces with the keyboard in an interesting way... The user interface becomes very important and the standard doesn't address that interface at all, because it can't. There is too much variety in that area.

I was reading *Twice Shy* this morning. It deals with computers. They have two computers, a Grantley and a Harris. I don't know if they are real or not, I suspect not. A program is written for one and wants to be run on the other. They are both in BASIC—what's involved in doing it? I don't think Dick Francis knows all that much about computers, but he discusses the problem of confusion, confusion, confusion, and he makes a point that manufacturers probably do this deliberately. They want to lock in their customers. They will make their system unique in some, perhaps, trivial way just to prevent their customers from changing to someone else's machine. And of course that prevents anyone else's customers from changing to their machine, so they lock themselves out of a market, too. It's not clear that they gain anything. It's not even clear that they do it deliberately.

Of the things you see people doing with FORTH, what do you find most encouraging and least encouraging?

The most encouraging development is the FORTH chips that are coming out. The reasons they are coming out seem to be the correct reasons, as well. Because FORTH is compact, it is feasible to implement the FORTH primitives in ROM on a chip; that's one of the advantages of being compact. The fact that you have infinite amounts of memory, theoretically, does not mean you have all the memory in the world in every situation. Ada is most grossly in violation of the memory ethic, but that is a language designed for the modern world, which explicitly assumes that arbitrarily large address spaces are available. And that is not the case when you are putting a computer in a bullet—you've got a tiny amount of memory because you don't have very much size to work with. And Ada is never going to program self-seeking, infra-red-guided bullets. It just can't possibly do that. So that is promising: the recognition that it is not the case that memory is

5th Annual

FORTH NATIONAL CONVENTION

FORTH-Based Systems: A Look into the Future

October 14–15, 1983

Hyatt Palo Alto

4290 El Camino Real, Palo Alto, CA 94306 USA

- **Exhibits**
- **Speakers**
- **Tutorials**
- **Vendor Meetings**
- **Panel Discussions**
- **Equipment Demonstrations**
- **Discussion Groups**
- **Worldwide FIG Meeting**
- **Banquet**
- **Awards**

FORTH is for everyone. The FORTH computer language is used in video games, operating systems, real-time control, word processing, spread sheet programs, business packages, DBMS, robotics, engineering & scientific calculations and more.

Learn about FORTH and make your life easier. The convention will show you how!

FORTH-Based Systems: A Look into the Future is the theme and will cover FORTH applications, FORTH-based instruments, FORTH-based operating systems, and more. Those wishing to participate and be speakers and/or panelists are urged to contact the Program Coordinator immediately. (Telephone the FIG hotline 415/962-8653)

PROGRAM

FRIDAY, OCTOBER 14
EXHIBITS Noon - 6 pm

11:30 am	Registration
1 pm	IBM/PC FORTH Systems
2 pm	Data Base Programs
3 pm	FORTH-Based Instruments - I
4 pm	FORTH-Based Instruments - II
5 pm	CAD/CAI/CAE/CAM (Computer Aided Systems)
6 pm	Exhibits Close

SATURDAY, OCTOBER 15
EXHIBITS 9 am - 5 pm

10 am	Aids for Handicaps
11 am	Automatic Programming Systems
Noon	Lunch
1 pm	FORTH Chips & Computers - I
2 pm	FORTH Chips & Computers - II
3 pm	Intelligent Peripherals
4 pm	FORTH-83 Standard, FORML Preview
5 pm	Exhibits Close

BANQUET

7 pm Saturday — Reservation and payment required — $25.00

ROBOTS, ROBOTS, ROBOTS..... Thomas Frisina, President, Androbot, Inc.

Convention registration is $5.00. Special convention room rates are available at the Hyatt Palo Alto. Contact FIG or the hotel and mention the FORTH convention. Telephone direct to Hyatt reservations by calling (800) 228-9000 and request the special FORTH Interest Group Convention rates for October 14th and 15th.

The FORTH Convention is sponsored by the FORTH Interest Group (FIG). The FORTH Interest Group is a non-profit organization of over 3,800 members and 40 chapters worldwide, devoted to the dissemination of FORTH-related information. FIG membership of $15.00/year ($27.00 overseas) includes a one-year subscription to **FORTH Dimensions**, the bimonthly publication of the group.

☐ Yes! I will attend the FORTH Convention.

☐ Number of pre-registered admissions _____ × $5.00 each $ _____

☐ Number of Banquet Tickets _____ × $25.00 each _____

☐ Yes! I want to join FIG and receive **FORTH Dimensions**. ($15.00 US, $27.00 foreign) _____

TOTAL CHECK TO FIG $ _____

☐ I want to exhibit, please send exhibitor information.

Name _____

Address _____

Company _____

City _____ State _____ Zip _____

Phone ( ) _____

Return to: **FORTH Interest Group** P.O. Box 1105, San Carlos, CA 94070 • **415/962-8653**

5th FORML Conference

November 23-25, 1983
Asilomar Conference Center
Pacific Grove, California, U.S.A.

FORML is a technically advanced conference of FORTH practitioners. The topics to be discussed will affect the future evolution of FORTH. FORTH programmers, managers, vendors, and users will benefit from several informative conference sessions. All attendees are asked to participate and are encouraged to write a paper for presentation in an oral or poster session.

Topics Suggested for Presentation

Hardware FORTH Implementation	**Nucleus Variations**
Large Address Space Environments	**Operating System Environments**
Multiprogramming Architectures	**System Generation Techniques**

Registration and Papers

Complete the registration form, selecting accomodations desired and send with your payment to FORML. Include a 100 word abstract of your proposed paper. Upon acceptance by FORML, a complete author's packet will be sent. Completed papers are due September 30, 1983.

Registration Form
Complete and return with check made out to:
FORML P.O. Box 51351, Palo Alto, Calif. 94303

Name _____

Company _____

Address _____

City _____ State _____ ZIP _____

Phone (day) _____ (evening) _____

I have been programming in FORTH for: (years) _____ (months) _____

Accommodations Desired:
Prices include coffee breaks, wine and cheese parties, use of Asilomar facilities, rooms Wednesday and Thursday nights, meals from lunch Wednesday through lunch Friday. Conference attendees receive notebooks of papers presented.

Conference attendees, share a double room:
number of people _____ × $200 = $_____
Attendees in single room (limited availability)
number of people _____ × $250 = $_____
Non-conference guests:
number of people _____ × $165 = $_____

TOTAL ENCLOSED $_____

Options: Vegetarian meals? ☐
Non-smoking roommate? ☐

FORML, P.O. Box 51351, Palo Alto, California 94303, U.S.A.

cheap. Memory is sometimes cheap, sometimes very expensive.

The least encouraging aspect is the fact that there is no large-scale education effort to train FORTH programmers. FORTH programmers are in short supply. And quality is undefined. Pretty much, if you want to hire a FORTH programmer you are going to have to train him, and the quality of training is up to you. If you find someone who says he knows FORTH, it is very hard to judge how much he has learned. If there were some courses which large numbers of people went through, you could calibrate the people by the course syllabus. There is just not a significant effort in that direction.

It's been harder in the past than now. It did not make sense for universities or high schools to teach people FORTH because there was no market for training. Now there is a market, albeit a small one, and probably every university should have a course in FORTH, or a portion of their basic computer training should address FORTH. I would like to establish a—university is perhaps too large a word—a place where people come and spend a week or two, a month or whatever, learning about computers and FORTH. If you are taking a course in computers which is taught in Pascal, you mostly don't learn about computers, you learn about Pascal. You learn about how a certain class of people think programs should be written and algorithms defined. But you don't learn much about the computer underneath the software.

Take the same course in FORTH. FORTH is more transparent, and you would focus more closely on the problem, on the ways of solving it, than on the theory of computation, which was supposed to make things easier. It isn't clear that kind of conceptual framework simplifies the problem; it changes the problem from one of worrying about bits to worrying about floating-point numbers. But it doesn't make it easier, it doesn't make it go away. It just changes it.

FORTH doesn't change it so much. If you want to learn about real-world signals and how you interface with them with latches and A/D convert-

ers, FORTH lets you get directly to that part of the problem instead of getting bound up in compilers and subroutines and things which aren't of fundamental interest.

What does FORTH forbode for languages to come?

It's a fascinating speculation. Given that FORTH is here but does not solve all of the problems, what should follow?

The next language has got to be spoken, because voice recognition is coming along very quickly. I don't know that people oughtn't to type at keyboards; keyboards are a very flexible interface device. But they don't want to, they aren't going to. They want to talk to their computers. I think FORTH could be a good spoken language if we eliminate some ambiguities that are in it.

The language needs to have a context. This is something missing in the discussions that take place about robots. I perceive a robot differently than people do at the moment. Of course, I maintain that I'm correct and they are wrong. A robot has to be aware of the environment. It has to know things that a machine doesn't. Everyone is kind of aware of this, but they don't see the profound requirements. The things a robot is interested in are not the things a human being is interested in. It really doesn't care what the temperature is, for instance; human beings are preoccupied with temperature—it had better be between 68 and 72 or there is going to be a comment made. But a robot ought to be mortally preoccupied with the exact state of its charge and the location of the nearest re-charge outlet.

In order to make a robot behave what we would call intelligently, we have to make sure it understands a lot of things. For instance, a lot of concepts simple to human beings are not easy to program. The concept of "on" or "under" or "within." The fact that small things can be put inside larger things, if the shape fits; that light things can be put on top of tables but heavy things really can't. We need to describe the world somehow in a way that is relevant to the computer. Whereupon we don't have to explain it anymore. It understands the whole class of things that, in fact, you don't understand, but at least you don't have to discuss with him. It's automatic.

FORTH carries a lot of this "context" on the parameter stack and on the return stack. The return stack is a marvellous place for storing the state of the machine. You cannot describe the state of the machine to a human being without dumping the contents of the stack. But the path you've followed through nested definitions to get to this place is a very good description. To a lesser extent, the numbers that happen to be on the parameter stack are a description of a different aspect of the current state of the machine. And both of those are invisible to the programmer, are handled automatically by the language. A new language wants to carry that same context invisibly. For instance, you might have pronoun references to things, where it is obvious to the human being what reference is being made. It needs to be equally as obvious to the machine. The two viewpoints need to be kept in agreement.

As the inventor of FORTH, do you find that people idolize you?

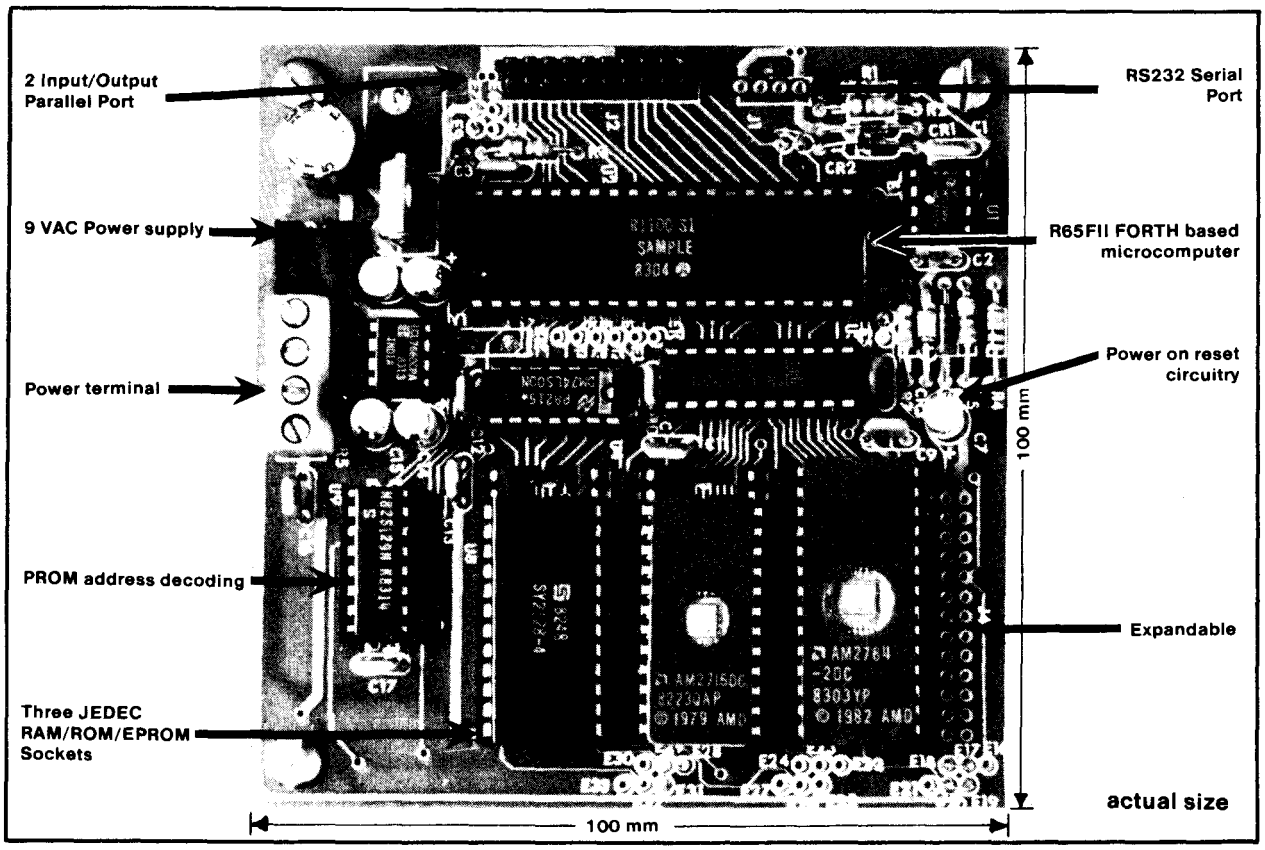
Yeah, more than they should. It's a cross I'm happy to bear, but there are two things worth noting. First, I did invent FORTH, but it was ten years or so ago. A long time. It is relevant to ask "What have you done recently?" And second, since then a lot of other people have contributed to FORTH. The FORTH we have now is far advanced over what I started with. The only reason I get the credit is that the remainder of the credit is diffused. If you want to point a finger, it's going to point at me because everyone else is harder to identify. From my point of view, FORTH is a tool which I developed for my own use, and that is still the way I see it. The FORTH I'm interested in is the one I'm using. I keep trying to find a more useful set of words that will make it even easier.

What motivates you?

Doing things well. That's true of a lot of scientists. Perhaps someone has addressed a problem in the past, but you see a way of doing it to one more decimal place. The decimal place is sufficient justification.

A PREMIER OFFERING TO THE FORTH COMMUNITY!

A limited number of R65F11 Microcomputer FORTH Development System - at a special price . . .



By the time you read this ad we should receive our first shipment of production R65F11 Microcomputers, the 6502 based single chip microcomputers with the run time portions of FORTH in ROM. This chip features a complete FORTH based operating system and is ideal for dedicated microcomputer applications. Our board, the NMIX-0011, surrounds the R65F11 with equally innovative circuitry that allows the chip to be a complete FORTH development system. (We call the board the "100 squared" for short, due to its extremely small size). All that is needed to do program development in FORTH is a CRT terminal or *microcomputer that speaks* RS232 (seven data, one start, two stop bits).

Look for a complete Euro card boardline coming soon -

The "100 squared" features on board rectification and regulation of power from a 9 volt AC or DC power source. Terminals are there if you prefer to use your own regulated 5V supply. An on board DC to DC convertor can provide negative voltage for the RS232 interface either way. Address decoding is accomplished by a bi-polar PROM that can be replaced by the user if necessary. A standard development PROM decoder is provided with the board. Three JEDEC 28 pin sockets are provided which will accept:

RAM's	*2016, 2128, 5517, 6116, 5564*
EPROM's	*2716, 2732, 2764*
EEPROM's	*2816A*

The board can program in circuit:

R2816A	*2764**

**requires additional VPP voltage supply.*

All this plus the powerful R65F11 which features:

- *FORTH kernel in ROM*
- *Enhanced 6502 CPU*
- *192-byte static RAM*
- *16 bidirectional, TTL-compatible I/O lines (two ports, R65F11)*
- *One 8-bit port with programmable latched input*
- *Two 16-bit programmable counter/ timers, with latches*
- *Serial port*
- *Ten interrupts*
- *Expandable to 16K bytes of external memory*
- *Flexible clock circuitry*
- *1 us minimum instruction execution time @ 2 MHz*
- *NMOS silicon gate, depletion load technology*
- *Single* +5V power supply
- 12 mW standby power for 32 bytes of the 192-byte RAM
- 40-pin DIP (R65F11)

We will be advertising very soon in the major trade journals. We anticipate demand to be so great that this will quickly become a limited availability item. We wanted to offer it first to the people that made the R65F11 possible - the people involved with the FORTH Interest Group. We are offering a special order price of $220.00. This is $30 off our list price, but, to reserve your board WE MUST HAVE YOU ORDER **NOW!** This is a *limited time offering!* ACT NOW.

Enclose Payment With Order To:

New Micros, Inc.
2100 N. Hwy. 360
Suite 1607
Grand Prairie, Texas 75050
(214) 660-1106
Telex 79-5551

I like to do things well. If I see someone else's word processor, I might say, "Yeah, he's got a couple of neat things there, but I see how I can put that in my word processor and it would be really nice." I'm motivated to do that. Particularly any neat, good idea that comes out in FORML or in the FORTH community, I will employ or add to my repertoire. Not just copying it, not just adding a subroutine, but thinking about it and maybe qualifying it a little bit to be more compatible with my way of doing things.

What do you do to get away from it all?

Go for a walk in the mountains. Last summer we took a hike in the Sierras. Used to have a boat, and would walk down to it at the harbor. I don't spend all my time at a keyboard, mostly because it takes a while for ideas to occur to me that are worth implementing. FORTH is nice in that regard. Anything I want to do I can do in an afternoon. It's almost never the case that a project would require any large amount of time to accomplish.

I enjoy driving. I'll get in the car and drive for six hours at the drop of a pin—usually on business, though. I'm going from here to see someone there, and it's just an excuse to be able to drive there instead of phoning or flying.

I escape to the books, I guess. Science fiction is my favorite—it's been a good year for science fiction: Asimov and Clark. Heinlein's been quiet for a year or two, since *The Number of the Beast*. I have to read that again, it was strange, fantastical. I think he has really said some good things. So I'm chasing after Heinlein, he's moving faster than I am. I'm not sure what he's saying in *The Number of the Beast*. I suspect there is something there.

I can get lost in a book like that for a day. But they can't write books as fast as I can read them. Even with all the authors I've got in my collection now, I can't spend too much time reading. It's not good, you're living in someone else's fantasy world, and the real world is dauntingly interesting in itself.

I wish I was more involved in some of the far-out endeavors. The space program, fusion, even something as mundane as the new fighter plane control systems that are supposed to both fly and fight the plane for the pilot. But none of those projects use FORTH in the sense that they ought to or that I would like them to. And I'm not really inclined to start fighting battles with people—it's such a long, hard sell. FORTH is still nibbling away at the underpinnings of technology, and it will probably prevail. But it will prevail not by edict from above but by infiltration from below.

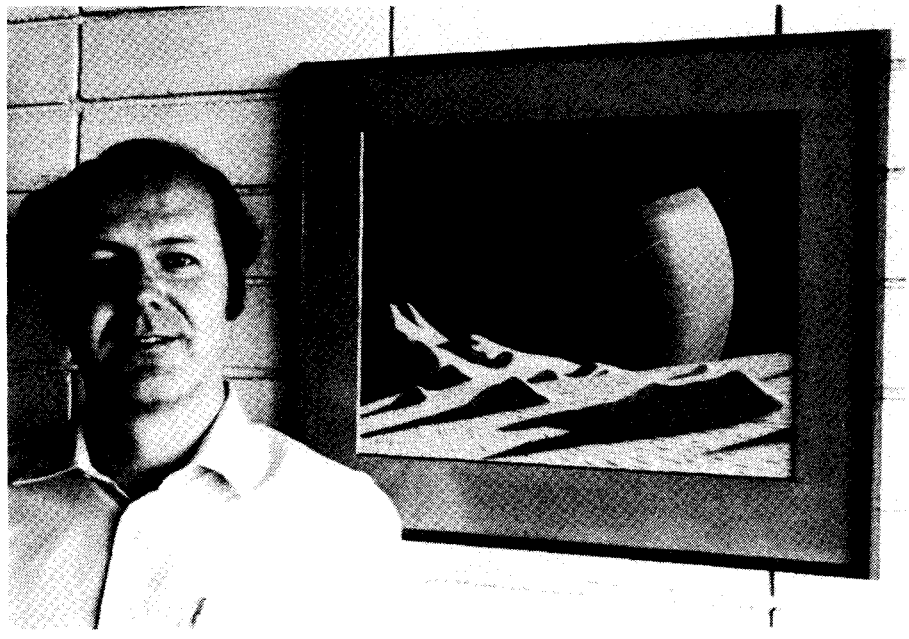
Do you see the concept of personal computing transforming our work, or our lives in general?

That's a very difficult subject. I don't know. I don't have a personal computer. I have use of this LSI-11, but I have never owned a computer. There is nothing I want a computer to do for me. It could turn on the television at prescribed times, but I can do that just fine. I can't promise in advance that I am going to want the television turned on. As a communications device that talks to information banks, I'm not involved in the professional life that requires access to information, so there is a limit to my information about the weather or the stock market, even bulletin boards. The thing I would like a computer to do for me is make phone calls. And it can, almost. In a few years I will be able to say someone's name and expect the computer to find them for me and put them on the speaker. And if it can't find them to make a note to try again later.

People as a whole—I think the computer has an important effect on their lives, but not a profound effect. No more so than, say, a television set. It may be that the purpose of the computer is in a cultural sense, that society needs computers in order to organize its affairs, but people—people don't.

So you don't necessarily endorse Toffler's third-wave vision of sweeping general changes, that computer technology can help to reverse the effects of the industrial revolution, e.g. pollution, commuting?

I was fascinated by these things in his thesis. On first reading, I sort of



nodded and said, "Yes, right on." But I don't think people are that malleable. There will be people who sit at home and work from home. Like myself. But I don't do it over a communication network through some computer in downtown Los Angeles, I just do it all by myself. When I finally get a product I will take it and personally interact with the people I am working with. I'm ambivalent about the place of computers in society. I don't really like computers in the sense that a youth might love a car. I think I more hate computers because they keep breaking and being awkward to deal with. They never live up to my hopes. I did think of one use for a computer, though. Back in the Sierras there were mosquitoes. I could see a little solar-powered or laser-based zapper I wear on my head, that shoots mosquitoes. And any mosquito that comes within two feet of me is dead!

And make sure it knows the difference between mosquitoes and people...

Is your line of sight clear? Supposing you miss! I got a hair that time... I think that is very feasible, especially if the computer was accurate enough that it could, say, shoot the wing right at the narrow joint. It wouldn't take much energy but it would disable the mosquito so that it was no longer a nuisance. I would rather have a computer doing that than shooting down ICBMs. I think that is much more personal, but that's not the kind of thing people have in mind.

Any general comments?

You have to get inside the world of software, the world of imagination; and you create your own world. You create your own problems and so it is almost frightening, the amount of power you've got. I've always had the sort of dream—I've always favored software over hardware. The hardware is the nuisance you have to put up with in order to have software. If you want to talk about a religious aspect of FORTH, it is to say, "Do you need hardware? Can you conceive of a way of representing these ideas, of making these castles in the air without any particular underpinning?" I'm sure that chips will get down to molecular size, but there is still going to be that matter at the core of things. Can you take energy fields, somehow, and weave them to make software? Maybe that's what the layers and layers of operating systems and languages are doing, taking you so far away from the hardware that you forget it is even there.

FORTH: Cheaper than Hardware

Peter J. Lagergren
Grand Prairie, Texas

Many articles have been written about the use of FORTH, as compared to other languages. However, we know of no articles that show the replacement of hardware components with FORTH software. Clearly, FORTH has intrinsic advantages over other programming languages when dedicated applications are involved. This paper describes a hardware-based system and the reduction in system cost and an increase in system reliability that resulted by converting to microprocessor control with FORTH as the programming language.

During this project, we faced the common problem of deciding whether to continue development on a hardware-based system which had initially promised to deliver the final product in a relatively short period of time at comparatively low unit cost. As the project progressed, the initially perceived difference in cost between producing units with a hardware base versus microprocessor control continued to narrow, and eventually it became obvious that a FORTH-based microprocessor-controlled system was not only inherently easier to design but was going to be cheaper to develop and produce than the hardware-based system. We estimate that the FORTH system required only one-fourth the development time that the hardware-based system consumed. Our data indicates that a completed hardware-based system would cost approximately $200,000. When performed with a FORTH-based microprocessor-controlled system, the same project will cost slightly less that $15,000 ($60,000 if done by a large company). This surprising result materially altered our attitude regarding the optimum approach.

The project was started at the request of a customer who operates large towboats on the Mississippi River. The customer was interested in determining the fuel consumption versus speed for these towboats. The vessels involved are relatively large, typically on the order of 150 to 175 feet in length, powered by two or three diesel locomotive engines. Combined horsepower ranges between 6000 BHP and 10,000 BHP. With typical fuel burns of 7,000 to 13,000 gallons of fuel per day, the ability to successfully manage these vessels' power/speed settings is of some importance. The algorithm chosen for the problem was relatively simple: compare net temperature-corrected fuel burned to speed made good through the water.

The primary difficulty in the program was the provision of vessel speed through the water. This problem has, until now, been economically intractable due to the turbulence associated with pushing a tow that approaches 3/8 mile in length; the flow around the hull is as likely to be moving forward as backward, rendering traditional speed-measuring devices useless. Thus, pilots of these multi-million dollar vessels often monitor their velocity by gauging their speed past fixed objects on the river bank. Navigation by the Mark I eyeball!

We felt that locating our speed sensors with the existing depth sounder would be perfect, since they then would be operating in undisturbed water ahead of the tow mass. It did seem reasonable at the same time to design a more rational solution to the depth-sounding requirement. If we could place a low-power transducer/transmitter/receiver at the tow head and use radio telemetry to send the generated data to the bridge, we could combine our speed information into the data stream without difficulty. We could save the customer the substantial sums spent on cable maintenance and provide a safer working environment for the deckhands, who had to string two coax sections the entire distance at each tow change, irrespective of time of day or weather conditions.

Initial trials of the system established both its utility and the correctness of our assumptions. That was the good news. The bad news was that the system could not discriminate between good returns and spurious echoes or noise. Obviously, the problem couldn't be completely intractable, since the depth sounders in use did have acceptable performance levels. However, currently available units are quite large, both in physical size and in component density. At this point, we entered a vicious cycle of test, analyze, redesign, and retest. It became apparent that to produce acceptable performance with hardware we would eventually be driven to the same level of complexity and cost as the units currently available. A better solution to the problem was clearly required.

With the advent of the Rockwell R65F11 FORTH-based microprocessor, we had the tool required to rapidly get up the development curve to an acceptable and, hopefully, technologically advanced product. We replaced all the discrete components of the original design, except the pulse-generating unit, with a 100mm x 100mm board with the R65F11 as the CPU. All the control, analysis, data formatting and system status functions which had been performed by five densely packed CMOS boards essentially were replaced with one FORTH-based micro, a latch and a ROM. As before, the data was delivered to the radio link for transmission. Use of FORTH as the programming language clearly was desired since the lead time to an acceptable program using machine code was far too long for the project requirements. The system architecture is as follows:

The front-end unit uses a program which shows FORTH at its maximum flexibility. The pulse width, transmit cycle control functions, variable gain control and receiver pulse discrimination are performed in machine code so that the execution of

these functions is limited to the minimum time required, freeing the processor for the far more important analytical functions, which are run in FORTH. For our purposes, we considered the speed of sound through water invariant, so that we could generate all calculated functions by referring to the system clock cycle time. The cycle begins with a 600 volt p-p transmission pulse with a short 832 microsecond pulse width. The FORTH master program then stages the gain control voltage over a parabolic gain schedule for a forty-millisecond period. The system counts the number of returns to determine if a "good" return was actually received. This information is then used to calculate a new gain schedule for the next pulse cycle so that the system will see only one return on each pulse cycle. If the system does not find an echo on the first several cycles, it begins a range-gated search mode that slowly increases the gain schedule and the pulse width of the transmitted pulse until a bottom echo is found. The gain schedule is predicated on the expectation that the rise and fall of the bottom of a waterway usually will not exceed thirty degrees of slope, which generates vertical velocities on the order of twenty feet per second or so at typical towboat velocities.

The FORTH range-gating routine also generates a single-digit number between zero and nine which is a magic number that we call the "confidence level." This number is a complex function based on a number of factors such as first hit in gate or out of gate, multiple hits, gain setting required to attain at least one hit and several other factors. This confidence level is then used to scale the gain and transmit schedules. The information in these schedules is then loaded into the machine code program which actually does the time-critical transmit/receive function. Clearly, the ability to perform these quite sophisticated analyses and to transfer control data directly into a machine code program is unique to FORTH. Without using the power and flexibility of FORTH, the lead time to successfully running this program would be quite long.

Subsequent to the receive/analysis cycle, the FORTH master program generates a speed value from the concurrently operating speedometer. At the end of each cycle the system assembles the depth, speed and confidence data into the data stream and, when requested by the bridge-mounted display unit by the transmission of a unit-unique security code, supplies the data to a modem/RF unit. After completion of the transmission, the unit reverts to the depth sounder/speed/listen mode. Polling takes place at a several cycles-per-second rate.

The bridge-mounted R65F11-based display unit serves as the overall system master by polling the one or two front-end units and reducing the data received via the modem. The CPU then outputs the depth/speed data to the LED drivers for the front

(Continued on page 32)

Recursive Sort on the Stack

Dr. Richard H. Turpin
Purdue University
Indianapolis, Indiana

The assigned task was to sort all the numbers on the stack and output them in increasing order (largest printed last). The solution is a recursive sort as explained below.

Two words were defined. The first, **SINK**, moves the largest item on the stack to the bottom. It is recursive and uses the return stack for temporary storage of data items. Basically, if the stack is more than one number deep, **SINK** moves the smaller of the top two stack items to the return stack. When **SINK** finds only one item on the stack it does nothing. In unfolding the recursion, all the items are retrieved from the return stack.

In the first pass **SINK** moves the largest item to the bottom of the stack. With each subsequent pass **SINK** moves the next largest item down in the stack. **SORT** executes **SINK** a number of times equal to the number of items on the stack minus 1, leaving the items ordered from largest at the bottom to smallest at the top. Finally, **SORT** outputs the numbers in the desired order.

```
: SINK DEPTH 1 > IF OVER OVER  ( If more than one no.  on )
       < IF SWAP THEN          ( stack then move smaller )
       >R                      ( item to return stack. )
       MYSELF                  ( SINK through stack )
       R>                      ( then back out )
       THEN ;


: SORT DEPTH 1- Ø DO SINK LOOP ( Sort numbers on stack. )
       DEPTH Ø DO . CR LOOP ;  ( Print them out. )

Example:    25 -4 326 1 25 Ø -628 75 SORT
            -628 -4 Ø 1 25 25 75 326
```

As defined above **SORT** always sorts the entire list of numbers whether it needs sorting or not. With the addition of a **FLAG** as shown below, **SORT** executes **SINK** at least once, but no more than is necessary to place the data in order.

```
VARIABLE FLAG
: SINK DEPTH 1 > IF OVER OVER
       < IF SWAP Ø FLAG ! THEN  ( If do swap, clear FLAG )
       >R MYSELF R> THEN ;

: SORT DEPTH 1- Ø DO 1 FLAG !   ( Set FLAG. )
       SINK                     ( Run through data. )
       FLAG @ IF LEAVE THEN     ( If FLAG still set then )
       LOOP                     ( no swap performed so )
       DEPTH Ø DO .  LOOP ;     ( sort is complete. )
```

End Listing

Tracer for Colon Definitions

Rieks Joosten
State University of Utrecht
The Netherlands

A tracer is a tool that can be very useful in debugging routines, by printing the names of the routines that are executed and printing stack contents. This paper describes a way of implementing such a tracer.

Introduction

There are many things a tracer can do, such as printing the names of executing routines, dumping the (arithmetic) stack before and after execution of the routine, dumping the return stack, user values, etc. It seems that such a tracer would be the tool that enables a programmer to exactly pinpoint the bug(s) in his routines, but the risk he runs is that he cannot do this if he gets swamped by the information supplied by the tracer. The highest form of a tracer would be one which tells you where things "went wrong." Such a tool has not been made yet.

It is my experience that when you print the name of a routine that executes, together with the (arithmetic) stack dump before and after the execution of this routine, you will have more than enough data to distill the bug(s). This will give you the opportunity to check the syntax of the new definition. Also, when you create compiler directives, you can watch them compile!

Using a Tracer

It is very simple to create a word that will print the name of the calling routine, when this routine itself is compiled as the first routine in the calling routine, *e.g.*,

```
: <WORD>
(TRACE:) <ROUTINE.1> <ROUTINE.2>
... <ROUTINE.N> (TRACE;) ;
```

where **(TRACE:)** is the routine that will print the stack contents as well as the name of the calling routine, being **<WORD>**. The stack contents after execution of **<WORD>** is printed by **(TRACE;)**, which is the last routine that is called by **<WORD>**.

There are two ways to ensure compilation of **(TRACE:)** and **(TRACE;)**. The first method is very laborious: editing the words into the source code before compilation. After debugging, the editing should be re-done to remove the debug facilities. Also, it does not guarantee that the routine **(TRACE:)** get executed as the first routine in the word, which is vital for its correct operation in printing the name of its calling routine. Also, there is no guarantee that **(TRACE;)** will be the last word compiled before the semi-colon, which may foul up the trace later on. Its implementation, however, is straightforward.

The second way is to redefine : and ; in such a way that they will compile **(TRACE:)** and **(TRACE;)**. This has no extra editing as a consequence, and it also guarantees that **(TRACE:)** and **(TRACE;)** are the first and the last routines executed respectively. The problem is how to redefine : and ; since while redefining them you will need both the "old" : and ;. In spite of the implementation difficulty, the latter suggestion is more attractive.

Redefining : and ;

Two possibilities are given here: first, you can redefine the : and ; in another vocabulary, and have them unaltered redefined. This way, you either use the : and ; in this new vocabulary or the ones in FORTH. When you redefine : and ; in the FORTH vocabulary to incorporate the tracer, you can use the "old" : and ; for the redefinition.

The second way is to define two words X and Y, and define them as you would have redefined : and ;. Then you change the names of X and Y into : and ; by writing the ASCII values of : and ; over the X and Y characters in each namefield. This requires knowledge of what the header looks like, but this is specified in most FORTH systems.

I chose the latter, because it is shorter code and at the time of creation, I only had one block available to put the source code into.

Definition of Words Used

The routines that you may not be familiar with are:

TNAME<CFA> --- < >
Prints the name of the routine whose code field address is on top of the stack.

ASCII-:< > --- <ASCII Value of : >
Leaves the ASCII value of the character : on the stack.

ASCII-;< > --- <ASCII Value of ; >
Leaves the ASCII value of the character ; on the stack. **VALUE** is a defining word that is used in the same way as **VARIABLE**. It creates so-called "to-constants" or "to-variables" (also called values) that obey the **TO** concept.

NFA<PFA> --- **<NFA>**
Changes the parameter field address of a routine into the namefield address of the same routine.

Note: The routines **ASCII-:** and **ASCII-;** actually are combined into one routine **ASCII-$** where **$** can be any character. Note: In the next code the word ." is used in the following syntax: ." "**<TEXT>**" where only one space is allowed between the quote in ." and the first text delimiting quote. This has been done because I think it is better syntax. It works because **WORD** will skip leading delimiters, and also it will skip the first trailing delimiter (Ref.: "Thoughts on the 79-Standard," *Proceedings of the Rochester FORTH 70-Standard Conference,* May 1981).

Summary

The tracer described here gives the possibility of optionally tracing words

whose source doesn't differ from 'normal' FORTH source code. Features like indentation make it easy to tell where FORTH is executing, and which routines spoil the correct execution.

Acknowledgments

I would like to thank Paul "Hank" Hamilton for the valuable ideas he gave me when working on this tracer, and Lawrence P. Forsley for reviewing this work; both are of the Laboratory for Laser Energetics at the University of Rochester. I would also like to thank the University of Rochester Computing Center for the use of an Apple II computer in preparing this work.

Letters *(Continued from page 4)*

undefined words. Please give careful specifications for those undefined words. I'm new to FORTH, and you can't assume that I know how to fill in the gaps. And I could use more help with files and I/O. We need a relational data base management system and DBMS services that can be requested from within FORTH programs. Is there a SFIG (Special FIG) on data bases?

The theoretical stuff in *FORTH Dimensions* is great. Now can you include some practical working-class material on how to write and debug useful programs? I'd also like to see some reviews and comparisons on the FORTH systems offered by different vendors. I hope to use FORTH on an IBM-PC and a PC lookalike, but I don't know what the different systems offer, which are more stable, powerful, easy to use, etc.

Keep the good FORTH stuff coming. This letter, by the way, was printed with QTF.

Dave Kuhlman
1821 P Street #2
Sacramento, CA 95814

Thanks for the input. We are issuing a call for more utilities and applications for publication in FORTH Dimensions. *We cannot strictly enforce a standard for* FORTH *words which appear in our pages, as some readers have requested, but we will ask authors to document them more carefully and to include check sums.—Ed.*

THE TRACER SOURCE CODE

```
0 VALUE ?TRACE          ( flag telling whether tracer is active or not  )
0 VALUE #INDENT         ( contains the number of indentation spaces     )


: CR#IN CR #INDENT SPACES ;     ( performs a CR and indents             )


: TRACE ?TRACE 0= DUP TO ?TRACE    ( toggle tracer flag. If tracer is   )
        IF   0 TO #INDENT  THEN ;  ( turned on, the indentation is reset)


: STACK CR#IN ." "STACK : "     ( print the contents of the stack, or   )
        DEPTH ?DUP              ( print 'stack empty'. If the contents  )
        IF 1 SWAP               ( of the stack is printed, the element  )
           DO    I PICK . -1    ( on top of the stack is printed right- )
           +LOOP ASCII-# EMIT    ( most. A '#' character is printed to   )
        ELSE ." "EMPTY " THEN ; ( prevent any other output to trouble   )
                                ( the picture of the stack              )


: (:)   ?TRACE
        IF   STACK                  ( If in trace mode, print the stack )
                                    ( and print the name of the calling )
                                    ( routine.                          )
             CR#IN R 2- CFA TNAME   ( Special for pre-increment systems )
           ( CR#IN R    CFA TNAME) ( for post-increment systems         )
             1 +TO #INDENT          ( increment the indentation counter )
        THEN ;


: (;)   ?TRACE                     ( If in trace mode, decrement the   )
        IF   -1 +TO #INDENT STACK  ( indentation counter and print the  )
        THEN ;                     ( contents of the stack             )

: X     [COMPILE] : COMPILE (:) ;   IMMEDIATE     ( redefine :         )
: Y     COMPILE (;) [COMPILE] ; ;   IMMEDIATE     ( redefine ; :       )

ASCII-:   ' X NFA 1+   C!       ( Store : in the namefield of X         )
ASCII-;   ' Y NFA 1+   C!       ( Store ; in the namefield of Y         )

( The last two lines change the headers of the routines X and Y into    )
( headers of : and ;
```

End Listing

A Simple Multi-Tasker

Ray Duncan
Los Angeles, California

This article presents a simple I/O-driven multi-tasker, written in FIG-FORTH, which allows one to execute several background tasks "concurrently" with one foreground task. It can be added to your FORTH system very easily, if your system uses vectored I/O or you have the source code for the nucleus. The approach shown here is quite unsophisticated in comparison to the multi-tasking facilities in poly-FORTH, and does not allow for multiple users. However, it considerably simplifies programming in applications which need to rapidly poll several peripheral devices and still remain responsive to operator input, and can also be used to support a print spooler.

The heart of the program is the routine **TASKER**, which is repeatedly called by the word **KEY** whenever it is waiting for keyboard input. **TASKER** examines **TASK_LIST**, whose length is defined at compile time by the constant **MAX_TASKS**. Each word in **TASK_LIST** is either zero or the CFA (code field address) of a background task. The background tasks are simply **EXECUTE**d in a round-robin fashion.

The foreground task has ultimate control of the keyboard, video display and disk drivers. By the term "foreground task" we are referring to any task which was invoked by simply entering its name followed by a carriage return.

Background tasks designed by the user must obey certain rules in order for the system as a whole to perform properly:

1) Each background task must be self-contained. It must leave the parameter and return stacks balanced (*i.e.* no extra values must be consumed or left behind). If any information must be maintained from one invocation of the background task to the next, it should be kept in a local variable.

```
Screen # 0
  0 ( Multi-tasker                                          12/01/82 )
  1
  2 DECIMAL
  3 10 CONSTANT MAX_TASKS    ( define max background tasks allowed)
  4 0 VARIABLE TASK_LIST     ( allocate task list, with extra )
  5 MAX_TASKS 2* ALLOT       ( zero position in list for KILL word)
  6 TASK_LIST MAX_TASKS 1+ 2* ERASE   ( initialize at compile time)
  7
  8 ( display names of all active tasks )
  9 : .TASKS          CLEARSCREEN ." Active tasks:"
 10                   MAX_TASKS 0
 11                   DO  I 2* TASK_LIST + @ ?DUP
 12                       IF   CR  I 1+ 2 .R  2 SPACES
 13                            PFA NFA ID.   THEN
 14                   LOOP  CR CR ;
 15 -->
```

```
Screen # 1
  0 ( Multi-tasker, cont.                                   12/01/82 )
  1
  2 ( examine the task list to match a task's code field )
  3 ( address a.k.a CFA, return pointer and/or flag.  )
  4 ( if found:       cfa --- addr t )
  5 ( if not found:   cfa --- f      )
  6
  7 : FIND-TASK       0
  8         BEGIN     DUP MAX_TASKS -
  9         WHILE     DUP 2* TASK_LIST + DUP @ 4 PICK =
 10                   IF  ROT ROT 2DROP 1 EXIT
 11                   THEN DROP 1+
 12         REPEAT    2DROP 0 ;
 13
 14 -->
 15
```

```
Screen # 2
  0 ( Multi-tasker, cont.                                   12/01/82 )
  1
  2 ( Add a background task to the TASK_LIST )
  3 ( Used in the form:  START cccc )
  4
  5 : START           ( --- )
  6         -FIND
  7         HERE COUNT TYPE
  8         IF    DROP  CFA   0 FIND-TASK
  9               IF    !       ( *** store CFA into task table *** )
 10                   ."  --- added to task list"
 11               ELSE ."  --- too many tasks" DROP
 12               THEN
 13         ELSE ."  --- not in dictionary"
 14         THEN CR ;
 15 -->
```

```
Screen # 3
  0 ( Multi-tasker, cont.                                    12/01/82 )
  1 ( Remove a task from TASK_LIST )
  2 ( used in the form:  KILL cccc )
  3 : KILL  -FIND
  4         HERE COUNT TYPE
  5         IF  DROP CFA  FIND-TASK
  6             IF    DUP 2+ SWAP
  7                   ( drag the rest of the task-list back over )
  8                   ( the task being removed, with zero at end)
  9                   TASK_LIST MAX_TASKS 1+ 2* + 3 PICK - CMOVE
 10                   ."  --- removed from task list"
 11             ELSE  ."  --- not in task list"
 12             THEN
 13         ELSE      ."  --- not in dictionary"
 14         THEN CR ;
 15 -->
```

```
Screen # 4
  0 ( Multi-tasker, cont.                                    12/01/82 )
  1
  2 : TASKER          MAX_TASKS 0
  3                   DO      I 2* TASK_LIST + @ ?DUP
  4                           IF   ( save current cursor position)
  5                                ( in case background task must )
  6                                ( write to screen )
  7                                SAVE_CURSOR
  8                                ( transfer to slave task )
  9                                EXECUTE
 10                                ( restore cursor position )
 11                                RESTORE_CURSOR
 12                           ELSE LEAVE   ( position empty, quit)
 13                           THEN
 14                  LOOP  ;
 15
```

```
Screen # 5
  0 ( Multi-tasker, cont.                                    12/01/82 )
  1
  2 ( read character from keyboard )
  3 ( stack effect:    --- b )
  4 : KEY             BEGIN
  5                         TASKER              ( cycle through)
  6                                             ( background tasks)
  7                         ?TERMINAL           ( is keyboard ready?)
  8
  9                   UNTIL                     ( if not loop )
 10                   USER_KEY @ EXECUTE ;      ( vectored execution )
 11                                             ( of actual keyboard)
 12                                             ( input routine )
 13
 14
 15
```

End Listing

2) A background task must execute to completion in a "reasonable" length of time, otherwise the user will perceive keyboard responses to be delayed. If all of the necessary work cannot be done quickly enough, it should be partitioned into two (or more) background tasks which pass data through local variables.

3) Background tasks should not, in general, attempt to access the disk or keyboard, unless you know for sure that the drivers for those devices are reentrant.

The words used to control the task list are as follows:

KILL ---
Used in the form:
KILL name <return>
Removes a task from the background task list.

START ---
Used in the form:
START name <return>
Adds a task to the background task list.

.TASKS ---
Displays the names of all active background tasks.

For more flexible operation of the simple multi-tasker, the dispatcher **TASKER** can also be invoked from such words as **?TERMINAL** and **BLOCK**. In applications where the foreground task is compute-bound rather than I/O-bound, another word (perhaps named **PAUSE**) should be provided so that the foreground task can give up control to the background tasks at appropriate intervals.

As they say in the computer science courses, elaboration of this multi-tasking facility is left as an exercise to the reader. Possible enhancements include the assignment of priorities to background tasks; periodic invocation of the dispatcher word **TASKER** from a real-time clock interrupt service routine; and, finally, the creation of a true multi-user, multi-tasking capability through user variables and separate stack areas.

Acknowledgments

Marty Petri, a FORTH consultant in Van Nuys, California, provided the original idea for this round-robin multi-tasker.

Simple FORTH Multi-Tasking Environment

Martin B. Petri
Van Nuys, California

I recently completed a data acquisition system (DAS) for a client, built on Z80 FORTH and Hardware Floating-Point Extensions from Laboratory Microsystems. This DAS is installed at a solar energy site built by the Jacobs Engineering Group for the Home Laundry Company in Pasadena, California.

This solar energy system was funded by the U.S. Department of Energy as a demonstration project. It is the first application of solar energy in the state of California by the Industrial Process Heat branch of the DOE for "Solar Production of Industrial Process Steam." The system furnishes 105 psi steam for laundry and dry cleaning applications, meeting 25% of the laundry's annual steam needs and resulting in savings of approximately 300 barrels of oil each year.

The solar system consists of 406 linear, parabolic-trough collectors mounted on a lightweight steel structure which spans the laundry's parking and storage area at roof height. The flow through these collectors and through the various pumps is monitored by a network of temperature, pressure and solar radiation sensors which are, in turn, monitored by a device called a "data logger." The data logger closes the control loop by electrically commanding the operation of valves and pumps to maintain the operation of the system.

The purpose of the data acquisition system, which is resident in an external Z-80-based microcomputer, is to interrogate the data logger at regular intervals. It then collects, formats and stores the resulting data, and uses this data to create reports for feedback to the Department of Energy.

During each data acquisition cycle, which occurs every two minutes, twenty-one calculations are performed. These results are saved in a file along with the original data collected. At the end of each hour, the data is read back and further calculations are performed. These hourly results are then saved as a record in a separate file. At the end of the day's operations, all hourly results are crunched and the resulting summary is appended to the end of the hourly reports.

To make things even more complicated, the client required several functions which he could turn on or off at will. These included a printout of the raw data collected at two-minute intervals, a tally of the calculations, trending of any given sensor, and a printout of the daily report for any given day within the month. Also required was the ability to interactively use the system to calculate functions which were not previously defined, emulating a pocket calculator.

One data acquisition cycle takes approximately seventeen seconds, leaving a time gap of one minute and forty-three seconds before the system is required for the next cycle. With this in mind, a multi-tasking environment with set priorities could be utilized effectively to operate during the time gap.

The priority scheme must be:

1) Data collection at an interval set by operator
2) Operator keyboard input
3) Any electives

Basically, the tasker is round-robin in nature. Each task is a previously defined FORTH word which executes entirely before control is passed back to **KEY**. While the system is waiting for keyboard input, it executes tasks until a character is ready. The CRT screen, with direct cursor addressing, becomes the visual workplace of the tasker. Each function of the application program is assigned as a task (*e.g.* collect, sort, crunch, save data). This means that the operator has direct access to every pre-defined word as well as the ability to define new words while in the multi-tasker environment. Very interesting!

Print spoolers are easily incorporated as a task. Simply move the data to a protected buffer and output one character from the buffer at each task cycle. In this manner, lengthy reports can be printed without tying up the system.

A Real-Time Debugger?

Variables containing the results of intricate calculations can be examined interactively or, in a monitoring mode, they can be defined as a task. With this method, the contents of the variable **ANSWER** will constantly be displayed on the CRT at coordinate x,y without causing any significant degradation of application speed.

```
: ?ANSWER x y GOTOXY
  ." Answer = " ANSWER ? ;

START ?ANSWER
```

The power to interrogate, display and modify during real-time program execution qualifies the Multi-Tasker as the second most valuable tool in my bag of tricks.

The first, of course, is FORTH.

Techniques Tutorial

Meta Compiling II

Henry Laxen
Berkeley, California

In Volume IV, number 6 of *FORTH Dimensions* we took a look at one of the underlying foundations of the Meta Compiling process, namely that of mapping the address space of the target system into the address space of the host system. If you don't understand the above sentence I suggest you reread the last article on Meta Compiling. This article will use the technique of address mapping discussed last time to implement the "guts" of a meta compiler. I must warn you that I am still leaving out a lot of details, which I will try to cover in the next article. This article will illustrate how to meta compile code definitions and simple colon definitions.

First, let's take a look at how we can meta compile code definitions. What we need is an assembler which generates its code in the target system. We discussed that briefly last time, and showed how to define a **JMP** instruction so that its opcode and address would be assembled in the target system rather than in the host. But that is only half of the battle. When we define a code word, we are defining a name in the target system, which must appear in the target dictionary. Also, if a future colon definition references this code word, we want to compile the code field address of this word in the target dictionary. Let's take this one step at a time. First we must define a name in the target dictionary. There is no magic in this; it depends heavily on the structure you decide on for names. The only thing you have to be careful about is to make sure the bytes are placed in the target system address space, and that you keep track of the link fields somehow. Figure One illustrates a word that will compile FIG-like headers into a target system. Notice that if we want to compile headerless code all we need to do is set **WIDTH-T** to zero. Headerless code is one of the fringe benefits of meta compiling, not its main purpose.

Now that we can lay down headers in the target system, we need to think about what happens when a word defined in the target system is supposed to be compiled in a colon definition. For example, suppose we have defined **DUP** and **+** as code words, and have meta compiled them so that their headers and their code bodies are resident in the target system. Now we want to define the word **2*** as follows:

```
: 2*  DUP + ;
```

What is supposed to happen? Well, we need to make the meta compiler do the same thing that would ordinarily happen in a running system; namely, **:** should create a name in the dictionary and should compile the code field addresses of **DUP** and **+** and **EXIT** into the parameter field of **2***. Finally, compiling should be terminated by the **;**. Notice that the behavior of **DUP** and **+** in the meta **:** context is totally different from their behavior in the normal FORTH context, namely **DUP** should duplicate something and **+** should add something. In the meta **:** context they compile something.

Now that we know what should happen inside a **:** definition, it is just a matter of implementation. The approach at this point is to construct a symbol table of all the words that are defined, and when building a **:** definition, to look up each word in this symbol table and compile the code field address corresponding to this word into the target system. In Pascal or BASIC this would probably require twenty pages of code. The skeleton FORTH code that does this is written in Figure Two. It probably requires a few words of explanation. First, **>IN** is the pointer into the input stream, which must be saved and restored because both **HEADER** and **CREATE** modify it. **IN-SYMBOLS** isn't defined elsewhere, but I will describe its function. It is highly dependent on how your system implements vocabularies. It must make sure the word that **CREATE** creates is in a sealed and separate vocabulary. This is very important, since almost all of the FORTH nucleus words will be defined during the meta compiling process, and it would be deadly to redefine them all in the FORTH vocabulary. Also, it guarantees that when a symbol is looked up, the meta one is found, not the corresponding FORTH one. The idea is to use the existing vocabulary structure to implement a symbol table by placing all of the meta names into it and sealing it so it does not chain to any other vocabularies. If we adopt such a structure, the regular **CREATE** can be used to enter a symbol into the symbol table and the regular **FIND** can be used to search it. Never buy what you can steal! Similarly the **IN-META** restores things back to the way they were.

Now let's look at the next phrase, namely **HERE-T**, and figure out what it is doing. It is saving the current address in the target system into the parameter field of the symbol that was just created. Since the header has already been created, **HERE-T** is the code field address of the word that has just been created. So by saving it in the parameter field of the symbol, we are remembering the code field address for future reference. This future reference takes place in the run-time portion of the definition. **MAKE-CODE** does nothing more than fetch the code field we just saved and compile it into the target system. Thus, if we use the meta definition of **CODE** that appears, we see that what it does is create a **HEADER** in the target system, as well as a symbol in the symbol table. Furthermore, it sets up the code field in the target system to point at its parameter field, just as every good code field should in an ITC system. Finally, it switches to the **ASSEMBLER** vocabulary

to allow compilation of machine-language opcodes. Later, if the word defined by **CODE** is executed, the **DOES>** portion comes into play, which simply compiles itself into the target system. Nothing could be simpler or more devious!

Now let's finish this discussion by looking at what must take place when we meta compile a : definition. Take a look at the code in Figure Three. First we must create a header and a symbol, just as we did for **CODE** words. Next we must lay down the address of the runtime for : in the code field of the word being defined. (I have assumed that **NEST** is a constant that returns that address for me.) Finally, we must enter a loop that looks words up in the symbol table and compiles their code fields into the target system. That function is performed by the meta version of]. The function of compiling the code fields is cleverly performed by executing the words that are found in the symbol table. That is why the **DOES>** portion of **TARGET-CREATE** compiles the code field that was saved. I have not provided code for the definition of **DEFINED** and **NUMBER-T** but I leave to your imagination what it is that they do.

The code I have presented is very simplistic, and is not really adequate as is. However, it does contain the central ideas that are needed in order to implement a meta compiler. Next time we will look at some of the subtler issues in meta compiling such as how to handle **IMMEDIATE** words, and what about **[COMPLETE]**? Until then, good luck, and may the FORTH be with you.

Henry Laxen is Chief Software Engineer for Universal Research, 150 North Hill Drive #10, Brisbane, CA 94005, specializing in the development of portable computers.


```
Scr # 54
  0 \ Fig 1.   Headers in Target System                    13MAY83HHL
  1 : S,-T   (S addr len -- )
  2    0 ?DO   DUP C@ C,-T   1+   LOOP   DROP   ;
  3 VARIABLE WIDTH-T
  4 VARIABLE LATEST-T
  5 : HEADER   (S -- )
  6    BL WORD   C@ 1+ WIDTH-T @ MIN   ?DUP IF
  7       HERE-T HERE ROT S,-T       ( Lay down name, save NFA )
  8       LATEST-T @ ,-T   ( Link Field )   DUP LATEST-T !
  9       128 SWAP THERE SET   128 HERE-T 1- THERE SET
 10       ( Set the high order bits at each end of the Name )
 11    THEN   ;
 12
 13
 14
 15
```

Figure One

```
Scr # 55
  0 \ Fig 2.   Create a Target Image and Symbol            13MAY83HHL
  1 : MAKE-CODE   (S addr -- )
  2    @ ,-T   ;
  3 : TARGET-CREATE   (S -- )
  4    >IN @   HEADER   >IN !    ( Without moving input stream )
  5    IN-SYMBOLS CREATE IN-META   HERE-T , ( Save cfa )
  6    DOES>   MAKE-CODE   ;
  7 : CODE   (S -- )
  8    TARGET-CREATE   HERE-T 2+ ,-T   ASSEMBLER   ;
  9
 10
 11
 12
 13
 14
 15
```

Figure Two

```
Scr # 56
  0 \ Fig 3.   High Level Meta Definitions                 13MAY83HHL
  1 : ]   (S -- )
  2    BEGIN
  3       DEFINED IF   EXECUTE   ELSE   NUMBER-T   THEN
  4    AGAIN   ;
  5 : :   (S -- )
  6    TARGET-CREATE   NEST ,-T   ]   ;
  7
  8
  9
 10
 11
 12
 13
 14
 15
OK
```

Figure Three

Review

The R65F11 FORTH Chip

Randy Dumse
Grand Prairie, Texas

The advent of the R65F11 FORTH-based microprocessor signals a new era of low-cost hardware tailor made for dedicated applications. This microcomputer hosts not only a very impressive list of hardware features, but also the run-time portions of the FORTH language and an operating system designed specifically for dedicated applications.

The kernal of FORTH in ROM consists of 133 words available for user definitions. These words are headerless, kept in the format of code field address followed by the parameter fields. About half of the definitions are machine coded. The remainder are high-level definitions. A typical FORTH definition executes in under 100 microseconds (BASIC is ten to a hundred times slower). Although a micro monitor is included in the internal ROM as part of the operating system, programming the R65F11 usually requires the use of the R65FR1 development ROM. This port contains the rest of FORTH, the non-run-time words and the heads of the words from the kernel. Many utilities specifically designed for easy development of stand-alone programs are also added. The development ROM even supports target compiling of headerless code.

It is possible to operate RSC-FORTH (Rockwell Single Chip FORTH) with separated heads and codes. This is an offshoot of the structure of the word definitions themselves. This method was first conceived by Mark Reardon of Rockwell International and is the factor that allows the run-time kernel to reside in internal ROM without the normal dictionary overhead. To accomplish this, one additional field is added to the FORTH word structure. The familiar length, name and link fields (which make up the heads of definitions) are separated from the code and parameter fields (which make up the code portion of the word) by an indirect pointer called the Parameter Field Address Pointer (PFAPTR). The entire definition may be in one place, or it may be separated into two parts. It is always possible to find the code from the head. (The reverse is not true, however.)

Programmers of dedicated applications will find the selection of words rich in content. All the math and stack functions of FIG-FORTH and several double-number extensions from FORTH-79 are included. The run-time constructs of DO, LOOP, BEGIN, WHILE, AGAIN and UNTIL programming structures and essentially all I/O words are in internal ROM. Even the low-level disk handlers below R/W are in the single-chip computer. This allows about 4 megabytes of mass storage by adding a 1793-type floppy disk controller and associated support circuitry.

Construction of minimal part computers is now feasible. The board which we at New Micros, Inc. have designed around the R65F11, is 100 millimeters on a side. It features an RS 232 port, two parallel TTL ports, an AC power supply, PROM-based address decoding, three JEDEC general purpose RAM/ROM/PROM sockets. These will accept 2016, 2128, 5517, 6116 and 5564 RAMS, 2716, 2732, and 2764 EPROMS, and 2816 EEROMS, etc. totaling 16K bytes. A user can easily develop a program in a high-level language on this computer, transfer the program to EPROM, remove his RS232 terminal and leave the computer system to perform a dedicated task.

This is only a brief introduction to the remarkable R65F11. There are many more features that deserve further attention. A full accounting of its power and utility could fill a book. In fact, the curious reader is encouraged to acquire the *RSC-FORTH Users Manual* from Rockwell International.

Voice of Victor 9000

Timothy Huang
Portland, Oregon

The Victor 9000 is one of the more advanced microcomputers currently available on the market. It has many extra features built in, such as 800 by 400 graphics resolution, 128K RAM, 1.2 megabyte single-sided drives, and sound/voice digitizing/generation. Please see the article entitled "Victor Victorious" by Phil Lemmons in the November 1982 *BYTE* magazine for a detailed description. This article deals with sound generation for which a lot of technical advice was provided by the nice people at Victor Technologies, Inc.

From the very limited system information that is available to the general public, I managed to figure out the following memory-mapped I/O addresses for the CODEC (COder and DECoder) hardware. All addresses are in hexadecimal.

Codec$clk at E8084 (word)
Codec$ctl at E808B (word)
Codec$sda at E8060 (word)
Volume at E802A (byte)
Vol$ctl at E802B (byte)
Vol$clk at E8028 (byte)

The program listed in Screen 205 is a simple trial to use this special feature of the Victor 9000. The program is self explanatory. The only comments that I should add here are:

ES-REG (--- addr) Variable for the ES register.

LC! (n addr ---) Long **C!**. To store <n> into the address specified by <addr + addr_in_ES-REG. This word allows us to store a number across the 64K boundary. Prior to use the **ES-REG** must be initialized to the correct segment high address.

SOUND-INIT (---) Initialize the CODEC chip. This routine must be issued prior to generating any sound. The values used are the same as in the CP/M-86 BIOS.

PITCH (n ---) Generate <n> pitch. It is currently set to accept only 8-bit numbers. **0 PITCH** will shout down the CODEC chip.

Even though this is a very short program, with a little imagination you can generate some very interesting sounds. For example, if you want to hear all the pitches that PITCH can generate, you can write the short words shown in Figure 1.

To experience different sound effects, you may also store different values into the I/O addresses used in the definition of the word **SOUND-INIT**. Or you may want to write some words so that you can write sounds and/or generate a hi-fi human voice. By the way, the machine is capable of digitizing and playing back the human voice, but until I find out more technical information, I am dead-ended. One of these days, I am going to write a program that will allow me to take the voice input and store the digitized data into FORTH virtual memory. Meanwhile, I encourage any of you who are interested in hearing real human voice output (not like mechanical Digitalker or Votrex) to pay a visit to your Victor dealer.

```
: STOP ( --- )  0 PITCH ;          shout down sound
  : ALL ( --- )                    generate all sound
      SOUND-INIT                   initialize chip
  256 0 DO                         all sounds
        I PITCH                    generated
         LOOP  STOP  ;             and shout it down
```

Figure One

```
SCR # 205
  0 \ CODEC SOUND                                          TDH04NOV82
  1 HEX
  2 : SOUND-INIT ( --- )
  3         E000 ES-REG !
  4         00 8060 LC! 5E 8061 LC!
  5         40 8060 LC! 0D 8061 LC!   80 8060 LC! 0F 8061 LC!
  6         C0 8060 LC! 00 8061 LC!   C0 808B LC!
  7         00 8084 LC! 00 8085 LC!   ;
  8
  9 : PITCH       ( n --- )  \ n = 0 to turn off the sound
 10         E000 ES-REG ! 80 8060 LC! 0F 8061 LC!
 11         8084 LC! 00 8085 LC! ;
 12 DECIMAL
 13
 14 ;S
 15
OK
```

End Listing

6502 and 6809 Absolute Branches

George Gaukel
Tacoma, Washington

Listing One

```
]
 0010            F79.001 BRANCH FOR 6502
 0020            USES POINTER FOR BRANCH AND
 0030            ELIMINATES NEED FOR OFFSETS
 0040;
 0050;
 0060L89
 0070            .BY $86 'BRANC' $C8 ; BRANCH
 0080            .SI L75
 0090BRAN
 0100            .SI =+2
 0110            LDA (IP),Y
 0120            PHA
 0130            INY
 0140            LDA (IP),Y
 0150            STA *IP+1
 0160            PLA
 0170            STA *IP
 0180            JMP NEXT+2
 0190;
 0200;
 0210;
 0220            (C) 1982 G.R. GAUKEL
//
```

Listing Two

```
]
 0010            F79.002 BRANCH FOR 6809
 0020            USES POINTER FOR BRANCH AND
 0030            ELIMINATES NEED FOR OFFSETS
 0040*
 0050*
 0060            WORDM 6,BRANC,H
 0070BRAN
 0080            FDB *+2
 0090            LDY ,Y
 0100            LDX ,Y++ * COMPILE NEXT INLINE
 0110            JMP [,X]
 0120            WORDM 7,0BRANC,H
 0130ZBRAN
 0140            FDB *+2
 0150            LDD ,U++
 0160            BNE ZBNO
 0170ZBYES       ********** COMPILE BRAN INLINE
 0180            LDY ,Y
 0190            LDX ,Y++ * COMPILE NEXT IN LINE
 0200            JMP [,X]
 0210ZBNO
 0220            LEAY 2,Y
 0230            LDX ,Y++ * COMPILE NEXT IN LINE
 0240            JMP [,X]
 0250*
 0260*
 0270*
 0280            (C) 1982 G.R. GAUKEL
//
```

(Listings Continued)

In the 6502 and 6809 implementations, the inner interpreter uses 16-bit absolute pointers for address resolution. This would imply a design philosophy of compiling absolute pointers whenever possible. The use of offsets in the branch control structures is an exception to the use of absolute addresses.

Let's take a look at the control structures in the models and see what they actually do. At compiler time they start with a direct pointer left on the stack by **HERE**, and use **BACK** and **THEN**; then they compute and compile an offset. At run time, the branch primitives convert the offset back to the same pointer we started with. If you think this is wasted effort, you're right. If we delete the subtraction at the high level and the addition in the primitives, we can realize some startling advantages in both run-time execution and clarity of assembler code.

First we change the branch primitives to simple vector switches by deleting the addition instructions. Listing One shows the modifications for the 6502. Three micro-instructions have been deleted (saving eight clock cycles). Listing Two shows the modifications for the 6809. The code condenses to one LDY,Y instruction (net savings of fourteen cycles). If **NEXT** is compiled inline, as shown, five additional cycles are saved. If the two-byte branch primitives and the four-byte **NEXT** primitives are also compiled inline in the 6809 loop primitives, there will be even further time savings during loops. What we have done is move the address at the location the IP is pointing to into the IP.

Next we change the high-level control definitions to leave an absolute pointer. Listing Three shows the

modifications required to Screens #73-74 of the model. **BACK** is eliminated from the dictionary, as it becomes a renaming for **COMMA**.

Finally, we need to edit the assembler test. Listing Four shows two examples for the 6502. Wherever we find a branch or loop we just compile the value of the label. The assembler text has now become much more readable and uniform. There are about fifty such labels that will have to be changed. A good text editor will be of value here. A canned version of FORTH can be changed to comply with the above. Just make sure that all offsets are found and changed.

Other implementations using absolute interpreters and offset control structures should realize improvements comparative to the above examples. The above alterations were tested in a recompiled in-house 79-Standard FORTH (6502) and do not alter the specified definitions. They do alter the definitions in the *Release 1 FIG-FORTH Installation Manual* which states that offsets must be used. The FORTH-79 Standard makes no reference to offsets, so they have apparently been moved to the 'Historical Museum' and remain as an implementation option.

There is a trade-off between absolute and relative branches. As the author points out, absolute branches are faster. Relative branches are used in the FIG model in order to make the **LATEST** *definition relocatable. In practice, this feature is almost never used (see Schleisiek,* FORML-1980, *on separated heads). For most users, speed is probably more valuable. As for* **BACK**, *it is probably good to retain it for uniformity with other systems, even if it is only a synonym for* **COMMA**.—*Michael Perry*

Listing Three

```
  SCR# 73 ###6502.FORTH###      DRIVE 1
00 \ CONDITIONAL COMPILIER, PER SHIRA/GAUKEL  GRG21MAY82
01 \ BACK DELETED BECOMES ECHO FOR COMMA
02
03 : BEGIN   ?COMP HERE 1 ; IMMEDIATE
04
05 : THEN    ?COMP 2 ?PAIRS HERE SWAP ! ; IMMEDIATE
06
07 : ENDIF   [COMPILE] THEN ; IMMEDIATE
08
09 : DO      COMPILE (DO) HERE 3 ; IMMEDIATE
10
11 : LOOP    3 ?PAIRS COMPILE (LOOP) , ; IMMEDIATE
12
13 : +LOOP   3 ?PAIRS COMPILE (+LOOP) , ; IMMEDIATE
14
15 : UNTIL   1 ?PAIRS COMPILE OBRANCH , ; IMMEDIATE -->
```

```
  SCR# 74 ###6502.FORTH###      DRIVE 1
00 \ CONDITIONAL COMPILIER, PER SHIRA/GAUKEL  GRG21MAY82
01 : END      [COMPILE] UNTIL ; IMMEDIATE
02
03 : AGAIN    1 ?PAIRS COMPILE BRANCH , ; IMMEDIATE
04
05 : REPEAT   >R >R COMPILE BRANCH , R> R>
06            2 - [COMPILE] ENDIF ; IMMEDIATE
07
08 : IF       COMPILE OBRANCH HERE 0 , 2 ; IMMEDIATE
09
10 : ELSE     2 ?PAIRS COMPILE BRANCH HERE 0 ,
11            SWAP 2 [COMPILE] THEN 2 ; IMMEDIATE
12
13 : WHILE    [COMPILE] IF 2+ ; IMMEDIATE
14 -->
15\ COMPILE BRANCH ADDRESSES NOT THE OFFSET
```

Listing Four

```
]
 0010;F79.007
 0020;EXAMPLES WITH OFFSETS ELIMINATED
 0030;POINTER/LABEL TO ADDR IS USED
 0035;**********************
 0040;
 0050L1657
 0060            .BY $89 '-TRAILIN' $C7 ; -TRAILING
 0070            .SI L1634
 0080DTRAI
 0090            .SI DOCOL
 0100            .SI DUP
 0110            .SI ZERO
 0120            .SI PDO
 0130L1663
 0140            .SI OVER
 0150            .SI OVER
 0160            .SI PLUS
 0170            .SI ONE
 0180            .SI SUB
 0190            .SI CAT
 0200            .SI BL
 0210            .SI SUB
 0220            .SI ZBRAN
 0230            .SI L1676
 0240            .SI LEAVE
 0250            .SI BRAN
 0260            .SI L1678
 0270L1676
 0280            .SI ONE
 0290            .SI SUB
 0300L1678
 0310            .SI PLOOP
 0320            .SI L1663
 0330            .SI SEMIS
 0340;
 0350;**********************
 0360;
 0370L2464
 0380            .BY $82 '+' $AD ; +-
 0390            .SI L2453
 0400PM
 0410            .SI DOCOL
 0420            .SI ZLESS
 0430            .SI ZBRAN
 0440            .SI L2471
 0450            .SI MINUS
 0460L2471
 0470            .SI SEMIS
 0480;
 0490;**********************
//
```

End Listing

FORTH Vendors

(Continued from page 39)

Intersystems Management
Computer Consultancy
Story Hill Rd. RFD3
Dunbarton, NH 03045
603/774-7762

Laxen, Henry H.
1259 Cornell Ave.
Berkeley, CA 94706
415/525-8582

McIntosh, Norman
2908 California Ave., #3
San Francisco, CA 94115
415/563-1246

Metalogic Corp.
4325 Miraleste Dr.
Rancho Palos Verdes, CA 90274
213/519-7013

Petri, Martin B.
Computer Consultants
16005 Sherman Way
Suite 104
Van Nuys, CA 91406
213/908-0160

Redding Co.
P.O. Box 498
Georgetown, CT 06829
203/938-9381

Schleisiek, Klaus
Eppendorfer Landstr. 16
D 2000 Hamburg 20
West Germany
(040)480 8154

Schrenk, Dr. Walter
Postfach 904
7500 Karlstruhe-41
West Germany

Software Engineering
6308 Troost Ave. #210
Kansas City, MO 64131
816/363-1024

Timin, Mitchel
3050 Rue d'Orlean #307
San Diego, CA 92110
619/222-4185

Softweaver
P.O. Box 7200
Santa Cruz, CA 95061
408/425-8700

Technology Management, Inc.
1520 S. Lyon St.
Santa Ana, CA 92705
714/835-9512

Debugging From a Full-Screen Editor

Tom Blakeslee
Woodside, California

One of the problems with most debugging aids is that you must continually look back and forth between a listing of the program and the screen to keep track of where you are. This program adds single-step capability to the Laxen full-screen editor (*Dr. Dobb's Journal*, Sept. 1981) so that the cursor pops from word to word as the program is executed one step at a time. The parameter stack contents are displayed at the bottom of the screen after each program step. Any program output to the screen will be displayed just below the stack display.

To debug a word, you simply put what you want on the stack before calling the editor. You then use the normal cursor control keys to position the cursor on the first word you want to execute. Each time you hit control-W you will enter the next word in sequence and display the results. By watching the stack display, you can see whether it is doing what you want.

If a bug is found while stepping through the program, you can make immediate changes since you are still in the editor with the cursor positioned at the offending word. After making a change you can immediately resume debugging.

The new function is installed by simply loading screen 98 (below) just before screen 80 in the Laxen editor. Screen 80 contains the **CASE:** statement which assigns control key functions. To assign the **STEP** function to control-W, simply replace the **BEEP** just after **INSERT-MODE** in that **CASE:** statement with **STEP**.

Pressing control-W will now cause the word or number at the cursor to be interpreted as though it had been entered at the keyboard and followed by a carriage return. The parameter stack will be dumped at the bottom of the screen, and the cursor will move right to the next word.

Comments and compile-only words can be skipped over by using the normal cursor control keys. A useful enhancement would be another function key for entering two words at a time. This would be useful for entering things such as ' or **WORD**, which cannot be interpreted without a following word. Implementation would be identical to **STEP** except that line 3 would have **R-WORD** twice.

The '79 standard MVP FORTH system was used with the Laxen editor, but the same technique could be applied to any full-screen editor.

```
98 LIST
SCR #98
  0 ( EDITOR DEBUG, STEP)
  1 : STEP ( --) ( LOADS ONE WORD & DISPLAYS STACK)
  2       BUFPOS DUP    ( START ADR OF WORD)
  3       R-WORD     ( MOVE CURSOR RIGHT 1 WORD)
  4       BUFPOS SWAP -  50 MIN  1-   ( LENGTH)
  5       DUP TIB @ +    0 SWAP !   ( NULLS TO END STREAM)
  6       TIB @ SWAP BMOVE    ( MOVE WORD TO TIB)
  7       0 19 CRTXY      ( MOVE CURSOR TO LOWER SCREEN)
  8       BLK @ >R     0 BLK !   ( SAVE , SEL TERM INPUT)
  9       >IN @ >R     0 >IN !
  A       INTERPRET
  B       0 18 CRTXY ( POSITION CURSOR FOR STACK DISPLAY)
  C       40 SPACES   ( CLEAR LINE FOR STACK DISPLAY)
  D       0 17 CRTXY  .S   ( SHOW STACK)
  E       R> >IN !       R> BLK !   ( RESTORE)
  F       0 MOVE-CURSOR   ( RESTORE CURSOR POSITION) ;
OK
```

FORTH Applications Conference

Kim Harris
Palo Alto, California

The 1983 Rochester FORTH Applications Conference was held at the University of Rochester in New York state on June 8 to 11, 1983. About 100 people attended. Robotics applications were emphasized, but many other topics were covered.

Eight invited speakers started the conference by describing robotics projects and related issues. There were about forty oral presentations of papers to be published in the conference proceedings, and there were poster sessions, equipment demonstrations, working groups and a panel discussion. Several presentations were excellent and several were poor.

Compared to previous FORTH conferences (both Rochester and FORML), some good trends were noticeable. There was more academic involvement; seven university professors attended and presented papers, and twenty-one university students did likewise. There were even two high school students showing "science fair" projects using the Jupiter-Ace computer.

There were more applications discussed than at previous FORTH conferences. A robotics project at the University of Massachusetts is integrating tactile feedback into robotics. One application is automated sheep shearing. A photograph showed a sheep held in one robot arm about two feet off the floor and shears held in another robot arm! Robotics in motion picture special effects was described. It included a sophisticated device which holds a camera and a miniature model, has 24 degrees of freedom, operates in real-time (*i.e.* 24 Hertz), and is programmed in FORTH on an LSI-11. The hardware demonstration session included two robot arms and Androbot's toy robot named TOPO. The successes of today's robotics was put into perspective by Don Davenport of Standard Oil of Ohio by comparing state of the art robots to the mosquito:

> "A mosquito has a tiny brain of only several thousand neurons, yet it is capable of flying, landing, drilling, feeding, avoiding attack, tactile and visual sensing, mating, and reproduction."

More large companies sent representatives and delivered talks on their projects than ever before. Even the National Bureau of Standards described a robotics project under development there. The project's scope is the complete automation of a manufacturing factory.

Charles Moore demonstrated a computer design automation program he has produced to assist him in the design of FORTH printed circuit boards and integrated circuits. He also talked on his view of the successes of FORTH and future of FORTH. He stated that FORTH is successful because it is the only language that has directly addressed the problem of communciations between a human and computer.

A panel discussion was held to discuss the impact of FORTH on large company management. Several people who have introduced FORTH into large companies related their experiences and gave advice.

Some selected highlights of the conference demonstrate significant developments in FORTH's position in the computer industry:

General Electric described an "expert system" implemented in FORTH. Such systems are very advanced, useful, new applications of computers which try to capture the wisdom of a human expert in a program. The expert system described tried to diagnose faults in diesel locomotives made and repaired by GE. The goal was to reduce repair time by semi-trained technicians to that of experts with over twenty years of experience. A prototype system was implemented in LISP on a DEC PDP-10 mainframe computer, but the final version needed to run on a portable minicomputer. The prototype was reimplemented in FORTH on a LSI-11/23, then development continued on both versions. In a couple of months, the FORTH version exceeded the capabilities of the LISP version. The project was a spectacular success and shows FORTH's applicability to an area previously "owned" by LISP and large machines.

A software tool was described called a "Functional Usage Analyzer." It performs analysis on FORTH programs using numerical correlation and pattern recognition to determine some aspects of their quality. The results are exciting and may lead to the automatic determination of software quality superior to current methods of analysis.

A university project on Functional Programming (FP) languages was described which extended FORTH along the lines of LISP and some new languages. FP languages promise to be powerful in advanced applications (like expert systems) and on new distributed computer architectures. This project is in the Computer Science department; that is a first for FORTH. It would be beneficial if cooperation could be developed between the FORTH community and the LISP community.

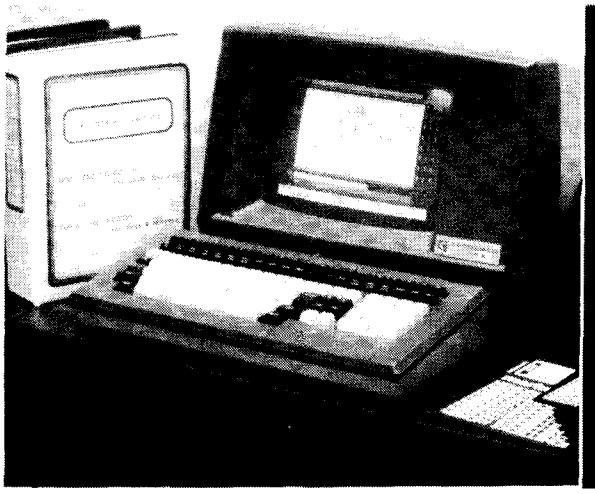
Dysan's floating point implementation in FORTH created considerable interest. It provides a transportable version of the IEEE standard floating-point package. There was a general consensus among conference attendees that floating-point operations need to be available to professional FORTH systems.

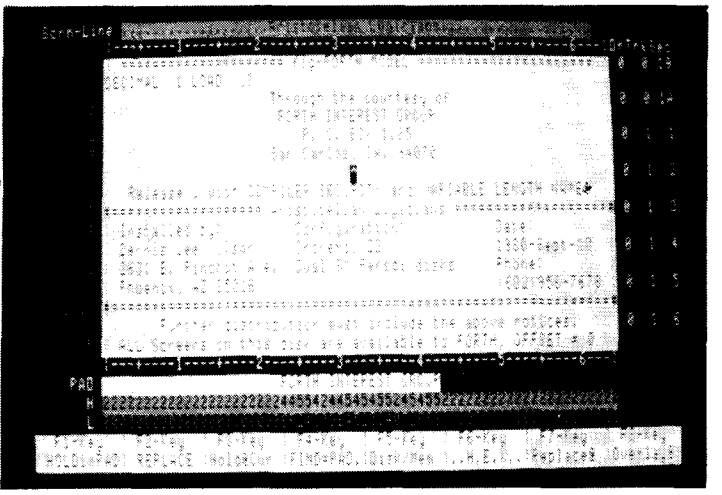
(Continued from page 14)

panel and also downloads the speed value into the fuel management system. The CPU also generates self-test functions and drives display lights that show the confidence level of the depth data and also another confidence level which indicates the relative goodness of the data received over the RF link. This is used both as a safety factor and for maintenance purposes. The use of a FORTH-based microprocessor drastically reduced the time required to complete this section of the project.

One of the primary goals of the program was to develop systems which were inherently reliable and relatively inexpensive to procure. To attain these goals, we adhered to two general principles—we used circuits which are applicable to other projects and we based the system on a bus structure. We chose the RM bus, primarily because of the utility of the Eurocard format. Since the intended use of the device was on shipboard, which could be considered a severe environment, the positive connector lock provided with the Eurocard contributes substantially to system survivability. We procured the system PCB's from stock items in the New Micros, Inc. line so the acquisition and design costs for this dedicated application were held to a reasonable level. The retail price of the units actually is substantially less than that charged for existing units, with far lower costs for maintenance.

The modular structure of the FORTH program used in this system was instrumental in our ability to quickly generate a working program and to modify the program as necessary to cope with real-world anomalies. As an indication of the extreme power of FORTH in developing dedicated systems, the time lag from the start of programming to the first field-functional unit was eight days. This translates into large savings in system overhead charges for R & D and materially affects the marketability of the product line. The ease with which this program could be modified to generate an entirely new line of products such as a storage tank level-sensing device is obvious. The conclusion is that FORTH-based dedicated systems can have a powerful effect on the future of industrial system design.





8080/Z80 FIG-FORTH for CP/M & CDOS systems
FULL-SCREEN EDITOR for DISK & MEMORY

$50 saves you keying the FIG FORTH model and many published FIG FORTH screens onto diskette and debugging them. You receive TWO diskettes (see below for formats available). The first disk is readable by Digital Research CP/M or Cromemco CDOS and contains 8080 source I keyed from the published listings of the FORTH INTEREST GROUP (FIG) plus a translated, enhanced version in ZILOG Z80 mnemonics. This disk also contains executable FORTH.COM files for Z80 & 8080 processors and a special one for Cromemco 3102 terminals.

The 2nd disk contains FORTH readable screens including an extensive FULL-SCREEN EDITOR FOR DISK & MEMORY. This editor is a powerful FORTH software development tool featuring detailed terminal profile descriptions with full cursor function, full and partial LINE-HOLD LINE-REPLACE and LINE-OVERLAY functions plus line insert/delete, character insert/delete, HEX character display/update and drive-track-sector display. The EDITOR may also be used to VIEW AND MODIFY MEMORY (a feature not available on any other full screen editor we know of.) This disk also has formatted memory and I/O port dump words and many items published in FORTH DIMENSIONS, including a FORTH TRACE utility, a model data base handler, an 8080 ASSEMBLER and a recursive decompiler.

The disks are packaged in a ring binder along with a complete listing of the FULL-SCREEN EDITOR and a copy of the FIG-FORTH INSTALLATION MANUAL (the language model of FIG-FORTH, a complete glossary, memory map, installation instructions and the FIG line editor listing and instructions).

This entire work is placed in the public domain in the manner and spirit of the work upon which it is based. Copies may be distributed when proper notices are included.

	USA	Foreign AIR
☐ **FIG-FORTH & Full Screen EDITOR package**		
Minimum system requirements:		
80x24 video screen w/ cursor addressability		
8080 or Z80 or compatible cpu		
CP/M or compatible operating system w/ 32K or more user RAM		
Select disk format below, (soft sectored only)	$50	$65
☐ 8" SSSD for CP/M (Single Side, Single Density)		
Cromemco CDOS formats, Single Side, S/D Density		
☐ 8" SSSD ☐ 8" SSDD ☐ 5¼" SSSD ☐ 5¼" SSDD		
Cromemco CDOS formats, Double Side, S/D Density		
☐ 8" DSSD ☐ 8" DSDD ☐ 5¼" DSSD ☐ 5¼" DSDD		
Other formats are being considered, tell us your needs.		
☐ Printed Z80 Assembly listing w/ xref (Zilog mnemonics)	$15	$18
☐ Printed 8080 Assembly listing	$15	$18
TOTAL $_____		_____

Price includes postage. No purchase orders without check. Arizona residents add sales tax. Make check or money order in US Funds on US bank, payable to:

Dennis Wilson c/o
Aristotelian Logicians
2631 East Pinchot Avenue
Phoenix, AZ 85016
(602) 956-7678

FORTH for Z-80® , 8086, 68000, and IBM® PC

FORTH Application Development Systems include interpreter/compiler with virtual memory management and multi-tasking, assembler, full screen editor, decompiler, utilities, and 130 + page manual. Standard random access files used for screen storage, extensions provided for access to all operating system functions.

Z-80 FORTH for CP/M® 2.2 or MP/M II ..... $ 50.00
8080 FORTH for CP/M 2.2 or MP/M II ..... $ 50.00
8086 FORTH for CP/M-86 or MS-DOS $100.00
PC/FORTH™ for PC-DOS, CP/M-86, or CCPM $100.00
68000 FORTH for CP/M-68K $250.00

83 - Standard version of all application development systems available soon. All registered users will be entitled to software update at nominal cost.

FORTH + Systems are 32 bit implementations that allow creation of programs as large as 1 megabyte. The entire memory address space of the 68000 or 8086/88 is supported directly for programs and data.

PC/FORTH + for PC-DOS or CP/M-86 ..... $250.00
8086 FORTH + for CP/M-86 ..... $250.00
68000 FORTH + for CP/M-68K $400.00

Extension Packages for FORTH systems

Software floating point (Z-80, 8086, PC only) $100.00
Intel 8087 support (8086, PC only) $100.00
AMD 9511 support (8086, Z-80 only) $100.00
Color graphics with animation support (PC only) $100.00
Symbolic interactive debugger (PC only) $100.00
Cross reference utility ..... $ 25.00
PC/GEN™ (custom character sets, PC only) ..... $ 50.00
PC/TERM communications program for PC and Smartmodem ..... $ 60.00
Hierarchical file manager ..... $ 50.00
B-tree index manager $125.00
B-tree index and file manager $200.00

QTF + Screen editor and text formatter by Leo Brodie, for IBM PC with IBM or Epson printer $100.00

Nautilus Cross Compiler allows you to expand or modify the FORTH nucleus, recompile on a host computer for a different target computer, generate headerless and ROMable code. Supports forward referencing. Produces executable image in RAM or disk file. No license fee for applications. Prerequisite: Application Development System for host computer.

Hosts: Z-80 (CP/M 2.2 or MP/M II), 8086/88 (CP/M-86 or MS-DOS), IBM PC (PC-DOS or CP/M-86), 68000 (CP/M-68K)
Targets: 8080, Z-80, 8086/88, 6502, LSI-11, 68000, 1802, Z-8

Cross-Compiler for one host and one target $300.00
Each additional target $100.00

AUGUSTA™ ADA subset compiler from Computer Linguistics, for Z-80 computers under CP/M 2.2 ..... $ 90.00
LEARNING FORTH computer-assisted tutorial by Laxen and Harris for CP/M, includes Brodie's "Starting FORTH" (8" format only) ..... $ 95.00
Z-80 Machine Tests Memory, disk, printer, and console tests with all source code in standard Zilog mnemonics ..... $ 50.00

8080 and Z-80 application development systems require 48 kbytes RAM and 1 disk drive, 8086 and 68000 require 64 kbytes. Prices include shipping by UPS or first class mail within USA and Canada. California residents add appropriate sales tax. Purchase orders accepted at our discretion. Master Charge and Visa accepted.

Disk formats available: Standard CP/M 8" SSSD, Northstar 5¼" QD, Micropolis 5¼" QD, Sage 5¼" DD, Apple 5¼", Victor 9000 5¼", Kaypro 5¼", Osborne 5¼" DD, Micromate 5¼", IBM PC 5¼", Standard MS-DOS 5¼" SSDD. Most other formats can be special ordered.

Laboratory Microsystems, Inc.
4147 Beethoven Street
Los Angeles, CA 90066
(213) 306-7412

FIG Chapter News

John D. Hall
Oakland, California

FIG is formalizing the connection between FIG chapters around the world in order to strengthen FORTH by strengthening the FORTH community. We were once one person, then many, and are now a community of thirty-eight chapters throughout the world. We soon will be stronger by an additional fifty-one chapters that are now forming. And yet, this is just the tip of what is possible. We are a community of 4,000 members likely to grow to 6,000 this year. Many of the chapters reflect areas where there are concentrations of FIG members, but many areas are not represented. Take a look at the list of FIG chapters. Is there a chapter in your area? If not, why not form one! If yes, are you supporting it? There are very few places in the U.S., Canada, Australia, England, Japan, New Zealand, Sweden and West Germany that do not have enough FIG members to form a chapter.

The purpose of the FORTH Interest Group is to disseminate information about FORTH, encourage education in FORTH and promote interest in FORTH. FORTH! You and I use FORTH and the better we can promote it, teach it and encourage its evolution, the better it will help us. Your effort can support a FIG chapter.

The space on these pages is limited, so in each issue I will list a few of the new chapters that are forming (see box). If you live in any of these areas, offer your support to the chapter. It takes a lot of voluntary effort to get these chapters started, but once they are functioning and reach a sufficient number to keep going, they are easier (but don't let down your guard, they will always need your support).

Chapter News? Well, did you know that the Greater Oregon FORTH Interest Group, GOFIG, has a newsletter? It's called the GOFIG Gazette and the way it is growing it may someday rival *FORTH Dimensions*. In May, it had tutorials on FORTH-79 **DOES>**, and FORTH-79 Vocabularies, as well as code on terminal I/O and upper- and lower-case conversion. You say you haven't seen it? Talk to your local chapter. All newsletters, meeting handouts and chapter meeting summaries sent to me are redistributed to all chapters each month.

At the May Orange County FIG meeting, Zane Thomas demonstrated his 68000 Alpha Micro FIG-FORTH

(both 16- and 32-bit). Wil Baden proposed a **THEN-IF** concept to replace the **CASE** statement. This may also be presented at FORML. Ed Wedemeyer presented a paper outlining how he uses FORTH to map PROMs.

FIG-Australia always has something interesting for us foreigners. Lance Collins says that besides conducting regular monthly meetings with members present, his chapter has several correspondence members who the chapter has to keep in contact with. No, Lance, we don't think it is all Out-Back!

As a last calm note: Write me! Write articles! Support your local chapter! Do something to get the rest of us stirred up! As a learned sage once said, "Ask not what FORTH can do for you, but ask what you can do for FORTH."

John Hall is the Chapter Coordinator for the FORTH Interest Group and is a consulting programmer in Oakland, California.

Chapters in Formation

Contact:

Charles Shattuck
206 Irene Ave.
Roseville, CA 95678

John Forsberg
17740 S.W. 109th Place
Perrine, FL 33157

Michael Ham
3110 Alpine Court
Iowa City, IA 52240

Kenneth R. Tenchard
6145 N. Sheridan Rd.
Chicago, IL 60660

Arne Flones
425 W. 9th
Wichita, KS 67203

S.A. Orrell
EG&G Kirkland Operations
P.O. Box 4339, Sta A
Albuquerque, NM 87196

Bsail Barnes
10348 146th Street
Edmonton, Alberta T5N 3A2

Jorge Phillips
Briceno
Carrera 8 #85-24, apt 202
Bogota, D.E., Colombia

Andy Biggs
41 Lode Way
Haddenham, Ely,
Cambridge CB6 3UL
England

M.J. Kerwick
17 Chapel Street
Carrick on Suir, County Tip
Ireland

Jerry Smith
G.W. Smith Assoc.
28 Center Street
Newark, DE 19711

Ron Skelton
1220 Winding Branch Circle
Atlanta, GA 30338

S. Matthew Prastein
Argonne National Lab
1EP362-E3038B
Argonne, IL 60439

Joe Kusner
515 East Liberty Street
Wauconda, IL 60084
312/526-2086

Emre Deniz Tufekcioglu
7823 Zimple - B
New Orleans, LA 70118
504/733-8629

Lee Hustead
2909 Toll Gate Road
Norristown, PA 19403
215/539-7989

Chris Huntley
1551 Howard Ave.
Burnaby, B.C. V5B 3S2

Niels Oesten
Brostykkevej 189
Hui Dovre, DK 2650
Denmark

Klaus Schleisiek
Eppendorfer Landstr. 16
D 2000 Hamburg 20
West Germany

Keith Elkin
Dianavagen 30
11543 Stockholm
Sweden

Fig Chapters

U.S.

• ARIZONA

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

• CALIFORNIA

Los Angeles Chapter
Monthly, 4th Sat., 11 a.m.
Allstate Savings
8800 So. Sepulveda Boulevard
Los Angeles
Call Phillip Wasson
213/649-1428

Northern California Chapter
Monthly, 4th Sat., 1 p.m.
FORML Workshop at 10 a.m.
Palo Alto area.
Contact FIG Hotline
415/962-8653

Orange County Chapter
Monthly, 4th Wed., 7 p.m.
Fullerton Savings
Talbert & Brookhurst
Fountain Valley

Monthly, 1st Wed., 7 p.m.
Mercury Savings
Beach Blvd. & Eddington
Huntington Beach
Call Noshir Jesung
714/842-3032

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

• COLORADO

Denver Chapter
Monthly, 1st Mon., 7 p.m.

Call Steven Sarns
303/477-5955

• MASSACHUSETTS

Boston Chapter
Monthly, 1st Wed., 5 p.m.
Mitre Corp. Cafeteria
Bedford, MA
Call Bob Demrow
617/688-5661 after 7 p.m.

• MICHIGAN

Detroit Chapter
Call Dean Vieau
313/493-5105

• MINNESOTA

MNFIG Chapter
Monthly, 1st Mon.
1156 Lincoln Avenue
St. Paul, MN
Call Fred Olson
612/588-9532

• MISSOURI

St. Louis Chapter
Monthly, 3rd Tue., 7 p.m.
Thornhill branch of
St. Louis County Library
Call David Douda
314/867-4482

• NEVADA

Las Vegas Chapter
Suite 900
101 Convention Center Drive
Las Vegas, NV
Call Gerald Hasty
702/453-3544

• NEW JERSEY

New Jersey Chapter
Call George Lyons
201/451-2905 eves.

• NEW YORK

New York Chapter
Monthly, 2nd Wed., 8 p.m.
FIG of NYC
P.O. Box 452
East Elmhurst, NY 11369
Call Tom Jung
212/432-1414 Ex. 157

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

Syracuse Chapter
Call C. Richard Corner
315/456-7436

• OKLAHOMA

Tulsa Chapter
Monthly, 3rd Tues., 7:30 p.m.
The Computer Store
4343 South Peoria
Tulsa, OK
Call Bob Giles
918/599-9304 or
Art Gorski
918/743-0113

• OHIO

Athens Chapter
Call Isreal Urieli
614/594-3731

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

• OREGON

Portland Chapter
Monthly, 2nd Sat., 1 p.m.
Computer & Things
3460 SW 185th, Aloha
Call Timothy Huang
503/289-9135

• TEXAS

Dallas/Ft. Worth Chapter
Monthly, 4th Thurs., 7 p.m.
Software Automation, Inc.
14333 Porton, Dallas
Call Marvin Elder
214/392-2802 or
Bill Drissel
214/264-9680

San Antonio Chapter
T.L. Schneider
8546 Broadway, Suite 207
San Antonio, TX 78217

• VERMONT

Vermont Fig Chapter
Monthly, 4th Thurs., 7:30 p.m.
The Isley Library,
3rd Floor Meeting Room
Middleburynes, VT 05491
Call Hal Clark
802/877-2911 days
802/452-4442 eves

• VIRGINIA

Potomac Chapter
Monthly, 1st Tues., 7 p.m.
Lee Center
Lee Highway at Lexington St.
Arlington, VA
Call Joel Shprentz
703/437-9218 eves.

FOREIGN

• AUSTRALIA

Australia Fig Chapter
Contact: Ritchie Laird
25 Gibsons Road
Sale, Victoria 3850
051/44-3445

FIG Australia Chapter
Contact: Lance Collins
65 Martin Road
Glen Iris, Victoria 3146
03/29-2600

Sydney Chapter
Monthly, 2nd Fri., 7 p.m.
Morven Brown Bldg., Rm LG16
Univ. of New South Wales
Sydney
Contact: Peter Tregeagle
10 Binda Rd., Yowie Bay
02/524-7490

• BELGIUM

Belgium Chapter
Contact: Luk Van Loock
Lariksdreff 20
B2120 Schoten
03/658-6343

• CANADA

Nova Scotia Chapter
Contact: Howard Harawitz
P.O. Box 688
Wolfville, Nova Scotia BOP 1X0
902/542-7812

Southern Ontario Chapter
Monthly, 1st Sat., 2 p.m.
General Sciences Bldg, Rm 312
McMaster University
Contact: Dr. N. Solntseff
Unit for Computer Science
McMaster University
Hamilton, Ontario L8S 4K1
416/525-9140 ext. 2065

Quebec Chapter
Call Gilles Paillard
418/871-1960 or
418/643-2561

• ITALY

FIG Italia
Contact: Marco Tausel
Via Gerolamo Forni 48
20161 Milano
01/645-8688

• NETHERLANDS

HCC-FORTH Interest Group Chapter
F.J. Meijer
Digicos
Aart V.D. Neerweg 31
Ouderkerk A.D.
Amstel, The Netherlands

• SOUTH AFRICA

Contact: Edward Murray
Forthwith Computers
P.O. Box 27175
Sunnyside, Pretoria 0132

• SWITZERLAND

Contact: Max Hugelshofer
ERNI & Co. elektro-Industrie
Stationsstrasse
8306 Bruttisellen
01/833-3333

• WEST GERMANY

West German Chapter
Klaus Schleisiek
FIG Deutschland
Postfach 202264
D 2000 Hamburg 20
West Germany

SPECIAL GROUPS

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

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

FIGGRAPH
Call Howard Pearlmutter
408/425-8700

MMSFORTH Users Groups
Monthly, 3rd Wed., 7 p.m.
Cochituate, MA
Dick Miller
617/653-6136
(25 groups world-wide)

FORTH System Vendors

(by Category)

(Codes refer to alphabetical listing e.g., A1 signifies AB Computers, etc.)

Processors

1802	C1, C2, F3, F6, L3
6502 (AIM, KIM, SYM)	R1, R2, S1
6800	C2, F3, F5, K1, L3, M6, T1
6801	P4
6809	C2, F3, L3, M6, S11, T1
68000	C2, C4, D1, E1, K1
68008	P4
8080/85	A5, C1, C2, F4, I5, L1, L3, M3, M6, R1, T3
Z80/89	A3, A5, C2, F4, I3, L1, M2, M3, M5, N1, T3
Z80000	I3
8086/88	C2, F2, F3, L1, L3, M6
9900	E2, L3

Operating Systems

CP/M	A3, A5, C2, F3, I3, L3, M1, M2, M6, T3
CP/M86	C2

Computers

Alpha Micro	P3, S3
Apple	A4, F4, I2, I4, J1, L4, M2, M6, M8, 02, 03
Atari	M6, P2, Q1, V1
Compaq	M5
Cromemco	A5, M2, M6
DEC PDP/LSI-11	C2, F3, L2, S3
Heath-89	M2, M6, M7
Hewlett-Packard 85	
Hewlett-Packard 9826/36	C4
IBM PC	A8, C2, F3, L1, M5, M6, Q2, S9, W2
IBM Other	L3, W1
Kaypro II/Xerox 820	M2
Micropolis	A2, M2, S2
North Star	I5, M2, P1, S7
Nova	C5
Ohio Scientific	A6, B1, C3, O1, S6, T2
Osborne	M2
Pet SWTPC	A1, A6, B1, C3, O1, S6, T2, T5
Poly Morphic Systems	A7
TRS-80 I, II, and/or III	I5, M2, M5, M6, S4, S5, S10
TRS-80 Color	A3, A8, F5, M4, S11, T1
Vector Graphics	M2

Other Products/Services

Applications	P4
Boards, Machine	F3, M3, P4, R2
Consultation	C2, C4, N1, P4, T3, W1
Cross Compilers	C2, F3, I3, M6, N1, P4
Products, Various	A5, C2, F3, I5, S8, W2
Training	C2, F3, I3, P4, W1

FORTH Vendors *(Alphabetical)*

The following vendors offer FORTH systems, applications, or consultation. FIG makes no judgement on any product, and takes no responsibility for the accuracy of this list. We encourage readers to keep us informed on availability of the products and services listed. Vendors may send additions and corrections to the Editor, and must include a copy of sales literature or advertising.

FORTH Systems

A

1. AB Computers
   252 Bethlehem Pike
   Colmar, PA 18915
   215/822-7727
2. Acropolis
   17453 Via Valencia
   San Lorenzo, CA 94580
   415/276-6050
4. Applied Analytics Inc.
   8910 Brookridge Dr., #300
   Upper Marlboro, MD 20870
5. Aristotelian Logicians
   2631 E. Pinchot Ave.
   Phoenix, AZ 85016
7. Abstract Systems, etc.
   RFD Lower Prospect Hill
   Chester, MA 01011
8. Armadillo Int'l Software
   P.O. Box 7661
   Austin, TX 78712
   512/459-7325

B

1. Blue Sky Products
   729 E. Willow
   Signal Hill, CA 90806

C

1. Chrapkiewicz, Thomas
   16175 Stricker
   East Detroit, MI 48021
2. CMOSOFT
   P.O. Box 44037
   Sylmar, CA 91342
3. COMSOL, Ltd.
   Treway House
   Hanworth Lane
   Chertsey, Surrey
   England KT16 9LA
4. Consumer Computers
   8907 La Mesa Blvd.
   La Mesa, CA 92041
   714/698-8088
5. Creative Solutions, Inc.
   4801 Randolph Rd.
   Rockville, MD 20852
6. Capstone Computing, Inc.
   5640 Southwyck Blvd., #2E
   Toledo, OH 43614
   419/866-5503

E

1. Emperical Research Group
   P.O. Box 1176
   Milton, WA 98354
   206/631-4855
2. Engineering Logic
   1252 13th Ave.
   Sacramento, CA 95822

F

1. Fantasia Systems, Inc.
   1059 The Alameda
   Belmont, CA 94002
   415/593-5700
3. FORTH, Inc.
   2309 Pacific Coast Highway
   Hermosa Beach, CA 90254
   213/372-8493
4. FORTHWare
   639 Crossridge Terrace
   Orinda, CA 94563
5. Frank Hogg Laboratory
   130 Midtown Plaza
   Syracuse, NY 13210
   315/474-7856
6. FSS
   P.O. Box 8403
   Austin, TX 78712
   512/477-2207

H

1. HAWG WILD Software
   P.O. Box 7668
   Little Rock, AR 72217

I

1. IDPC Company
   P.O. Box 11594
   Philadelphia, PA 19116
   215/676-3235

2. IUS (Cap'n Software)
281 Arlington Ave.
Berkeley, CA 94704
415/525-9452

3. Inner Access
517K Marine View
Belmont, CA 94002
415/591-8295

4. Insoft
10175 S.W. Barbur Blvd.
Suite #202B
Portland, OR 97219
503/244-4181

5. Interactive Computer Systems, Inc.
6403 Di Marco Rd.
Tampa, FL 33614

J

1. JPS Microsystems, Inc.
361 Steelcase Rd., W.
Markham, Ontario
Canada L3R 3V8
416/475-2383

K

1. Kukulies, Christoph
Ing. Buro Datentec
Heinrichsallee 35
Aachen, 5100
West Germany

L

1. Laboratory Microsystems
4147 Beethoven St.
Los Angeles, CA 90066
213/306-7412

2. Laboratory Software Systems, Inc.
3634 Mandeville Canyon
Los Angeles, CA 90049
213/472-6995

3. Lynx
3301 Ocean Park, #301
Santa Monica, CA 90405
213/450-2466

4. Lyons, George
280 Henderson St.
Jersey City, NJ 07302
201/451-2905

M

1. M & B Design
820 Sweetbay Dr.
Sunnyvale, CA 94086

2. MicroMotion
12077 Wilshire Blvd., #506
Los Angeles, CA 90025
213/821-4340

3. Microsystems, Inc.
2500 E. Foothill Blvd., #102
Pasadena, CA 91107
213/577-1477

4. Micro Works, The
P.O. Box 1110
Del Mar, CA 92014
714/942-2400

5. Miller Microcomputer
61 Lake Shore Rd.
Natick, MA 01760
617/653-6136

6. Mountain View Press
P.O. Box 4656
Mountain View, CA 94040
415/961-4103

7. MCA
8 Newfield Ln.
Newtown, CT 06470

8. Metacrafts Ltd.
Beech Trees, 144 Crewe Rd.
Shavington, Crewe CW1
5AJ
England

N

1. Nautilus Systems
P.O. Box 1098
Santa Cruz, CA 95061
408/475-7461

O

1. OSI Software & Hardware
3336 Avondale Court
Windsor, Ontario
Canada N9E 1X6
519/969-2500

2. Offete Enterprises
1306 S "B" St.
San Mateo, CA 94402

3. On-Going Ideas
RD #1, Box 810
Starksboro, VT 05487
802/453-4442

P

1. Perkel Software Systems
1636 N. Sherman
Springfield, MO 65803

2. Pink Noise Studios
P.O. Box 785
Crockett, CA 94525
415/787-1534

3. Professional Mgmt. Services
724 Arastradero Rd., #109
Palo Alto, CA 94306
408/252-2218

4. Peopleware Systems Inc.
5190 West 76th St.
Minneapolis, MN 55435
612/831-0872

Q

1. Quality Software
6660 Reseda Blvd., #105
Reseda, CA 91335

2. Quest Research, Inc.
P.O. Box 2553
Huntsville, AL 35804
800/558-8088

R

2. Rockwell International
Microelectronics Devices
P.O. Box 3669
Anaheim, CA 92803
714/632-2862

S

1. Saturn Software, Ltd.
P.O. Box 397
New Westminister, BC
V3L 4Y7 Canada

2. Shaw Labs, Ltd.
P.O. Box 3471
Hayward, CA 94540
415/276-6050

3. Sierra Computer Co.
617 Mark NE
Albuquerque, NM 87123

4. Sirius Systems
7528 Oak Ridge Highway
Knoxville, TN 37921
615/693-6583

5. Software Federation
44 University Drive
Arlington Hts., IL 60004
312/259-1355

6. Software Works, The
1032 Elwell Ct., #210
Palo Alto, CA 94303
415/960-1800

7. Supersoft Associates
P.O. Box 1628
Champaign, IL 61820
217/359-2112

8. Satellite Software Systems
288 West Center
Orem, UT 84057
801/224-8554

9. Spectrum Data Systems
5667 Phelps Luck Dr.
Columbia, MD 21045
301/992-5635

10. Stearns, Hoyt Electronics
4131 E. Cannon Dr.
Phoenix, AZ 85028
602/996-1717

T

1. Talbot Microsystems
1927 Curtis Ave.
Redondo Beach, CA 90278

2. Technical Products Co.
P.O. Box 12983
Gainsville, FL 32604
904/372-8439

3. Timin Engineering Co.
C/o Martian Technologies
8348 Center Dr. Suite F
La Mesa, CA 92041
619/464-2924

4. Transportable Software
P.O. Box 1049
Hightstown, NJ 08520
609/448-4175

V

1. Valpar International
3801 E. 34th St.
Tucson, AZ 85713
800/528-7070

W

1. Ward Systems Group
8013 Meadowview Dr.
Frederick, MD 21701

2. Worldwide Software
2555 Buena Vista Ave.
Berkeley, CA 94708
415/644-2850

Z

1. Zimmer, Tom
292 Falcato Dr.
Milpitas, CA 95035

Boards & Machines Only
See System Vendor Chart for others

Controlex Corp.
16005 Sherman Way
Van Nuys, CA 91406
213/780-8877

Datricon
7911 NE 33rd Dr., #200
Portland, OR 97211
503/284-8277

Golden River Corp.
7315 Reddfield Ct.
Falls Church, CA 22043

Triangle Digital Services Ltd.
23 Campus Road
London E17 5PG
England

Application Packages Only
See System Vendor Chart for others

Curry Associates
P.O. Box 11324
Palo Alto, CA 94306
415/322-1463

InnoSys
2150 Shattuck Ave.
Berkeley, CA 94704
415/843-8114

Consultation & Training Only
See System Vendor Chart for others

Bartholomew, Alan
2210 Wilshire Blvd. #289
Santa Monica, CA 90403
213/394-0796

Boulton, Dave
581 Oakridge Dr.
Redwood City, CA 94062

Brodie, Leo
9720 Baden Ave.
Chatsworth, CA 91311
213/998-8302

Eastgate Systems Inc.
P.O. Box 1307
Cambridge, MA 02238

Girton, George
1753 Franklin
Santa Monica, CA 90404
213/829-1074

Go FORTH
504 Lakemead Way
Redwood City, CA 94062
415/366-6124

Harris, Kim R.
Forthright Enterprises
P.O. Box 50911
Palo Alto, CA 94303
415/858-0933

(Continued on page 29)

FORTH INTEREST GROUP

MAIL ORDER

	USA	FOREIGN AIR
☐ Membership in FORTH Interest Group and Volume V of FORTH DIMENSIONS	$15	$27
☐ Back Volumes of FORTH DIMENSIONS. **Price per each.**	$15	$18
☐ I ☐ II ☐ III ☐ IV		
☐ fig-FORTH Installation Manual, containing the language model of fig-FORTH, a complete glossary, memory map and installation instructions	$15	$18
☐ Assembly Language Source Listings of fig-FORTH for specific CPU's and machines. The above manual is required for installation. Check appropriate box(es). **Price per each.**	$15	$18
☐ 1802 ☐ 6502 ☐ 6800 ☐ 6809 ☐ VAX ☐ Z80		
☐ 8080 ☐ 8086/8088 ☐ 9900 ☐ APPLE II ☐ ECLIPSE		
☐ PACE ☐ NOVA ☐ PDP-11 ☐ 68000 ☐ ALPHA MICRO		
☐ "Starting FORTH" by Brodie. BEST book on FORTH. (Paperback)	$18	$22
☐ "Starting FORTH" by Brodie. (Hard Cover)	$22	$27
☐ PROCEEDINGS 1980 FORML (FORTH Modification Lab) Conference	$25	$35
☐ PROCEEDINGS 1981 FORML Conference, Both Volumes	$40	$55
☐ Volume I, Language Structure	$25	$35
☐ Volume II, Systems and Applications	$25	$35
☐ PROCEEDINGS 1982 FORML Conference	$25	$35
☐ PROCEEDINGS 1981 FORTH Univ. of Rochester Conference	$25	$35
☐ PROCEEDINGS 1982 FORTH Univ. of Rochester Conference	$25	$35
☐ FORTH-79 Standard, a publication of the FORTH Standards Team	$15	$18
☐ Kitt Peak Primer, by Stevens. An in-depth self-study primer.	$25	$35
☐ BYTE Magazine Reprints of FORTH articles, 8/80 to 4/81	$ 5	$10
☐ FIG T-shirts: ☐ Small ☐ Medium ☐ Large ☐ X-Large	$10	$12
☐ Poster, August 1980 BYTE cover, 16" x 22"	$ 3	$ 5
☐ FORTH Programmer's Reference Card. If ordered separately, send a stamped, addressed envelope.		FREE
☐ Dr. Dobb's Journal, Two FORTH Issues, 9/81 & 9/82	$ 7	$10
TOTAL	$_____	

NAME_____ MAIL STOP/APT_____

ORGANIZATION_____ PHONE ( )_____

ADDRESS_____

CITY_____ STATE_____ ZIP_____ COUNTRY_____

VISA #_____ MASTERCARD #_____

Expiration Date_____ (Minimum of $15.00 on charge cards)

Make check or money order in US Funds on US bank, payable to: **FIG.** All prices include postage. **No purchase orders without check.** California residents add sales tax. 1/83

ORDER PHONE NUMBER: (415) 962-8653

FORTH INTEREST GROUP • PO BOX 1105 • SAN CARLOS, CA 94070

FORTH INTEREST GROUP
P.O. Box 1105
San Carlos, CA 94070

BULK RATE
U.S. POSTAGE
PAID
Permit No. 26
[illegible], CA

ROBERT SMITH
2300 ST. FRANCIS DR
PALO ALTO, CA 94303

Address Correction Requested