FORTH Dimensions

Volume 6, Number 5

January/February 1985
$2.50

Simple Modem I/O

Augmented TRACE

Quicksort and Swords

Why Forth Isn't Slow

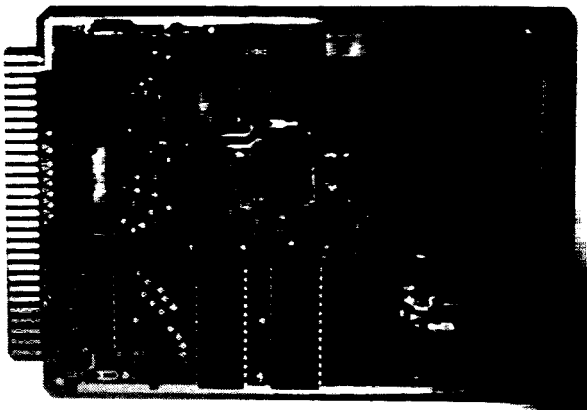
FORML China Tour

The **ForthCard**

STD BUS INTERFACE

DOWNLOAD SOURCE FROM YOUR PC

OPTIONAL 5v REGULATOR ON CARD



PARALLEL I/O

EPROM/EEPROM PROGRAMMER

232 I/O

ROCK

24 KBYTES

64 KBYTES

ADDRESS D

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

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

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

$399 reg.

single qua mit

limited offer D

Part # STD65F

ForthCard, D

2kbyte RAM, 2

and Manuals.

HiTech Equipment Corporation

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

FORTH Dimensions
Published by the
Forth Interest Group

Volume VI, Number 5
January/February 1985

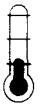
Editor
Marlin Ouverson

Production
Cynthia Lawson

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

Subscription to *Forth Dimensions* is free with membership in the Forth Interest Group at $15.00 per year ($27.00 foreign air). For membership, change of address and to submit material for publication, the address is: Forth Interest Group, P.O. Box 8231, San Jose, California 95155.

Symbol Table



Simple; introductory tutorials and simple applications of Forth.

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

Advanced; requiring study and a thorough understanding of Forth.

Code and examples conform to Forth-83 standard.

Code and examples conform to Forth-79 standard.

Code and examples conform to fig-FORTH.

Deals with new proposals and modifications to standard Forth systems.

FORTH Dimensions

FEATURES

DEPARTMENTS

Letters to the Editor

Strong is Weak

Editor:

One of the weaknesses of Forth is also its greatest strength: the simplicity of the intrinsic language and its extensibility. This means that Forth itself is wonderfully compact and clean. However, a compact, clean language is not necessarily ideal for software development.

Pascal is very clean and compact, yet ISO Pascal is impossible to use for any real work; it is, among other things, so sparse that a programmer must use all sorts of tricks to do anything. On the other hand, languages like PL/I or Ada supply everything you need (and more) to do anything you want.

As the writer for the "Designers Debate" column in *Computer Language* magazine (yes, I'm responsible for the Forth C column), I have been following the Fortran 8x committee (X3J3). Fortran, although it is obsolete, was, in its time, a clean, simple language providing both access to the machine and high-level constructs.

However, in attempts to keep the language alive and current, many new "features" are being added to it. Historically, compiler vendors would write compilers which would accept the standard (Fortran 66 or 77), then add various "features" or "extensions" to make it a little easier to use. Variations proliferated.

Recently, at a public forum, the panel was asked, "With all the additions, isn't Fortran losing its simplicity? Can't you just leave the language alone and standardize on the extensions?" The response was, "Yes, it's too large; yes, we want to keep it simple; but whose features do we accept?"

The same thing has happened with Pascal. There are so many different extended versions of Pascal that portability is out of the question. Indeed, if you want to use Pascal, you will probably have to sacrifice portability in favor of usability.

Now, Forth is not a dinosaur like Fortran, nor is it "closed" like Pascal. It is one of the few languages which support extensions of themselves. Therefore, it seems, there can be as many "extended" versions of Forth running loose as there are Forth programmers.

One of my problems as the Forth guru where I work was to maintain some semblance of order with the Forth system we use. With any other language, this is nearly trivial. With Forth, I spent a lot of time organizing the mass of "extensions" that people threw at me daily. The result was a collection of in-house standard extension words which were released to all programmers. Naturally, it was not an easy task.

Forth, like C, is particularly well suited to system-level work. Many people seem to think that it could use more high-level data constructs, judging by the articles I've seen in *Forth Dimensions* and the annual Forth issue of *Dr. Dobb's Journal*. This is undoubtedly true.

I propose, however, that the language be kept as simple and clean as possible. We should avoid the numerous, perpetually changing "standards" that Fortran has undergone. I believe that it would be desirable to start looking at the various extensions to Forth as areas requiring standardization. The Forth Vendors Group (FVG) floating-point proposal, for example, as set forth in the latest Forth issue of *Dr. Dobb's Journal*, is a major step in that direction.

In the published Forth/C debate, the C proponent made the claim that Forth lacks standard utilities and tools. David Lawson from King City, California wrote to say, "That was true in the past, Forth proponents are now making a serious effort to keep a standard language with a library of utilities slowly being built. For instance, almost every issue of *Forth Dimensions* has some utilities or ways to build them."

This is true, I admit. However, this is just a collection of utilities, not necessarily standardized, and sometimes flawed. An acquaintance of mine discovered some serious errors in a floating-point utility published in *Forth Dimensions* a while back.

There are numerous common packages that could be standardized to some degree in the name of portability. Floating point is one. Others include graphics, terminal I/O, high-level data structures, multi-tasking, etc. The idea is to come up with descriptions of user-level words, giving the stack picture and purpose. Implementation details should be avoided.

Naturally, this is a major task. It should not be delayed overly much, however. At some time, the investments made in individual extensions will become so great that any attempt to find order will be met with, "Sure, standardize, but make mine the standard."

Sincerely,

Ken Takara
San Jose, California

Back to Recursion

Dear Editor:

In the article on recursion (*Forth Dimensions* VI/4), Michael Ham discusses Charles Moore's suggested word **RECURSIVE** which would be used in the definition of **GCD** in the following manner:

```
: GCD ( a b --- gcd ) RECURSIVE ?DUP IF SWAP OVER MOD GCD THEN ;
```

However, according to Ham "...the word ; must now clear (rather than toggle) the smudge bit when the definition is complete." If you define the word **RECURSIVE** as follows, using the word **SMUDGE** from MVP-FORTH:

```
: RECURSIVE SMUDGE ; IMMEDIATE
```

the definition of **GCD** becomes:

```
: GCD RECURSIVE ?DUP IF SWAP OVER MOD GCD RECURSIVE THEN ;
```

then you don't have to worry about the word ; clearing the smudge bit because **RECURSIVE** again toggles the smudge bit before ; is executed. Another example is the definition of the word **N!** ("n factorial"):

```
: N! ( n --- n! ) RECURSIVE DUP 1 = NOT IF DUP 1 - N! RECURSIVE * THEN;
```

where n is greater than or equal to one.

Sincerely,

Ron Apra
San Jose, California

Editorial

Author Recognition

The dust had barely settled following our return from three weeks of Forth conferences in China, when the annual FIG convention and FORML Asilomar Conference swept us up. We are back, firmly in the saddle again, and have finally dealt with the accumulation of correspondence and prospective articles.

We are now able to announce FIG's new policy in recognition of authors of Forth articles. In order to encourage authors to write about Forth, a one-year FIG membership will be awarded to authors of Forth articles whose published length is at least one page (includes authors of material published in *Forth Dimensions*). Authors of published Forth-related letters to editors of *non-Forth-specific publications* will receive a coupon worth ten dollars toward FIG membership. A few restrictions: all credits apply only to FIG membership for the single year following publication and are non-transferable; author must send a photocopy of the original item, magazine name and issue date to the Forth Interest Group within sixty days of publication (accompanied by an English synopsis/translation if original is in another language); and eligible items must have been first published after October 1, 1984. So get busy — renewal time is coming!

Finally, this month we are pleased to introduce a new department. Each "Application Tutorial" will convey useful code and theory to relative newcomers. John S. James has volunteered to serve as the primary contributor of this material. John's abilities as an educator and Forth programmer are well known, and we are proud to present his work in our pages on a regular basis.

— Marlin Ouverson
Editor

Grass-Roots Forth

Dear FIG:

It appears to me that if FIG and its members are really serious about promoting the widespread use of Forth, then less effort should be directed toward trying to convert members of the established computer industry away from their own favorite computer language, and more effort should be spent trying to reach the grass-roots computer user, the entry-level computer programmer, the innocent children and wide-eyed home computer neophytes who have not yet been conditioned or committed to one language or another.

Let's face it, if children (and other beginning programmers) were equally exposed to BASIC and Forth, many (and probably most of the better programmers) would prefer programming in Forth! And as these programmers advance to high levels, they would certainly carry Forth along with them.

Step One: Set a goal. So how can the Forth Interest Group bring this situation to pass? The first step is to adopt a goal and encourage as many FIG members as possible to join in the effort. Personally, I would like to see FIG adopt the goal of *actively promoting and supporting the Forth language, particularly in regards to entry-level programmers.*

If such a goal seems inappropriate for the organization as a whole, then I challenge every FIG member personally to adopt such a goal and to do whatever he can to encourage the growth of Forth among newcomers.

Step Two: Make Forth available. If Forth is to compete with BASIC, it must be as accessible as BASIC. This means it must be *free!* There is no other possible way for beginners to have an opportunity to seriously try it and find out whether or not they like it. A public-domain version of Forth (PD-Forth) must be made available for their computer — one which they can freely copy and distribute to their friends. It is not enough that it be available for their microprocessor (as fig-FORTH has been in the past), it must be computer-specific and *ready to run.*

I would not at all be surprised to find out that suitable PD-Forths (probably versions of fig-FORTH) have already been written for all of the more popular home computers. But if so, then where are they?

Actually, a large number of PD-Forths is not necessary. Only three — for the Commodore 64, Apple IIe compatibles and IBM PC compatibles — are really needed to reach the vast majority of today's entry-level programmers. So even if no current PD-Forths existed, three dedicated individuals could do the whole thing. Or a couple of far-sighted vendors could also do the job by releasing their obsolete versions of Forth. Not only would they be increasing the number of prospective purchasers of their other Forth products, they would also be obtaining free distribution of any advertisements placed on their PD-Forth boot-up screen.

Step Three: Make Forth visible. It is not enough to make Forth available, it must also be visible because many programmers (particularly at the entry level) may not be aware of its availability or its potential.

I am convinced that there are many opportunities for individual FIG members to have fun, gain recognition, get paid and promote Forth all at the same time by writing articles in popular computer programming magazines. I know that magazines such as *COMPUTE, Gazette, RUN* and *Ahoy* are always looking for good, original programs and are even willing to pay for them. Forth is one of the few useful languages which can be made short enough to be published in such magazines. And once the language it-

self has been published, the door is wide open for tutorial articles and application programs.

Step Four: Support the new Forth users. Since FIG is already supporting Forth users, nothing new would have to be done here, except placing more emphasis on beginners as their numbers increase.

And in case you're wondering why I don't follow my own advice and write my own version of PD-Forth (ho, boy, don't I wish I could!), it's because I'm still a beginner myself and I still need *your* support.

Lionel Hewett
Kingsville, Texas

Death Wish

Dear Marlin:

I believe the Forth standards committee has a death wish as they continue to create new dialects with each "standard" they emit. My current concern is that readers of the August 1984 *BYTE* do not turn off to Forth when they read "Forth-83: the Evolution Continues." A careful reader will soon realize Forth-83 is more revolution than an upward evolution from Forth-79 or the FIG model (Forth-78).

By using new names, dialects need not be created. For example, the Forth-83 **DO LOOP** and **DO +LOOP** are very different animals from the Forth-78 **DO LOOP** and **DO +LOOP**. I would like to be able to use both (while maintaining portability with the FIG model and without having to add extensions). I would be able to use both if the Forth-83 versions were named **DO83**, **LOOP83** and **+LOOP83**. More important to me: I would use the new keywords only when their new properties are relevant, while continuing to use the proven Forth-78 keywords as before. In this way, evolution can take place with a minimum of risk. Note my use of Forth-78. The Forth-79 Standard is incomplete and, thus, useless to me.

As an experienced Forth programmer, I find the changes from Forth-78 (listed in *BYTE* table three) to be marginal and, in the main, unnecessary. For example, / and **/MOD** now use floored division, which is very useful in some applications. However, that is not the division the world knows and uses. I believe the committee is naive not to retain "common division" and use new names for new animals. Furthermore, implementations of these new keywords have not been fully evaluated; practical consequences are not known.

I do not perceive any preemptive benefits. More specifically, the new definitions are not such significant improvements over the old that they justify replacement — additions would make more sense.

I suggest the committee address new areas such as graphics, mathematics and data bases in lieu of massaging yesterday to death. Finally, I recommend the committee recognize the intrinsic democracy of extensibility by using it.

Sincerely yours,

Nicholas Pappas
Oakland, California

How to Learn Forth

William F. Ragsdale
Hayward, California

"Ask the Doctor" is Forth Dimensions' *health maintenance organization devoted to your understanding and use of Forth. Questions of a problem-solving nature, on locating references, or just about contemporary techniques are most appropriate. When needed, our good doctor will call in specialists. Published letters will receive a preprint of the column as a direct reply.*

Your doctor's mailman has just left, leaving an interesting assortment of questions and comments from readers such as yourself. While reading and sorting, your faithful practitioner has just placed four more letters on a looming, teetering pile. It wobbles and slumps to the floor — a call to action!

What common thread runs through this accumulation? The frustration of people struggling to get a toehold to learning Forth. They cry out:

—How can I get started?
—Which Forth?
—I have no documentation!
—Whom do I ask?

To assist those of you represented by these forty letters, your faithful counsellor will address 1) selecting a Forth system for learning, and 2) a plan for systematic self-education.

To clarify the task of learning Forth, take as your goal the ability to read Forth programs by others and to write modest programs of your own. Three keys are needed to unlock the mystery of Forth. First is your desire. Next is a learning environment. Finally, a study plan to guide your efforts. If any of these are missing, expect trouble; when all are present, you soon should experience a thrill from the mastery of a new intellectual resource.

To aid in your learning task, one noble authority (guess!) suggests the following list of items and their relative importance to you:

- Specific computer — 5%
- Forth version and documentation — 10%
- *Starting Forth* text — 25%
- Attitude + method — 60%

Most correspondents say something like, "Which Forth is the best for me to learn on?" or "I have Forth from the TI users' library but no documentation." You should realize that the time invested in learning Forth will be fifty to 300 hours. We first wish to evaluate the suitability of specific systems for learning. If you are missing elements of the above list, the required time could be doubled or, worse, you could become so demoralized that you abandon the effort.

The Computer

Which computer is best? Whatever one you own now! Forth is the great equalizer: From a learning standpoint, hardware carries little impact. Later, some systems may aid specific applications work but, for now, not to worry. You can start with a VIC-20 or better — this means a computer with full keyboard and at least 16K of memory. The Sinclair or Jupiter Ace is not suggested, as you will be fighting the keyboard or memory limitations. A disk should be available, even if you don't begin with one. Mass storage is a key element in Forth; your learning will stop abruptly without easy access to data storage. Cassette tape is a distant, but livable, second choice.

To summarize Forth's minimum needs in a computer:

- 16K memory
- full, ASCII keyboard
- 50K mass storage
- 16-line by 64-character display

Forth Version

The Forth you choose should be evaluated for its learning value. Just because you find it in a free library, or get a copy from a friend, does not mean it will be necessarily appropriate for your learning effort. A questionnaire is included with this month's column. The good doctor suggests that, after using it to evaluate your choice, you mail in a copy. A summary of the results will be published in future issues of this column for the aid of all.

The score card has a maximum possible point total of thirteen. Any version scoring less than seven will impede, rather than aid, your effort. A number of items have been included without point values. This information will be helpful for advanced use, but will not play a significant part in your initial skill development.

The importance of certain features to the learning process is shown by the point/rating scale. Other scales would be appropriate for other needs; this one is slanted only to your learning needs.

Your candidate system should be based on a broadly documented language model. For the future, Forth-83 is growing in importance. But to date, Forth-79 and fig-FORTH are better represented in current books. Not recommended are variant dialects such as SL5 (Stackworks) or TransForth. These systems either have renamed keywords (e.g., **C@ *and* C!** or are missing some (e.g., **DOES>**).

Since our textbook will be Leo Brodie's *Starting Forth*, either polyFORTH or Forth-79 versions are most appropriate. But your future work is likely to be most affected by Forth-83, and the learning differences are small. Since that book discusses aspects of fig-FORTH, polyFORTH and Forth-79, you will unavoidably experience some confusion — though not insurmountable — regardless of your version of Forth.

Forth Documentation

Our scorecard suggests at least fifty pages of Forth documentation should

accompany your system. With the wealth of available books, you need not depend solely on your supplier to teach you Forth. In fact, most manuals give only the facts, covering fundamental word definitions, editing and compiling. Style, examples and application will be learned elsewhere. So...if your manual covers the basics of installation, use and options, education can be derived from other sources.

Your system documentation must cover machine-specific details:

1) How to start up (boot)
2) Customization or loading options
3) Peripherals supported (e.g., printer, alternate disks)
4) Use of the editor
5) Use of host operating system or filer
6) Glossary with deviations/extensions from a base-line system (Forth-83, Forth-79, fig-FORTH)

You should expect initial load modules (boot-up object code) of 6 to 15K, and possible options on top of that. Access to mass storage (disk or tape) and a text editor are essential.

Study Plan

If you are a total novice with Forth, read *Understanding Forth* by Reymann. Each aspect of Forth is covered in a brief, enlightening overview. The material provides an excellent knowledge base which will be detailed in *Starting Forth*. (Both books are available from the Forth Interest Group — see order form in this issue.)

Next, you should work through *Starting Forth* with your system. This means line-by-line, example-by-example, on your system. It does not mean reading in bed, just before dropping off to sleep. The chapters and pages to be covered are tabulated below. Work every example. Master every problem.

When stuck, don't skip. Look for an equivalent use or definition in your system vendor's manual. Ask a friend. Call the vendor. Ask why he deviates from *Starting Forth*. Your Forth is extensible and modifiable. Ask the vendor why he doesn't provide an overlay to match *Starting Forth*.

By vendor, of course, the good doctor is referring to the person or firm who developed or tailored your system for your computer. Users groups (and FIG) generally are not staffed to handle such problems, so find the real originator or get a system with better support.

Have you opened your mind to learn? Forget the methods of BASIC, C and Pascal for the time being. Later you will have the perspective to compare and evaluate, but not while learning.

Study Sequence

Our study plan within *Starting Forth* covers nearly all of the Forth-83 Required Word Set. These words are also in Forth-79 and so are well covered by Brodie. Double numbers, numeric input and numeric output format conversion should be skipped now for later, more advanced study.

1) Fundamentals, pp. 1-30
2) Stack, pp. 31-55 (omit pg. 50)
3) Editor, pp. 57-87
4) Conditionals, pp. 89-106
5) Return Stack, pp. 107-112 (defer pp. 113-122)
6) Looping, pp. 127-147
7) Numbers, pp. 149-163 (defer pp. 164-175)
8) Data, pp. 183-192 (defer pp. 193-194, 195-213)
9) Vocabularies, pp. 242-243 only
10) Storage, pp. 253-268
11) Examples, pp. 317-348

Word Competency

At the conclusion of your study program, you should have experience with, and be comfortable in the use of, the following Forth words:

Math:

+ - */ /MOD 1+ 1- 2+ 2- ABS NEGATE

Input/Output:

KEY EXPECT COUNT CR SPACE SPACES EMIT ." TYPE -TRAILING

Comparisons:

= - < U< > 0= 0< 0> MIN MAX

Definers:

: ; CONSTANT VARIABLE CREATE

Dictionary:

FORGET ALLOT HERE , C,

Logic:

AND NOT OR XOR

Memory:

! @ +! C! C@ CMOVE CMOVE> FILL ERASE

Output:

. U.R U. HEX DECIMAL

Stacks:

SWAP DUP OVER ROT DROP PICK ROLL ?DUP >R R> I

Control:

IF ELSE THEN DO LOOP +LOOP BEGIN UNTIL LEAVE

Disk Use:

LIST LOAD FLUSH COPY WIPE UPDATE BLOCK EMPTY-BUFFERS

Bibliography

Joseph Reymann, *Understanding Forth*, Alfred Publishing Co., Prentice-Hall, 1981.

Leo Brodie, *Starting Forth*, Prentice-Hall, 1981.

Evaluation Form
Forth Systems for Learners

For learning value, a system should score seven points or more. Please send a copy of your evaluation to "Ask the Doctor," *Forth Dimensions*, P.O. Box 8231, San Jose, California 95155.

Product: _____

Vendor: _____

Distributor: _____

Runs on: _____

Cost: _____

_____ Object size (one point for each 5K up to 20K)

_____ Options in source form (one point if these comprise at least ten blocks)

_____ Editor (one point if it conforms to *Starting Forth*; no for a FIG-style editor)

_____ Editor display (scrolling lines earn no points; screen earns one point)

_____ Mass storage (no points for tape; *minus three points* if disk is not supported; one point for hard disk support)

_____ Dialect (two points for Forth-83; one point for Forth-79; one point for polyFORTH; no points for fig-FORTH)

_____ Absolute conformance to one of the above dialects: add one

_____ Documentation (one point if more than fifty pages)

_____ Support (one point if vendor answers telephoned questions)

_____ Assembler (yes/no; no points)

_____ Floating point (yes/no; no points)

_____ Access to host files (yes/no; no points)

Please list any other extensions: _____

Application Tutorial

Simple Modem I/O Words

John S. James
Santa Cruz, California

"Application Tutorials" show how to use Forth to get results. They assume some knowledge of the language; for example, they won't explain what a stack is, or how to use **IF** *or* **DO**. *Nor will they cover advanced material, such as metacompiling a new Forth system. Instead, they focus on tools for everyday work.*

The Forth-83 Standard will be used. Often, we will use the excellent F83 implementation of that standard by Laxen and Perry. Most practical applications need some words not in the standard (for example, the assembler mnemonics), and we will explain such words when necessary.

We will be guided by the Forth coding conventions being developed by Kim Harris. These conventions concern the physical appearance of the code and do not affect programming techniques. Those unfamiliar with the conventions will notice little change from practices common already, except for the **;P** *to mark the "purpose field" of interface comments.*

The code in these tutorials is released into the public domain without restriction.

Scope of This Tutorial

We will show how to write to and read from a serial communication port connected to a modem. In this example, the computer is an IBM PC (compatibles should run the same code), and we will do the I/O through system calls to the BIOS (Basic Input/Output System), which is provided in ROM with the computer. If you are using a different computer, this code can serve as a model for implementing the same modem operations on it.

We will show how to define system calls, using assembly language. Then we will define general-purpose modem words. Finally, a dumb-terminal program will illustrate how to use the I/O words; this program can also be useful in its own right. Also, we will briefly discuss some techniques not used in this code, such as buffered I/O and direct I/O to port addresses.

Overview

Modems can be connected to computers in different ways. Often, a twenty-five-pin cable connects a modem to a serial port. (Most of the twenty-five pins are not really necessary, and sometimes a five-pin cable is used instead.) Also, the port and modem are often combined in the same plug-in card, or the modem may be built into the computer.

The programmer often has a choice of how to read from or write to the port (and the attached modem). The computer manufacturer may provide operating-system calls for this purpose, or the programmer may bypass the operating system and use the port directly. It's usually best to use the system calls if they can do the required job, in order to make the program more likely to be compatible with possible future upgrades of the computer. In this example, we will use system calls to the ROM BIOS software provided with the IBM PC.

Defining the System Call

Some Forth systems come with some or all of the system calls already defined. Others have a "call" word which allows easy access. But in this example, we will write from scratch the system call we need, using the Forth assembler.

If you are using the F83 system on an IBM PC or compatible, you can just copy this code. A different Forth-83 system (not the F83 implementation) may have different assembler mnemonics or other symbols, so the code would look different, although identical machine language would be generated. And other Forth systems for the IBM PC might not use the 8088 stack (as F83 does), or they could require that registers used for the call be restored. In such a case, consult documentation on how to use the system's assembler, or analyze the source of simple code words such as **DUP** or **+**.

For the IBM PC, the documentation on how to use its BIOS system calls is in the *IBM Personal Computer Technical Reference Manual*, Appendix A, the ROM BIOS listing, in comments in front of the BIOS code which implements each system call. We will not reproduce that information here but, basically, you load the AH, AL and DX registers with arguments, then do an INT 14 (hex) to make the call. Results come back in the AH and AL registers.

The Code

On screen #3 (the first screen), **INITIAL-BITS** holds the bit pattern which tells the BIOS software the speed, parity and other options wanted. **STATUS-BITS** will hold any status information returned by the system call.

CALL14 is our code word to do interrupt fourteen (hex), which calls the BIOS serial-communication software. The first stack argument contains the values to be placed into AX (the AH and AL registers) on the 8088. The top stack argument must contain a zero or one to select RS-232 card number one or two (the computer supports two serial ports). **NEXT** assembles the instruction to return control to Forth, and **END-CODE** ends the code definition.

CALL14X is defined in terms of **CALL14** and just makes the call more convenient by providing the number of the RS-232 card in use and by automatically saving a copy of the results returned (including all status bits)in **STATUS-BITS**.

In screen #4, **M-EMIT** writes a character to the port, much as the Forth word **EMIT** writes to the terminal.

The **100 +** (in hex) puts a one into the high-order byte of the word whose low-order byte contains the character to be written. These bytes will get into the AH and AL registers, respectively; a one in the AH register is a command to the BIOS serial-communication routine to write the character in the AL.

M-EMIT relies on the BIOS to wait, if necessary, until the port has finished writing the last character and is ready to write the next. If you are implementing these words on a different computer, you might have to put a delay loop in **M-EMIT**.

CALL14X always returns a result which is the value of the AH and AL registers; a copy of that result has already been saved in **STATUS-BITS**. **M-EMIT** drops the stack copy, because it does not return any argument on the stack.

M-STATUS performs a call to read current status information. The system returns sixteen status bits. But in this simple example, we use **M-STATUS** only once (and use only one of the returned bits), in defining **M-KEY?** which tests whether a character is now ready to be read from the modem, and returns a true/false flag.

M-KEY reads a character from the modem. First, it waits until a character is present, then it executes **CALL14X** with a two in the AH, which tells the BIOS to read from the modem. Finally, it masks status bits off of the returned character. (These bits have all been saved in **STATUS-BITS** already, so they are not lost.)

We used the delay loop **BEGIN M-KEY? UNTIL** because different computers handle timeout differently if no key comes in from the remote terminal; for example, the IBM PC times out in one second, while some computers wait indefinitely. We wait until a character is available, in order to define **M-KEY** in a way that can work the same on all equipment.

M-INITIALIZE must run before any port I/O is done. In more advanced programs it may be executed more than once, to change the communication options (speed, parity, stop bits and data bits), perhaps to try out different speeds to match a remote terminal dialing in.

Terminal Program

Screen #5 uses these modem words to build a simple dumb-terminal program. The **BEGIN UNTIL** loop keeps a "false" flag on the stack as an exit switch; the loop will exit if the flag gets set to "true." The loop just keeps looking for any key from the modem and, if any key comes in, writes it to the terminal. And vice versa, from the terminal to the modem.

The word **KEY?** is outside the Forth-83 Standard. This word returns a flag which is "true" if a key has been struck at the local keyboard and is available for **KEY** to read; or is "false" if no key is available. Most Forth systems have such a word, perhaps with a different name such as **?TERMINAL**.

There should be a way for users to exit from the terminal program so that they do not need to reboot when they are finished using it. We chose control-D (value "4") as the character which can be typed from the terminal to exit the loop.

Sample Session

Figure one shows a sample use of this program. We load the program first, then execute **TERMINAL**. Then we start typing to the port (and modem).

In this example, we are using a Hayes modem with its switches in the factory setting. The typed AT gets the modem's attention, and the D is a command which instructs the modem to dial the following telephone number. In this case we called the FIG-Tree, a twenty-four hour computer conference about Forth, which is open to anyone with a 300 BPS modem.

Further Development

You could improve **TERMINAL** fairly easily, for example, to have it save a copy of input and/or output in memory, or to allow text to be typed into memory while offline for

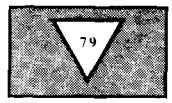
transmission later (perhaps hours later, by using a timer if available, or a delay loop otherwise) in order to save phone charges on long-distance calls.

However, the modem input/output words defined here could not receive data while the computer was busy for an extended time on some other task, such as disk I/O. For example, these definitions could not be used to receive files too large to fit into memory unless the sending computer could pause while disk output was taking place; otherwise, characters being transmitted would be lost while the computer was busy with the disk. More advanced communication software would have an interrupt routine to receive characters immediately when they came in, no matter what else the computer was doing at the time, and would place the received characters into a buffer. Then, **M-KEY** would get its characters from the buffer. A separate buffer would be used for **M-EMIT**.

```
A>F83

8086 Forth 83 Model
Version 2.1.0 Modified 01Jun84
OPEN FD.BLK  ok
3 LOAD 4 LOAD 5 LOAD TRUE isn't unique FALSE isn't unique  ok
TERMINAL
AT D 1-415-538-3580
CONNECT


     WELCOME TO THE CONFERENCE TREE

          TERMINAL LINE LENGTH (20-80,
            OR CARRIAGE RETURN FOR 80)?
          LOWER CASE OK (C/R=YES)?

TYPE 'READ HELP' ANY TIME
OR 'READ CONFERENCES' TO START

'S' KEY TO PAUSE OR RESUME PRINTING

FOR COMPLETE INSTRUCTIONS ON USE, TYPE "R HELP C"  (OMIT THE QUOTES!)

COMMAND?  READ CONFERENCES STARTING 1-OCT-84

*** CHUCK.MOORE-OCT-27    22-OCT-84
  PARENT=SILICON-VALLEY        USAGE= 79
************************************************************
* The Silicon Valley FIG chapter is proud to announce      *
* that it October guest speaker will be Charles Moore      *
* the inventor of the FORTH language.                      *
*                                                          *
* He will tell us about his latest project, the CHIP.      *
*                                                          *
* The meeting will be held at the usual time and place:    *
*                                                          *
* Dysan auditorium at 1:00 pm, Saturday, October 27.       *
*                                                          *
* Notice: Silicon Valley FIG members have seating priority. *
*                                                          *
************************************************************
*** DEMENTED-TYPO          30-OCT-84
```

```
Scr # 3          A:FD.BLK
  0 \ Communication: constants, variables, system call     JJ 11Nov84
  1 -1 CONSTANT TRUE    0 CONSTANT FALSE  \ For better clarity
  2 1 CONSTANT COM-PORT    \ To address RS-232 port number 1
  3 VARIABLE INITIAL-BITS    \ To tell BIOS the speed, parity, etc.
  4 2 BASE !   01001010 INITIAL-BITS !   DECIMAL
  5    \ Binary for convenience; 300 bps, odd parity, 1 stop 7 data
  6    \ For more info, see IBM PC Technical Reference, Appendix A
  7 VARIABLE STATUS-BITS    \ To hold status info from system calls
  8 HEX
  9 CODE CALL14    \ xax1 n -- xax2    ;P Communication I/O call
 10    DX POP    AX POP    14 INT    AX PUSH    NEXT    END-CODE
 11 : CALL14X    \ xax1 -- xax2    ;P Call selected card.  Save bits.
 12    COM-PORT 1-    CALL14    DUP STATUS-BITS ! ;
 13 DECIMAL
 14
 15

Scr # 4          A:FD.BLK
  0    \ Communication: write, read, get status, init      JJ 11Nov84
  1 HEX
  2 : M-EMIT    \ b --    ;P Write to communication port
  3    100 + ( puts '1' in 'AH' )    CALL14X DROP ;
  4 : M-STATUS    \ -- x   ;P  Get port status, 16 bits; see Tech Ref
  5    300 ( '3' in 'AH' ) CALL14X ;
  6 : M-KEY?    \ -- ?    ;P True if modem char. ready to read
  7    M-STATUS 100 AND ( Test the data-ready bit )
  8    IF    TRUE    ELSE    FALSE    THEN ;    \ Improve the True flag
  9 : M-KEY    \ -- b    ;P Read a character
 10    BEGIN M-KEY? UNTIL    200 CALL14X    00FF AND ;
 11    \ '2' To 'AH'.  Mask off status bits returned.
 12 : M-INITIALIZE    \ --    ;P Initialize the port
 13    INITIAL-BITS @ CALL14X DROP ;
 14 DECIMAL
 15

Scr # 5          A:FD.BLK
  0    \ Communication: simple dumb terminal               JJ 11Nov84
  1 : TERMINAL    \ --    ;P Dumb terminal; exit on Control-D
  2    CR    M-INITIALIZE
  3    BEGIN
  4      FALSE    \ Exit switch: default is don't exit the loop
  5      M-KEY? IF    M-KEY EMIT    THEN
  6      KEY? IF    KEY 127 AND ( Mask off parity )
  7        DUP 4 = IF    DROP ( The '4' )    DROP TRUE ( Exit loop)
  8        ELSE    M-EMIT    THEN    THEN
  9    UNTIL ;
 10
 11
 12
 13
 14
 15
```

An Augmented TRACE



Andreas Goppold
Hamburg, West Germany

I have come to appreciate the interactive power of Forth, especially since discovering the ability to display all information relevant to a program. The **TRACE** function is a very powerful tool, indeed, and I have elaborated a little on Paul van der Eijk's utility (*Forth Dimensions* III/2). The changes I made are as follows:

1) Redefined a few kernel definitions (screen 17). This gives a more complete trace when the nesting level of a new word is only one. I could have rewritten the **NEXT** routine to trace everything, but that remains as a project for the future.

2) Renamed the routine. I call it **TRON** since the computer is effectively making a movie of its own operation (screen 16).

3) The routine now finds the nesting level of a colon definition. This is heuristically done by searching the return stack and throwing out addresses which can't be fitted to an acceptable NFA. In this manner, one can eliminate most of the static introduced by the use of >**R** (screen 14, lines 7 — 9).

4) The stack now is printed bottom-left, top-right. This way, data that stays on the stack remains in the same position on the printout, making it much more readable (screen 15, lines 8 — 9).

5) Use of decimal numbering. I can't think in hex; since Forth is nice enough to do all the translation work for me, I allow it to do so.

6) Ability to stop and restart the **TRACE** printout. Also, one may now set nesting levels to control when **TRON** will begin to give full information. One can have prespecified nesting levels such that for most of a program **TRON** will print only up to a certain nesting level, let's say two, and then from a specific word on, it will print up to six nesting levels (screens 10 — 11).

7) Printing the names of words whose addresses are on the return stack (screen 13) is now supported.

8) A **HELP** screen is available (screen 9).

9) Indentation of **ID.**s to indicate nesting depth (see example).

Sample Usage

Screens 64 — 65 contain the definitions of words which are traced below. The first word traced is " which is defined beginning on line 8 of screen 65. The next word traced is $@ and is defined on lines 10 — 11 of screen 64.

```
Screen # 64
 0 ( String stack extension, continued  -  mod. A. Goppold )
 1 : $P!  $0 $P !           ; ( $P! empties $STK by resetting $P )
 2     ( Returns value of $P )
 3
 4 : $P@  $P @ DUP DUP $0 > SWAP $END < OR IF
 5         ," $STK-PTR-ERR ! " $P! QUIT    THEN    ;
 6 ( new$p  --- )
 7 : CHK$OVL  DUP $END < IF ," $STACK OVERFLOW ! " QUIT THEN   ;
 8
 9 ( adr len  ---                        fetch string to $STK )
10 : $@   DUP >R $P@ SWAP - 1- CHK$OVL 1+ SWAP OVER R CMOVE
11                                  1- R> OVER  C! $P ! ;
12                                                      -->
13
14
15

Screen # 65
 0 ( String stack extension, continued )
 1 ( addr  ---    ,   fetch  string together w/ length byte   )
 2 : $L@   DUP C@ 1+ $P@ OVER - CHK$OVL  DUP >R SWAP CMOVE
 3                              R> $P ! ;
 4
 5 : (")  R DUP 2+ SWAP @ ( moves in-line string to $STK )
 6   DUP 2+ R> + >R $@ ;
 7
 8 : "    ( if compiling emplace an in-line string to be )
 9        ( moved to string stack at execution time,     )
10        ( in direct exec. put enclosed string on string stack. )
11   22 STATE @
12   IF COMPILE (") 0 C, WORD HERE C@ -1 ALLOT DUP , ALLOT
13   ELSE 0 C, WORD HERE C@ -1 ALLOT HERE !
14        HERE DUP 2+ SWAP @ $@
15   ENDIF ; IMMEDIATE                                  -->
```

```
"                empty stack
  @                34 49116                              @
  C@             31947                                   C@
  !                 5 31946                              !
  DUP            31946                                   DUP
  SWAP           31946 31948                             SWAP
  @              31948 31946                             @
  $@             31948     5                             $@
    DUP          31948     5                             DUP $@
      $P@        31948     5                             $P@ EXECUTE $@
        @        31948     5 18076                       @ $P@ EXECUTE $@
        DUP      31948     5 17941                       DUP $P@ EXECUTE
         DUP     31948     5 17941 17941                 DUP $P@ EXECUTE
         SWAP    31948     5 17941 17941     0           SWAP $P@ EXECUTE
       SWAP      31948     5 17941                       SWAP EXECUTE $@
       -         31948 17941     5                       - EXECUTE $@
       CHK$OVL   31948 17935                             CHK$OVL EXECUTE
         DUP     31948 17935                             DUP CHK$OVL EXEC
       OVER      17936 31948                             OVER EXECUTE $@
       CMOVE     17936 31948 17936     5                 CMOVE EXECUTE $@
     OVER        17935     5                             OVER $@
     C!          17935     5 17935                       C! $@
     !           17935 18076                             ! $@ ok
```

```
Screen # 6
 0 ( Trace, mods, A, Goppold , latest mod: 25-01-84 )
 1 FORTH DEFINITIONS
 2 0 VARIABLE TFLAG    0 VARIABLE DEPTSAV    2 VARIABLE DEPTLIM
 3 0 VARIABLE INDSAV   0 VARIABLE RPFLAG     1 VARIABLE BPFLAG
 4 4 VARIABLE CSTLIM                         0 VARIABLE SLFLAG
 5
 6 HERE 11 + CONSTANT RSAV   50 ALLOT
 7 HERE 12 + CONSTANT BPSAV  30 ALLOT
 8 HERE 12 + CONSTANT NMSAV  30 ALLOT
 9
10                                                      -->
11
12
13
14
15

Screen # 9
 0 ( Trace, mods, A, Goppold )
 1 : TAB 40 OUT @ - SPACES ;
 2 : H CLEARSCREEN 20 2 GOTOXY ." TRACE HELP SCREEN"   CR  CR
 3 ." TRACE has the following options:" CR
 4 ." TRON starts TRACE " TAB ." TROF ends trace mode"   CR  CR
 5 ." ^S : halt TRACE" TAB ." ^Q : resume TRACE "        CR
 6 ." ^W : toggle SLOW mode" TAB ." ESC quit execution " CR
 7 ." ^P : (after ^S only) enter Printing mode,"
 8 ."  exit by ^S"                                       CR CR
 9 ." ^R : set nesting level 1 for TRACE " CR
10 ." TRACE is usually on nesting level 1.  A different nesting lev
11 el can be" CR ." specified, and when TRACE has reached a certain
12  word, Mode 2 is started. " CR CR
13 ." ^E : set Word where to enter TRACE mode 2" CR
14 ." ^D : set nesting level for TRACE mode 2 "  CR CR CR  ;  -->
15
```

```
Screen # 10
  0 ( Trace, mods. A. Goppold                    )
  1
  2 : BPT CR ." start TRACE mode 2 at : " BPSAV DUP 1+ IN$ DUP IF
  3        )R SWAP R    CMOVE R) SWAP C! THEN     0 BPFLAG ! ;
  4 : SLOW ."  "   1000 0 DO I DUP * DROP LOOP ;
  5 : SET-DPT CR ." trace depth level : " #IN DEPTLIM ! ;
  6 : FLIPSL  SLFLAG @ 0= SLFLAG ! ;
  7 : SET-CST CR ." constant depth level : " #IN CSTLIM ! ;    -->
  8
  9
 10
 11
 12
 13
 14
 15

Screen # 11
  0 ( Trace, mods. A. Goppold )
  1 : STOP  ."  "  KEY CONSOLE  CASE 16 OF PRINTER ENDOF
  2    15 OF CONSOLE  ENDOF         72 OF H ENDOF
  3    05 OF BPT      ENDOF         18 OF SET-CST ENDOF
  4    23 OF FLIPSL   ENDOF
  5    04 OF SET-DPT  ENDOF    ENDCASE ;
  6 : TTEST ?TERMINAL                IF CONSOLE   KEY CASE
  7    83 OF STOP    ENDOF   ( S )    72 OF H ENDOF
  8    19 OF STOP    ENDOF   ( ^S )   18 OF SET-CST ENDOF  ( ^R )
  9    05 OF BPT     ENDOF   ( ^E )
 10    23 OF FLIPSL  ENDOF   ( ^W )
 11    04 OF SET-DPT ENDOF   ( ^D )
 12    27 OF QUIT    ENDOF         ENDCASE THEN ;  -->
 13
 14
 15

Screen # 12
  0 ( Trace, mods. A. Goppold )
  1
  2 ( nfa len  --- )
  3 : TEST-LEN
  4  1 = IF DUP 1+ C@ 127 AND DUP 94 ( SWAP 32 ) AND ELSE 0  THEN ;
  5
  6 ( nfa len  --- )
  7 : TEST-CHR  OVER + OVER 1+ 1 ROT ROT
  8        DO I C@ DUP 32 ) SWAP 126 ( AND AND LOOP  ;
  9
 10 ( nfa flag  --- )
 11 : WRT-RET   RPFLAG @                                    IF
 12        IF    ID.          ELSE DROP ." - "    THEN     ELSE
 13        IF ."   " DROP     ELSE  DROP          THEN     THEN ;
 14                                                        -->
 15
```

```
Screen # 13
  0 ( Trace, mods. A. Goppold )
  1
  2 : RETPRT  RPFLAG @
  3    IF   OUT @ 80 SWAP - 50 MIN 0 MAX SPACES 8
  4    ELSE 14
  5    THEN 0  DO   RSAV @ 2+ I 2* + DUP R0 @ 4 - (
  6    IF @ 2- @ 2+  NFA DUP C@ 31 AND DUP 1 )
  7       IF  TEST-CHR   ELSE   TEST-LEN   THEN  WRT-RET
  8    ELSE  DROP LEAVE
  9    THEN  LOOP ;                                        -->
 10
 11
 12
 13
 14
 15

Screen # 14
  0 ( Trace, adopted from Paul van der Eijk, mods. A. Goppold )
  1  ( R0 @ RSAV @ - 2/ DEPTSAV @ 0= IF DUP DEPTSAV ! THEN DEPTSAV @
  2  - DUP 1 ) IF 15 MIN 1 DO I INDSAV ! ."   " LOOP ELSE DROP THEN)
  3
  4 : (TRACE)   TFLAG @     ( inserted as 1st word of definition )
  5 IF  RP@ RSAV ! TTEST
  6   BPFLAG @ 0=
  7   IF R 2- NFA 1+ NMSAV BPSAV C@ CMOVE NMSAV DUP BPSAV C@ + 1-
  8     DUP C@ 127 AND SWAP C!  BPSAV DUP 1+ SWAP C@ S= ( match wd )
  9     IF 1 DUP BPFLAG !
 10     ELSE ( word not yet occured, chk r-depth if less than limit)
 11       R0 @ RSAV @ - 2/  ( :r.depth )
 12       DEPTSAV @ 0= IF DUP DEPTSAV ! THEN
 13       DEPTSAV @ - ( :adj.depth ) DEPTLIM @ (     ( 1 if ( lim )
 14     THEN
 15   ELSE 1  ( bpflag=1 )    THEN          -->

Screen # 15
  0 ( Trace, adopted from Paul van der Eijk, mods. A. Goppold )
  1       R0 @ RSAV @ - 2/  ( :r.depth )
  2       DEPTSAV @ - ( :adj.depth ) CSTTLIM @ (  AND
  3   IF CR  0 RPFLAG ! RETPRT
  4      R 2- NFA     ID.       ( back to NFA for name )
  5      OUT @ 30 SWAP - 30 MIN 0 MAX  SPACES ( tab over )
  6      SP@ S0 @ =
  7        IF ."  empty stack"
  8        ELSE SP@ 2- DUP 16 + S0 @ 2-   MIN      ( show up to top )
  9          DO  I ."  " @ 0 5 D.R -2 +LOOP        ( 8 words of st. )
 10        ENDIF
 11      4381   @ IF 1 RPFLAG ! RETPRT   THEN
 12      SLFLAG @ IF SLOW                THEN
 13   THEN
 14 ENDIF   ;                                        -->
 15
```

1985 Rochester Forth Conference

June 12 - 15, 1985
University of Rochester
Rochester, New York

The fifth Rochester Forth Conference will be held at the University of Rochester, and sponsored by the Institute for Applied Forth Research, Inc. The focus of the Conference will be on Software Engineering and Software Management.

Call for Papers

There is a call for papers on the following topics:

- Software Engineering, and Software Management Practices
- Forth Applications, including, but not limited to: real-time, business, medical, space-based, laboratory and personal systems; and Forth microchip applications.
- Forth Technology, including finite state machines, meta-compilers, Forth implementations, control structures, and hybrid hardware/software systems.

Papers may be presented in either platform or poster sessions. Please submit a 200 word abstract by March 30th, 1985. Papers must be received by April 30th, 1985, and are limited to a maximum of four single spaced, camera-ready pages. Longer papers may be presented at the Conference but should be submitted to the refereed *Journal of Forth Application and Research*.

Abstracts and papers should be sent to the conference chairman: Lawrence P. Forsley, Laboratory for Laser Energetics, 250 East River Road, Rochester, New York 14623. For more information, call or write Ms. Maria Gress, Institute for Applied Forth Research, 70 Elmwood Avenue, Rochester, NY 14611 (716) 235-0168.

```
Screen # 16
 0 ( Trace colon words, continued )
 1                                        ( turn on trace mode )
 2 : TRON  1 TFLAG !   0 DEPTSAV !  1 BPFLAG !
 3   CR 20 SPACES ." Type  H for Help Menu "  ;
 4 : TROF  0 TFLAG !               ;  ( turn off trace mode )
 5
 6 : :             ( redefine colon to insert trace word )
 7   ?EXEC         ( must be executing )
 8   !CSP          ( compiler security )
 9   CURRENT @ CONTEXT !   ( set context vocabulary )
10   CREATE        ( build the dictionary header )
11   ' (TRACE)     ( find CFA of trace runtime routine )
12   CFA DUP @     ( and compile it )
13   HERE 2 - ! ,
14   ]             ( enter compilation mode )
15   ; IMMEDIATE                                          -->

Screen # 17
 0 ( TRACE function cont'd  )
 1 : ! ! ;  : (NUMBER) (NUMBER) ;  : + + ;  : * * ;  : +! +! ;
 2 : - - ;  : -FIND -FIND ;  : , , ;  : /MOD /MOD ;  : DUP DUP ;
 3 : @ @ ;  : BLANKS BLANKS ;  : BLOCK BLOCK ;  : C! C! ;
 4 : C@ C@ ;  : CMOVE CMOVE ;  : DROP DROP ;  : ENCLOSE ENCLOSE ;
 5 : FDOS FDOS ;  : MATCH MATCH ;  : AND AND ;  : OR OR ;
 6 : ROT ROT ;  : SP! SP! ;  : SWAP SWAP ;  : TYPE TYPE ;
 7 : OVER OVER ; : PICK PICK ;
 8                                                       -->
 9 : DROP          DROP ;  ( redefine with the trace enabled )
10 : TEST          50 0 DO
11                 I DUP DROP
12                 LOOP ;
13 TRON  TEST   TROF
14 ;S
15
```

Selected Publications

WE APOLOGIZE. The last issue of FORTH Dimensions stated that 5 newly added publications were included on the order form, but through a series of miscalculated events, they were not included. These publications are available from FIG and you will find them listed with their prices on the order form on the reverse side of this page.

A BIBLIOGRAPHY OF FORTH REFERENCES

Second Edition, September 1984
Thea Martin, Editor

The second edition of A Bibliography has over 1300 references to Forth related papers, books and articles from the US and abroad indexed by subject and author.

MASTERING FORTH

by Anita Anderson and Martin Tracy

A step-by-step tutorial to the high level, stack oriented Forth Computer language. Formerly titled FORTH TOOLS, this unique guide introduces you to each of the commands required by the Forth 83 International Standard.

THE JOURNAL OF FORTH APPLICATION AND RESEARCH, V. 2, #1

This issue of The Journal is devoted to hardware implementations of Forth. The focus is the "Forth engine...a board set or chip which will allow a board or board set to execute Forth as its primary programming language."

1984 ROCHESTER FORTH CONFERENCE

Real Time Systems.

A compilation of the papers and abstracts presented at the 1984 Rochester Forth Conference held June 6-9, 1984 at the University of Rochester in New York.

1983 FORML PROCEEDINGS

The proceedings from the fifth annual FORML conference which was held November 23 to 25, 1983 at the Asilomar Conference Grounds, Monterey, California. FORML (the Forth Modification Laboratory) is an informal forum for sharing and discussing new or unproven proposals intended to benefit Forth. This 350 page volume contains 42 papers or abstracts.

The Forth Interest Group Order Form has two newly added publications for this issue:

THE COMPLETE FORTH

by Alan Winfield.

This book is a complete guide to FORTH programming. The first half of the book introduces the language through examples and frequent comparison with BASIC. The later chapters delve into some of the more unusual capabilities of FORTH, many of which have no equivalent in other languages. The FORTH-79 standard dialect of FORTH is adopted throughout the book, although common departures from this standard are detailed as footnotes.

The book is intended for anyone who wishes to learn and use FORTH. Some familiarity with microcomputers and BASIC is assumed, but no prior knowledge of FORTH is required. The book should equally well serve as a useful reference of ideas and techniques for practicing FORTH programmers.

THE JOURNAL OF FORTH APPLICATIONS AND RESEARCH, V. 2, #2

This issue of The Journal is devoted to real time systems.

SPECIAL
FORML Proceedings, 1980-83

This issue we are offering a special price on Proceedings from the FORML Conferences of 1980 through 1983.

FORML was created in 1979 *by Kim Harris and John Spencer*. Two conferences were held in 1980, one in UK in January and one in November at the Asilomar Conference Center in Monterey, where the annual conference has been held traditionally ever since. The Conference Manager is Robert Reiling and the Conference Director is Kim Harris. FORML is now a part of the Forth Interest Group, and is not connected with the Forth Standards Team.

Many of the most active and influential people in the Forth community are regulars at FORML conferences. A number of ideas presented and discussed have been commonly adopted. Some have influenced the Forth Standard. FORML has helped determine the future of Forth.

The 5 volume set offered here at a special price as indicated on the order form on the reverse of this page, contains papers of the presentations of the conferences beginning with the November conference of 1980 through the 1983 conference. This Special Proceedings offer expires **April 1, 1985**.

FORTH INTEREST GROUP MAIL ORDER FORM

NAME _____ COMPANY _____

STREET _____ CITY _____

STATE/PROV. _____ ZIP _____ COUNTRY _____

TELEPHONE ( ) _____ DATE _____

	PRICES US/FOREIGN AIR	
Membership in the FORTH Interest Group & Volume 6 of FORTH Dimensions	$15/27	_____
Volume 1 FORTH Dimensions	15/18	_____
Volume 2 FORTH Dimensions	15/18	_____
Volume 3 FORTH Dimensions	15/18	_____
Volume 4 FORTH Dimensions	15/18	_____
Volume 5 FORTH Dimensions	15/18	_____
BOOKS ABOUT FORTH		
All About FORTH	$25/35	_____
Beginning FORTH	17/21	_____
Complete FORTH	16/20	_____
FORTH Encyclopedia	25/35	_____
FORTH Fundamentals, V. 1	16/20	_____
FORTH Fundamentals, V. 2	13/16	_____
Mastering Forth (Formerly Forth Tools)	18/22	_____
Starting FORTH (Soft Cover)	19/22	_____
Starting FORTH (Hard Cover)	23/28	_____
Thinking FORTH (Soft Cover)	16/20	_____
Thinking FORTH (Hard Cover)	23/28	_____
Threaded Interpretive Languages	23/28	_____
Understanding FORTH	3/5	_____
REFERENCE		
FORTH 83 Standard	$15/18	_____
FORTH 79 Standard	15/18	_____
Bibliography of Forth References, 2nd Ed.	15/18	_____
CONFERENCE PROCEEDINGS		
FORML Proceedings 1980	$25/35	_____
FORML Proceedings 1981 (2 V.)	40/45	_____
FORML Proceedings 1982	25/35	_____
FORML Proceedings 1983	25/35	_____
Rochester Proceedings 1981	25/35	_____
Rochester Proceedings 1982	25/35	_____
Rochester Proceedings 1983	25/35	_____
Rochester Proceedings 1984	25/35	_____

SPECIAL

FORML Proceedings 1980-83 (5 Volume Set)

☐ $ 75 U.S. and Canada

☐ $ 90 Foreign Surface

☐ $125 Foreign Air Exp. 4/1/85. _____

	PRICES US/FOREIGN AIR	
JOURNAL OF FORTH APPLICATIONS AND RESEARCH		
Journal of FORTH Research V. 1 #1	$15/18	_____
Journal of FORTH Research V. 1 #2	15/18	_____
Journal of FORTH Research V. 2 #1	15/18	_____
Journal of FORTH Research V. 2 #2	15/18	_____
REPRINTS		
Byte Reprints	$3.50/5	_____
Popular Computing 9/83	3.50/5	_____
Dr. Dobb's 9/82	3.50/5	_____
Dr. Dobb's 9/83	3.50/5	_____
Dr. Dobb's 9/84	3.50/5	_____
HISTORICAL DOCUMENTS		
Kitt Peak Primer	$25/35	_____
fig-FORTH Installation Manual	15/18	_____
ASSEMBLY LANGUAGE SOURCE LISTINGS		
1802	$15/18	_____
6502	15/18	_____
6800	15/18	_____
6809	15/18	_____
68000	15/18	_____
8080	15/18	_____
8086/88	15/18	_____
9900	15/18	_____
ALPHA MICRO	15/18	_____
Apple II	15/18	_____
ECLIPSE	15/18	_____
IBM/PC	15/18	_____
NOVA	15/18	_____
PACE	15/18	_____
PDP-11	15/18	_____
VAX	15/18	_____
Z80	15/18	_____
T-Shirt Size: _____	$10/12	_____
Poster (BYTE Cover)	3/5	_____
Handy Reference Card	FREE	_____

SUBTOTAL _____

CA Residents Add 6½% Sales Tax _____

TOTAL _____

☐ VISA ☐ Mastercard # _____ Expiration Date _____

$15 Minimum On VISA/Mastercard Orders.
All Prices Include Shipping.

Make Check or money order payable in US funds drawn on a US Bank to: FIG.
PAYMENT MUST ACCOMPANY ALL ORDERS (Including Purchase Orders).

OFFICE USE ONLY

By _____ Date _____ MO _____ TO _____ PU _____ Auth No _____

Shipped By _____ Date _____ Weight _____ UPS _____ USPS _____

Hold _____ Date _____ Weight _____ UPS _____ USPS _____

ORDER PHONE: (408) 277-0668

FORTH INTEREST GROUP • P.O. BOX 8231 • SAN JOSE, CA 95155

6/5

Quicksort and Swords


Wil Baden
Costa Mesa, California

Of the known algorithms for internal sorting on a single processor, the most efficient is C.A.R. Hoare's "Quicksort" first published in the British Computer Society's *Computer Journal* in 1962. This is also one of the simplest to implement in Forth. In BASIC and Fortran, it is so messy to program that, for most applications, a simpler, less efficient algorithm is used instead. Recursion and address pointers make it easy in Forth (and Pascal). With the use of vectored execution for the comparison function, it can be a general tool for many applications. Roughly parallel versions in Pascal and Forth are given here. The Pascal is intended to be explanatory of the Forth, although there are differences due to the nature of the two languages. The Forth is, by design, not factored into smaller words because speed is an important objective — every word will affect the efficiency of execution.

If you read the two versions in parallel, you will see that equivalent operations are done in the same sequence, although the logical structures are different. The difference is to take advantage of Forth's features and to avoid degenerate recursive calls.

In a practical Forth implementation, it is important that the following check be made:

ROT (m,j,i,n) 2OVER 2OVER − + > IF 2SWAP (i,n,m,j) THEN

This line verifies that the parameter and return stacks do not overflow. It could be left out and the program would almost always work; but Murphy's law says that someday in the far distant future, when you have long forgotten about possible causes, your system will crash if you do not have this line.

The performance is excellent for most applications. If it is not good enough, you can tune it in easy stages by gradually replacing high-level phrases with code definitions.

The normal use of **SORT** is as a tag sort: it is used to order an array of

```
: SWORDS  ( : list words in vocabulary in alphabetic sequence.)
    NAMES  ( addr,n)                        ( load names.)
    DUP .  ." names defined" CR
    ?DUP SORTED ID<                         ( sort names.)
    .NAMES  ;                               ( list names.)
```

Figure One

```
: ID<  ( nfa1,nfa2 -- f )       ( compare FIG-Forth style namefields )
    2DUP COUNT 31 ( width) AND OVER + SWAP  ( nfa1,nfa2,nfa1+w,nfa2 )
    DO  1+  DUP C@  I C@ -
        IF  C@ 127 ( char) AND I C@ 127 ( char) AND -  ( ...,c1-c2 )  DUP
            IF  0< SWAP ROT DROP  ( f,nfa2 )  THEN
            0= LEAVE
        THEN
    LOOP  ( nfa1,nfa2,nonzero -or- f,0 )
    IF  C@ 31 ( width) AND SWAP C@ 31 ( width) AND > THEN ;
```

Figure Two

```
: NAMES  ( -- a,n : build array of nfa's from dictionary.)
   PAD C/L + DUP  ( a1,a2)
   CONTEXT @ ( a1,a2,nfa of last word in "CONTEXT" )
   BEGIN @ -DUP WHILE DUP >R OVER ! 2+ R> PFA LFA REPEAT
   OVER - 2 /  ( convert a1,a2 to a1,n )  ;
```

Figure Three

```
: .NAMES  ( a,n : list n words located by addresses starting at a )
   0 DO  DUP @ ID.  2 SPACES  2+ LOOP DROP  ;
```

Figure Four

```
( FIG-Forth )
: CR  ( -- )  CR  0 OUT !  ;   ( you may already do this.)
: ?LINE  ( n : newline if not room for n characters.)
    OUT @ + RMARGIN @ > IF CR  THEN ;

( F83 )
: ?LINE  ( n : newline if not room for n characters.)
    #OUT @ + RMARGIN @ > IF CR  THEN ;
```

Figure Five

```
( FIG-Forth )
: ?TAB  ( n : tab to next multiple of n if not already there.)
   OUT @ OVER MOD -DUP IF - SPACES  ELSE DROP  THEN ;

( F83 )
: ?TAB  ( n : tab to next multiple of n if not already there.)
   #OUT @ NEGATE SWAP MOD SPACES  ;
```

Figure Six

```
: TAB.ID  ( nfa,n : display namefield in n-wide column.)
   DUP ?TAB
   OVER C@ 31 ( width) AND 1+ RMARGIN @ ROT MOD + ?LINE
   .ID  ;
```

Figure Seven

```
: .NAMES  ( a,n : list n names located by starting address a )
   0 DO DUP NAME-TAB TAB.ID  2+ LOOP DROP ;
```

Figure Eight

```
: NAMES  ( -- a,n : build array of nfa's from dictionary.)
   CR CONTEXT @ BODY> >NAME .ID  ( print vocabulary name )
   PAD C/L + DUP  ( a1,a2 )
   #THREADS 0
   DO  CONTEXT @ I 2* +
       BEGIN @ ?DUP WHILE DUP >R L>NAME OVER !  2+ R> REPEAT
   LOOP
   OVER - 2/ ;
```

Figure Nine

```
414 NAMES DEFINED
!    "    #    #>    #BUFFERS    #OUT    #S    #THREADS    #TIB    #VOCS
R    %DIR    '    'WORD    (    ()    (.)    (CONVEY)    (COPY)    (D.)
.)    *    */    */MOD    +    +!    +LOOP    ,    ."    -    -->    -LINE
 -TRAILING    .    ."    .(    .DIR    .ID    .R    .S    .TO    /
NG    0    0.    0<    0<>    0=    0>    1    1+    1-    2    2!    2*
 2/    2@    2CONSTANT    2DROP    2DUP    2OVER    2ROT    2SWAP
 3    3DUP    4DUP    :    ;CODE    <#    <>    <RETURN>    =
 >IN    >LINK    ?BRANCH    ?CONVEY    ?CSP    ?DO    ?DUP
 ?ERROR    ?EXIT    ?PAIRS    ?STACK    ?TERMINAL    ?TRANSMIT
    RECONVEY    RECURSE    NEGATE    PAIRS
    ROLL    ROOT    ROT    RUN    RX    RX?
SE-SYSTEM    SCAN    SCR    SCR'    SCRN    SET    SIGN    SKIP
CE    SOURCE0    SOURCESTREAM    SP@    SPACE    SPACES    SPAN
US    STREAM    STREAM0    SUBJECT    SWAP    TAB    TAB.ID    TERMINAL
NALO    TEXT    THEN    THEN    THENIF    THENIF    THRU    TIB    TONE
 TUCK    TX    TYPE    U.    U.R    U2/    U<    UM*    UM/MOD    UNTIL
 UPDATE    UPDATED?    VARIABLE    VLIN    VOC-LINK    VOCABULARY
 WHERE    WHILE    WHILE    WITHIN    WORD    WORDS    XOR    [    ['],
E]    \    ]
```

Sample of output from the Forth-83 version.
Figure Ten

```
FORTH 414 NAMES DEFINED
!    "    #    #>    #BUFFERS    #OUT    #S
#TIB    #VOCS    $NUMBER    %DIR    '    'WORD    (
(.)    (CONVEY)    (COPY)    (D.)    (S    (U.)    *
SP@    +    +!    +LOOP    ,    ."    -
SOURCE    -ROT    -TRAILING    .    .R    .S    .(
SUBJECT    THEN    THEN    .S    .TO    THRU    /STRING
THEN    TIB    TONE    0<    0=    U.R    U2/
TUCK    TX    U.    U<    UM*    UPDATE    UPDATED?
UM/MOD    UNTIL    UNTIL    WARNING    WHERE    WHILE    WITHIN
VOCABULARY    VLIN    VOC-LINK    WORD    WORDS    XOR    [    [']
[COMPILE]    \
```

Sample of output from the fig-FORTH version.
Figure Eleven

locators (i.e., addresses, pointers, etc.) of records or data. The user first puts into the array the locators of the data to be sorted. The user also assigns to the deferred word **PRECEDES** the address of a word which will compare the keys of the data specified by two locators. If a file is to be sorted, this word should read records as needed. This word should return nonzero for true when the data pointed to by the first locator should precede the data pointed to by the second locator. **SORT** requires on the stack the address of the first cell of the array and the number of locators in the array.

To make **SORT** easy to use, the word **SORTED** is provided. It is used in the form:

array number **SORTED** relation

where "relation" is the address of the word which will compare keys. This will sort "number" locators beginning at address "array" according to "relation." (See **SWORDS** in figure one.) The usual sequence of operations is to load an array with locators, order the array, and then process (often by printing) the data as located in the ordered array.

As a useful application, we give an alphabetically ordered list of the words in your dictionary. We will show a simple version in fig-FORTH and a better one in Forth-83. The fig-FORTH user should be able to figure out how to make the changes for the upgrade. The definition of the end-user word is the same in both dialects. (See definition of **SWORDS** in figure one.)

The word to compare name fields is also the same in both dialects. It is made complicated (and slow) by the fact that we need to take care of the high-order bit in the last byte of a name. We compare the two words byte-for-byte for their minimum length, and then check a mismatching byte (see **ID?** in figure two). If it is the same character, we make the shorter word precede the longer one. In this application, the slowness is not very important, since the order of most words will be decided by comparing only a few characters.

In both versions, we assume there is enough memory available above **PAD**

```
You may require:
( Forth-83 )
: RECURSE  ( ? -- ? )  LAST @ NAME> ,  ;  IMMEDIATE
: CRASH  ( -- )   TRUE ABORT" undefined execution vector."  ;
: DEFER  ( -- )   CREATE  ['] CRASH ,  DOES>  ( -- )  @ EXECUTE ;
: (IS)   ( a -- ) R> DUP 2+ >R  @ >BODY !  ;
: IS     ( a -- ) STATE @ IF COMPILE (IS)  ELSE ' >BODY !  THEN ; IMMEDIATE
( FIG-Forth )
: RECURSE  ( ? -- ? )  LATEST PFA CFA ,  ;  IMMEDIATE
( Change ABORT" ..." to IF ." ..." ABORT THEN  in SORT and CRASH )
: DEFER  ( -- )  <BUILDS ' CRASH CFA , DOES>  ( -- ) @ EXECUTE ;
: (IS)   ( pfa -- ) CFA R> DUP 2+ >R  @ ( cfa) 4 + ! ;
: IS     ( pfa -- ) STATE @
        IF COMPILE (IS) ELSE CFA [COMPILE] ' ( pfa) 2+ ! THEN ; IMMEDIATE
: (SORTED) R> DUP 2+ >R @ ( cfa) 2+ ( pfa) IS PRECEDES  SORT  ;

   We hope that you have CODE definitions of "2DUP", "2SWAP", "2DROP", and
"2OVER".  Performance will definitely suffer if you do not have them.
If you do not have them, you can improve performance a little by replacing
some words with their high level definitions, e.g., replace "2DUP" with
"OVER OVER".
```

Figure Twelve

```
{ Hoare's QUICKSORT -- parallel demonstration in Pascal and Forth }

function precedes (x, y : position) : boolean ;
begin
    precedes := (key[x] < key[y]) ;    { compare key at positions }
end ;

procedure quick (m, n : address) ;     { order tag[m]..tag[n] }
var
    i, j        : address ;
    pivot, temp : position ;
begin
    if (m < n) then begin
        pivot := tag[m + ((n - m) div 2)] ;
        i := m ;  j := n ;
        repeat                              { partition }
            while (precedes(tag[i], pivot)) do i := succ(i) ;
            while (precedes(pivot, tag[j])) do j := pred(j) ;
            if (i <= j) then begin
                temp := tag[i] ; tag[i] := tag[j] ; tag[j] := temp ;
                i := succ(i) ;  j := pred(j)
            end
        until (i > j) ;
        if (j-m < n-i) then begin quick (m,j) ; quick (i,n) end
        else                begin quick (i,n) ; quick (m,j) end
    end
end ;

procedure sort (a : address ; n : integer) ;
begin
    quick (a, a + (n-1))
end ;
```

Quicksort in Pascal

Figure Thirteen

FORTH IS NOW VERY.FAST!

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

DEVELOP YOUR APPLICATIONS IN A TOTAL FORTH ENVIRONMENT.

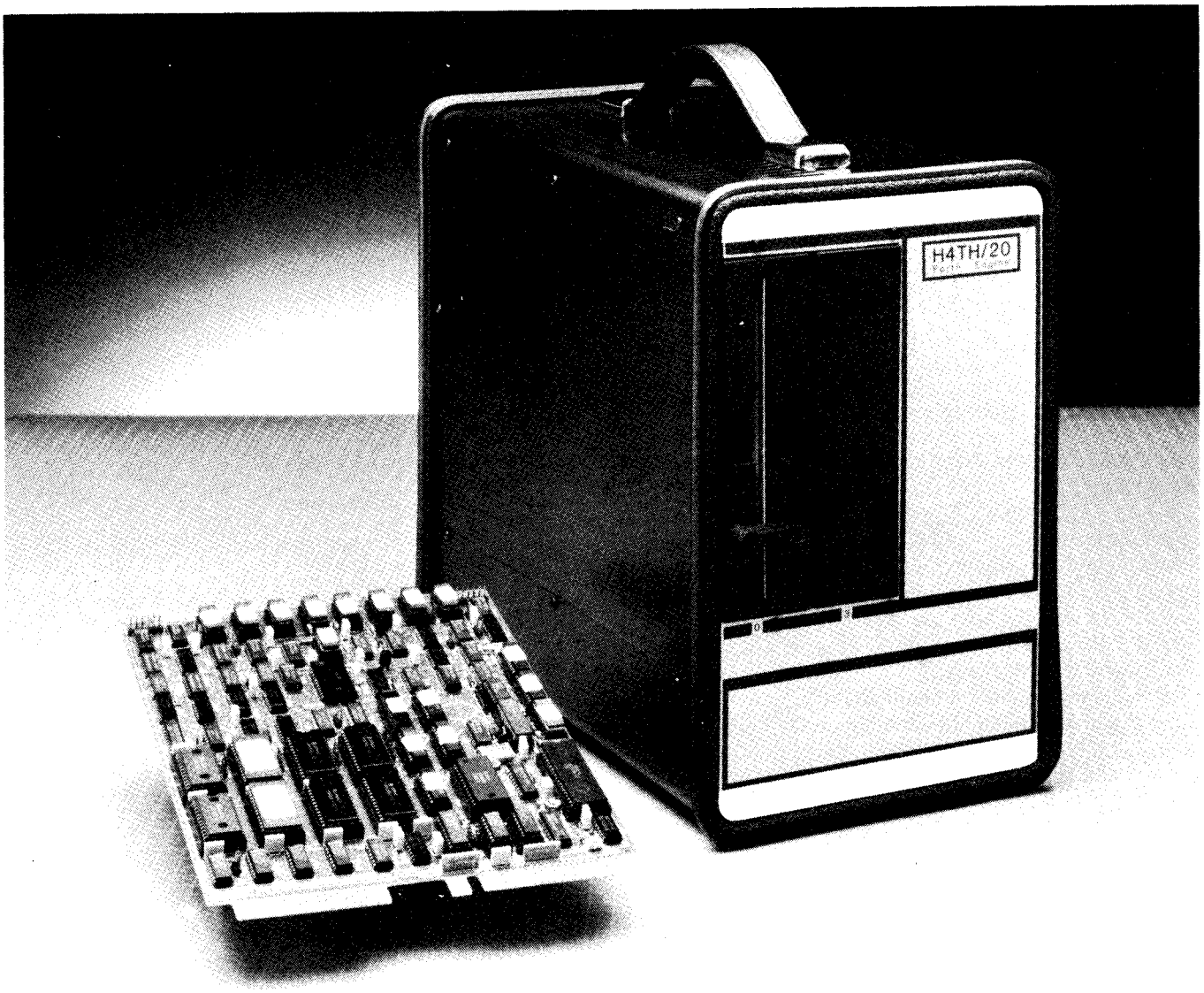
MICROPROGRAMMED BIT SLICE FORTH ENGINE
.Microcoded forth kernel
.Microcoded forth primitives
.Multi-level task switching architecture for real time applications
.Optional writable control store

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

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

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

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



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

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

for the address of each word plus a little more. We choose as the address of the start of the array **PAD** + **C/L** (see **NAMES** in figure three).

It is now easy to list the words. The number of words and the address of the array are on the stack. We use the fig-FORTH **ID.** to list each word (see **.NAMES** in figure four). The resulting list will be in sequence, but some words will cross over two lines. Let's fix that by printing the words in columns. I like columns nine characters in width for name fields, though you may prefer a greater width. If a name is too long for one column, it will use the next column:

9 CONSTANT NAME-TAB (column width for name fields)

We also need a variable for the maximum number of characters in a line. This is a variable so you can change it for different output devices. We call it **RMARGIN** and it is already defined in Laxen and Perry's F83. Seventy-two is a good value for most systems.

Now we want a way to avoid running over the current line. fig-FORTH and F83 are similar in this regard (see the two definitions of **?LINE** in figure five). Now you should have recognized that **#OUT** is the new name for **OUT**. You will also have guessed that a useful feature of **CR** in many systems has been made part of F83, namely **#OUT** is reset to zero.

Another useful feature of fig-FORTH has been incorporated in F83: **EMIT** and other words which emit characters increment **#OUT**. We are going to use this to tab to the next column if we are not already there. (See the definitions of **?TAB** in figure six; the F83 version takes advantage of floored division.)

That does it. Now we can display names in columns. We will use the new name **.ID** of **ID.** from F83. (See **TAB.ID** in figure seven.) It is not necessary for **RMARGIN** @ to be a multiple of **NAME-TAB**. This gives us a new definition for **.NAMES** in fig-FORTH and in F83 (see **.NAMES** in figure eight).

The F83 definition of **NAMES** is like fig-FORTH but is done for each thread in the multi-thread dictionary. As an extra feature, we print the name of the vocabulary (see **NAMES** in figure nine). With this code, sorting 500 names takes twenty-five seconds on a 1Mz 6502, or fourteen seconds on a 5Mz Z80.

These are words you may require to run the accompanying code. We hope you have **CODE** definitions of **2DUP**, **2SWAP**, **2DROP** and **2OVER**. Performance will definitely suffer if you do not have them. If not, you can improve performance a little by replacing some words with their high-level definitions (e.g., replace **2DUP** with **OVER OVER**).

```
( C.A.R.HOARE'S QUICKSORT.                          WWB/WWB 811012 )
( Note: cellsize is 2 bytes. )
DEFER PRECEDES    ( a1,a2 -- f )                    ( vectored executon )

: QUICK  ( a[m],a[n] -- )                           ( partition a[m]..a[n] )
    2DUP  OVER - 2/ -2 AND + @ >R          ( take middle value as pivot )
    2DUP SWAP
    BEGIN          ( m,n,j,i )                      ( m <= i <= j <= n )
        BEGIN  DUP @  R@  PRECEDES  WHILE  2+  REPEAT  SWAP
        BEGIN  R@ OVER @  PRECEDES  WHILE  2-  REPEAT  SWAP
        2DUP U< NOT
        IF  2DUP 2DUP @ >R  @ SWAP !  R> SWAP !  SWAP 2- SWAP 2+ THEN
        2DUP U<
    UNTIL  R> DROP  ( m,n,j,i )                     ( m <= j < i <= n )
    ROT  ( m,j,i,n )  2OVER 2OVER  - + > IF  2SWAP  ( i,n,m,j )  THEN
    2DUP U< IF  RECURSE  ELSE  2DROP  THEN      ( shorter part )
    2DUP U< IF  RECURSE  ELSE  2DROP  THEN      ( longer part )  ;

: SORT  ( a,n -- )              ( order a[0]..a[n-1] by 'PRECEDES'. )
    ?DUP 0= ABORT" nothing to sort."
    1- 2* OVER +  ( a[0],a[n-1] )  QUICK ;

: (SORTED)  R> DUP 2+ >R  @ IS PRECEDES  SORT  ;
: SORTED  ( a,n --<relation> )  ( order a[0]..a[n-1] by <relation> )
    STATE @
    IF  COMPILE (SORTED)
    ELSE  ' IS PRECEDES  SORT  THEN  ;  IMMEDIATE
```

Quicksort in Forth
Figure Fourteen

Why Forth Isn't Slow

Adin Tevet
Haifa, Israel

When a friend explained Forth to me a couple of years ago, I dismissed the idea as impractical. I was sure that any language in which a word executed words which in turn executed other words, etc., just had to be slow. Now that I am happily coding in Forth, I have begun to think about why Forth isn't as slow as I had imagined it to be.

My mental image had been something like that of figure one-a, which suggests a sequence of word executions that might be rather long before finally getting down to machine code. You do get a sequence which *could* be long if each word executes just one other word (as illustrated in figure one-b) but this seems to be rare. In a scan of fifty colon definitions published in *Forth Dimensions*, I found no one-word definitions.

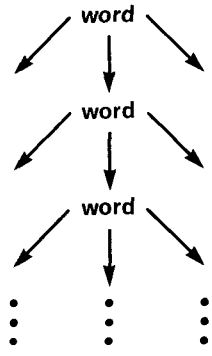


Figure One-a

Figure One-b

The next worst case is when each word executes just two other words. The picture then looks like figure two. Here P stands for the primary words coded in assembly language and S stands for secondary words, which execute previously defined words. This picture suggests that many secondaries are needed to execute the program, but this isn't so. In any binary tree, if #P is the number of primaries (terminal nodes) and #S is the number of secondaries (intermediate nodes), then #S = #P − 1.

This is easy to prove. The simplest case is a tree consisting of just one primary. For this tree, we have

$$\#S = 0 = 1 - 1 = \#P - 1$$

Any other binary tree can then be grown from this simplest tree by repeatedly replacing a terminal node P with S. Each replacement just adds one to #S and one to #P, so the equation #S = #P − 1 remains true. This formula says that even in the extreme case where every secondary word in a program executes just two words, there will be only one secondary executed for each primary executed. Thus, threaded code only doubles the number of words that are executed. This is already quite a bit better than I had originally imagined.

Analyzing a little deeper, suppose that all secondary words execute exactly N words. Then,

$$\#S = (\#P - 1)/(N - 1)$$

This is also easy to prove: each primary is replaced by one secondary and N primaries in each step as a tree grows. The same formula also holds in the completely general case where secondary words execute *any* number of previously defined words. N is then the average number of words executed by the secondaries in the tree. (A proof of this is similar to the other proofs; I will supply one on request.)

Now, if E^P is the average execution time for a primary and E^S is the average execution time for a secondary, then total execution time can be derived as in figure three. The expression $\#P \times E^P$ is the execution time for the primaries and

$$(1 + E^S/(E^P \times (N - 1)))$$

is the factor representing the overhead cost incurred by the secondaries. A factor of one would indicate no overhead at all, a factor of two would indicate an overhead of 100% over execution time for the primaries alone. If the tree were flattened down to a single secondary that executes all of the primaries in the program, then the overhead factor would be one.

Just to get a quick estimate of the overhead for secondaries in a real system (fig-FORTH on an LSI 11/23), I counted assembly language instructions in the implementation of several words, ignoring the differing cycle times of the instructions. I found a total of seven instructions in the implementations of **DOCOL** and **SEMIS** for run-time execution of colon defini-

tions, the most common form of secondaries. Over a group of twenty primaries, I found an average of 4.4 instructions per primary (including **NEXT**). This gives a very rough approximation of $E^S / E^P = 7 / 4.4$. I assumed that N could be estimated by counting words in a large number of colon definitions. The number of words executed at run time by a secondary is increased by a backward branch (as in **LOOP**) and decreased by a forward branch (as in **IF**), so I ignored colon definitions containing any branching. From fifty such colon definitions found in *Forth Dimensions*, I counted an average of N = 6.5 words. Note that N also depends on programming style, with shorter Forth definitions giving smaller values of N. These values provided an estimate of overhead

$$= 1 + E^S/E^P \times (N - 1)$$

$$= 1 + 7/(4.4 \times (6.5 - 1))$$

$$= 1.29$$

which represents a 29% increase in execution time due to the execution of secondaries. Because of the many simplifying assumptions used in arriving at this figure, it should not be taken too literally. But even as a very rough approximation, it shows that the execution cost of secondary words is not high.

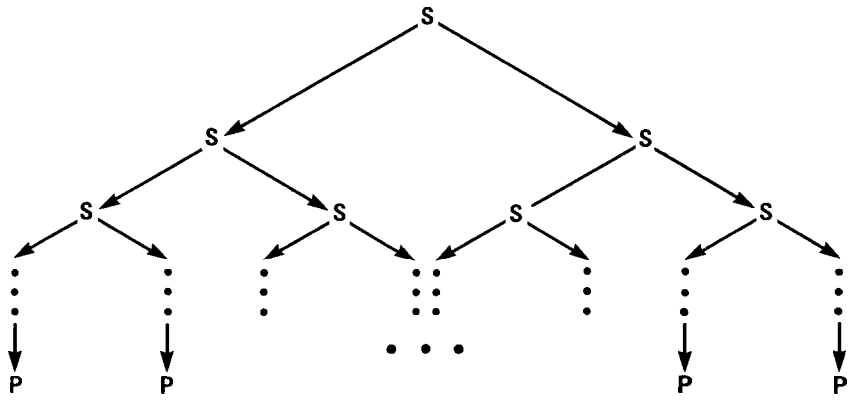


Figure Two

SOFTNET

High-Level Packet Communication

Jens Zander
Robert Forchheimer
Linkoping, Sweden

SOFTNET is a packet-radio concept under development in Sweden. The network is distributed and all nodes are programmable via the network during normal operation. This concept represents an unconventional approach to the protocol issue and offers elegant solutions to the higher-level communication problems. This paper gives a programming model of the network, along with some examples.

Introduction

The SOFTNET approach was conceived in 1980 and was discussed among Swedish radio amateurs. The discussion led to a proposal for an experimental network in the 432 MHz band utilizing bit rates up to 100 Kbps. During 1981, this draft was presented to the Swedish Telecommunication Administration. The administration responded in a positive way, giving the packet radio group at Linkoping University virtually free hands. This group, consisting of six people, is currently involved in developing prototype nodes and basic software for the network.

The main concept behind SOFTNET is that all packets are considered to be programs of a *network language*. These programs are interpreted in the nodes as soon as they arrive. Nodes can be programmed by any number of users simultaneously without unwanted interaction. This approach makes it possible for a user to define his own high-level services like datagrams, virtual calls, file transfers and mailboxes. The concept also allows changes at lower levels during operation, permitting redefinition of LINK-level/Access protocols. A detailed description of these ideas can be found in references one, two, three and five.

Node Model

In a SOFTNET node, an incoming packet that has passed the link level is given to the node computer for interpretation. Here a standardized set of instructions are available. The kernel of this set is simply a Forth interpreter to which has been added functions that control the node hardware. Thus, any user may execute his own Forth program in any of the nodes that he can reach. This way, he is able to instruct another node to either deliver the packet to the owner of the node or to retransmit it so that the node merely acts as a repeater. Forth allows the creation of private directories, so the user may also store programs in remote nodes. These programs may either "wake up" upon the arrival of a packet from the user or upon an internal signal (e.g., the real-time clock) produced by the remote node itself. Describing the node, thus, reduces to describing a *programming model*. In the Forth case, this is done by simply listing all the available functions or "words".[4] Figure one summarizes the packet format from the user's point of view.

In fact, the link-level protocol has been added to the Forth kernel so that also the link-level information is handled by a Forth interpreter. This permits on-line reprogramming and extension of the link protocol such as new versions of HDLS, access algorithms, etc. Thus, from the first byte to the last, a SOFTNET packet is simply a set of Forth statements. From a practical point of view, it is a good idea to conceptually keep instructions at the link level apart from the higher-level programs, since changes at the link level have to be coordinated among the users.

The Node:
Multi-user, Multi-tasking

Processing at the link level requires real-time performance, while higher-level tasks are less time constrained. On the other hand, the link processor serves one packet at a time sequentially, while higher-level tasks may run concurrently. Also, the programming activities of one user should not influence any other. Thus, a SOFTNET node must be able to support parallel tasks besides being able to keep apart the current users of the node. For the prototype implementation, our choice was a dual processor (6809) system. One of the processors is solely devoted to link-level processing. The second processor contains a multi-tasking Forth interpreter and is shared among the users. A special task — the owner process — interfaces the node to the owner's equipment, which can be anything from a dumb terminal to a full-grown computer system. In the latter case, the dual processor Forth system is simply considered a modem between the owner's system and the network.

Figure One

Node Programming Example

Consider the simple network given in figure two. Four nodes are connected by two-way radio paths as indicated by the lines between the nodes. Suppose a user is located at node A and has the specific task to deliver a large number of packets to node E (i.e., he wants to establish a "virtual" call to E). This can be done in at least two ways. The simplest thing to do is just to add a retransmit command to all packets as shown in figure three-a. The command % takes the next symbol as an (one-hop) address and transmits the rest of the packet to that address. This goes on, dropping one address each time, until the remaining packet reaches node E, where the data portion is transferred to the owner of the node. This procedure may, however, consume valuable packet space, especially when many intermediate nodes are used. We can instead make use of the programming features and instruct the intermediate nodes B and C just to pass along the packet to the next node in line. This can be done as in figure three-b. Here we define a new function named, say, **VCE** in the intermediate

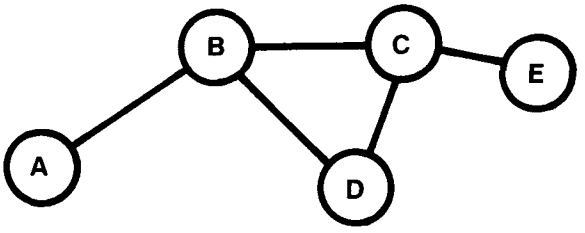


Figure Two

```
% B  % C  %E  OWNER  <data>          a)

% B  : VCE ."VCE" % C ;

                                      b)
% B  % C  : VCE ."OWNER" % E ;

: VCE ."VCE" % B ;

                                      c)
VCE <data>
```

Figure Three

nodes and in our own node. A new definition is made Forth-style, starting with a colon and ending with a semicolon. The effect of executing **VCE** is to pass on the rest of the packet to the next node in line. Also, the function places a copy of its name first in the packet for repeated execution in the succeeding nodes.

Project Status

Since the advent of the project at Linkoping University, a rapidly growing number of interested radio amateurs has joined the discussions. A SOFTNET User Group (SUG) is being formed as a subgroup of AMSAT-SM. To date, this group has received about one hundred applications for membership.

Hardware development has made considerable progress. The node computer board is under production and a first shipment of fifty kits was delivered in February 1984. Also, the PC layout for the link computer board is complete. The packet radio utilizes a duo-binary direct FSK modulation scheme with favorable bandwidth properties. Transmission is synchronous and MFM coding is used to recover clock information. Due to problems in the design of the radio, testing of the digital hardware and software had to be done on a cable-bound local network. A system with up to four nodes has so far been successfully demonstrated and has provided useful results for further software development.

Conclusions

The SOFTNET concept, with its fully programmable nodes, will give the user opportunity not only to communicate, but to conduct experiments in network architecture and network protocols. The concept is applicable to all kinds of communication networks. An implementation using a local network cable has been successfully tested and a UHF radio broadcast network is under construction.[2]

References

1. Persson, I. and Forchheimer, R.: "Design Considerations of a Distributed Packet Radio Network Using the Amateur Radio Bands," Internal Report, LiTH-ISY-0408, May 1980.

2. Zander, J. and Forchheimer, R.: "Preliminary Specifications for a Distributed Packet Radio Network for Computer and Radio Amateurs," Internal Report, LiTH-ISY-0424, January 1980.

3. Zander, J. and Forchheimer, R.: "Softnet Packet Radio in Sweden," ARRL Amateur Radio Computing Networking Conference, Gaithesburg, Maryland, October 1981.

4. Forchheimer, R. and Zander, J.: *Softnet User's Manual*, Linkoping 1983.

5. Qvigstad, F. and Matts, S.: "Construction of a Packet Radio Node Computer," Internal Report, LiTH-ISY-0491, December 1981.

In Review

FORML 1984 Asilomar Conference

Friday, November 25, 1984 — clouds were moving in from the Pacific, contrasting against the brilliant sky. Traffic was backed up the freeway for miles due to the opening of the new Monterey Bay Aquarium, causing a half-hour delay in the start of the program. As always, the wind-swept pines and wildlife at the Asilomar Conference Center provided a quiet but intense environment for the annual FORML conference.

Attendance at the meeting differs from that of the FIG convention (held the previous weekend in San Jose) in that registration is limited to a small number; the technical level of papers is generally quite high; and criticism, debate and other forms of audience participation are an integral part of the presentations. Introductions showed a sophisticated professional audience and a vigorous representation of Forth experts. The one hundred FIG members in attendance had traveled from Taiwan (ROC), England, West Germany, Switzerland and many parts of the United States.

Sam Suan Chen attended as a representative of the Taiwan FIG Chapter. In addition to his own paper "In-Word Parameter Words," he accepted an invitation to serve as chairman of a session on interfaces.

The first speaker in that session was Charles Moore. In addition to providing a progress report on the Forth chip, he told how he started its design totally ignorant of how to document on paper the design of a computer, a non-trivial problem. He finally realized he could use Forth to document that design. Code in his paper shows how that can be done.

His method is one of describing the logic to the computer, incorporating the equations (which he had put into postfix form). His paper describes the Forth CAD system which evolved into a simulation of the correctness of logic levels and signal propogation timing.

Charles Buckley next discussed a "Graduate Course in Smart Product Design." In 1977, Professor L.J. Leifer started the course in the Design Division of the Mechanical Engineering department at Stanford University. It has been found there that the design projects go much more quickly by eliminating dependence on outside programmers. The course is an intense, one-year MS program which condenses the equivalent of five quarters' work in two quarters.

The course covers software to such an extent that the students have full access to the machine's abilities. In the past they had to cut the syllabus in half, or else work so fast as to set students' heads spinning. But for the past two years they have used Forth and find it a phenomenal improvement.

The Forth environment can replace entire subsystems, considerably easing the learning curve of gaining familiarity with the available tools. The ease of using Forth with assembly language was another major advantage. Its simplicity and consistency make it quick to learn, and a noticeable improvement in the quality of student projects has become evident.

John James and William Volk both spoke of subjects which could improve the portability of Forth programs. James' "Towards Standardized Modem Words" addresses the problem of enabling Forth routines to communicate easily with various kinds of modem hardware. Another goal was to insulate the end user from the need to know anything about parity, word length, stop bits, etc. Volk's "Portable Graphics Wordset" speaks of the problem of running a portable program on different systems in such a way that the screen will look the same. Side benefits of doing this are windows and viewports.

The third session consisted of only two papers, but they were central to the "Expert Systems" theme of this year's conference. Dana Redington's contribution concerned the FORTES Polysomnographer in use at the Stanford Sleep Lab. He prefaced his presentation with the explanation that in the field of artificial intelligence, there are cognitive science (how the mind works, science-oriented) and knowledge engineering (trying to create a machine to mimic the mind).

There are two major approaches to knowledge engineering, or expert systems. The traditional approach requires that the information must be modular in form; and one must understand the difference between software and knowledgeware (work toward knowledge engines and building knowledge bases).

An expert system mimics a human expert. There are two classes: consultants (non-real-time) and operators (real-time). The components of either are a knowledge-slate, knowledge-engines, and knowledge-bases, each with various specific sub-functions. For this particular application, the lab conditions for sleep research mandated a small, quiet computer and budget required it come "off the shelf." Forth was installed and extended to become an expert system. Forth is described as a very good language for these purposes, particularly because LISP and PROLOG don't work fast or well on microcomputers.

Jack Park then shared some of his experience working with expert systems programmed in Forth. He pointed out that any kind of program that encapsulates information is, technically, an expert system. The point of departure for genuine examples, though, is that they get away from the programmer controlling the program flow and let the information itself direct the program flow.

In the general "Forth Philosophy" part of the program, Tom Hand provided an anecdote interesting to the many teachers and schools evaluating and developing Forth curricula. He teaches an elective course at the Florida Institute of Technology in which students are required to develop a Forth-related thesis. In another course, on operating systems, learning Forth was optional; at first, most students opted to stay away from it, but later it became evident that those who had Forth were completing their projects much more quickly and with fewer problems. As the course progresses, more students turn to Forth.

In the same afternoon, Ron Braithwaite spoke about "Object-Oriented

Programming," a subject which inspired a good deal of audience participation. There were varied opinions about whether data typing was a good idea (or even needed). The simple definition of "objected-oriented" was offered by Wil Baden in two words: **CREATE DOES**>. Food for thought!

By this time, the audience was primed for a heady debate about concepts so broad as to have little direct impact on the day-to-day routine of programmers and systems developers. Andreas Goppold's paper bore the sub-title, "An essay into the timeframe network of instrumental language." Instrumental language was defined as a class of languages which allows man to interface with what, by the use of that language, came to be called "objective reality." Hence his interest in the relations and processes involved, rather than in the data itself. Goppold traced the evolution and impact of spoken and written language in the past five thousand years and made some projections about communication in the coming millenia.

Don Colburn's "Direct Connect" presentation was a discussion of man-to-man and man-to-machine communication. He used Blissymbolics to demonstrate iconic transmission of information.

At this point, the audience fairly burst into active participation as hands were raised around the room. There was a question about whether icons lead us back five thousand years to the hieroglyphic state, or whether they are a step forward into universal meaning. Several participants reminded us of the large role that aural input plays in our communication. However, most agreed that visual input is vital. John James suggested as an interesting experiment the development of a computer with which we communicate only in sounds (words or otherwise), and which responds to us in like manner. The goal of the experiment would be to program the computer to do a useful task. As this session ended, individuals continued debating various points in small groups.

A popular part of the FORML conference was that devoted to impromptu lectures. A scattering of topics was presented, each given five minutes or less. During that period, Klaus Schleisiek spoke on the recurring theme of the importance of file systems. He feels that while many of the typical systems today are not intelligently designed, this represents a challenge which Forth can help to confront.

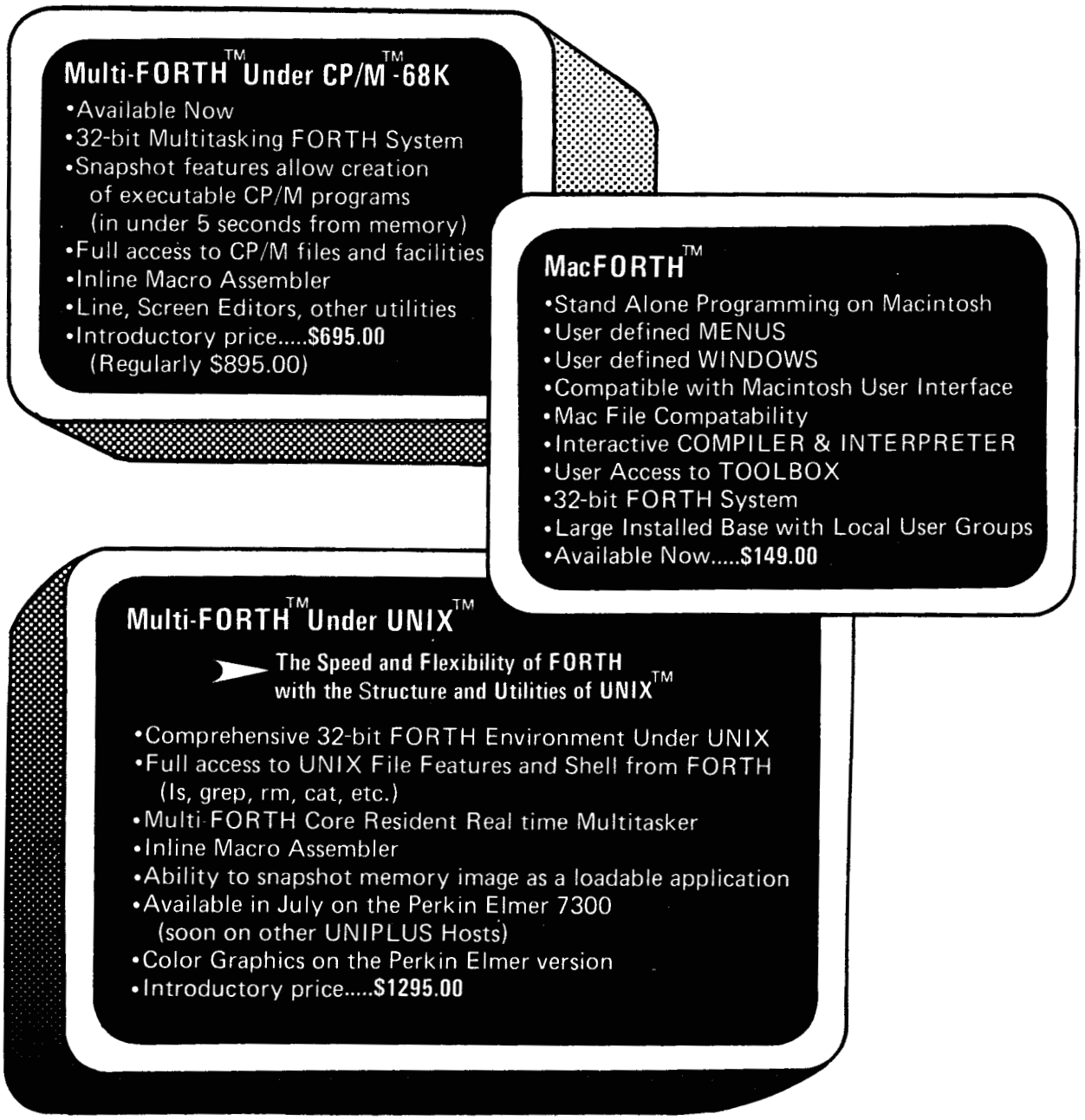
John Irwin uses PC-DOS from Forth to access the PC file system. He uses the word **R/W** to process a flag to call system functions, an approach which minimizes the changes needed within Forth to implement files.

Guy Grotke spoke on his simple metacompiler. The premise was that if one is willing to accept certain limitations, one can write so simple a metacompiler that it is possible to use it in teaching situations.

Zafar Essak, a physician, spoke of the medical applications of Forth. He underlined the importance of considering how the system we design will affect its users. His way of designing medical office management software: any event in the office can be grouped under the general heading of "client encountering professional." Then each type of encounter (scheduling, problems and medications, investigation, consulting, etc.) can be assigned a logical relationship.

Wil Baden presented his simple rules for spacing between Forth words for improved readability. In his method, words that put something on the stack are called nouns, those that consume items are verbs, modifiers are adjectives, and words leaving the stack unchanged are likened to interjections. Nouns followed by nouns, adjectives or verbs have only one space after them. An adjective followed by another adjective has only one space; adjectives followed by a noun or by a verb, use two spaces. Interjections are followed by two spaces except when the following word in another interjection. Verbs are always followed by a double space. And if that verbal description seems confusing, a simple chart using abbreviations is shown:

Introducing 3 New 68000 FORTH Systems



68000 FORTH Systems also available on HP Series 200 and Motorola VME10

For more information contact

CREATIVE SOLUTIONS

4701 Randolph Rd. Ste.12 Rockville, Maryland 20852
(301)984-0262

UNIX is a registered trademark of AT&T • CP/M is a registered trademark of Digital Research

	n	a	v	i
n	−	−	−	+
a	+	−	+	+
v	+	+	+	+
i	+	+	+	−

Leo Brodie addressed those who teach and write about Forth. He finds the pfa, cfa, etc., ambiguous and has adopted Kim Harris' revised notation. Because the contents of the code field can be an address, the address of the code field itself is now written acf, and for the same reason the address of the parameter field is now referred to as the apf.

Chuck Moore spoke on "hacking Forth," fearing that the fun aspect of programming is being lost. He related that hackers at a recent conference spoke more on how one cannot make money by giving away software than about how important it is that information (if not the software itself) be freely available and shared.

Mike Perry then made several points. The progress report on F83: very few extensions have been done and little has been shared (as was originally hoped); F83 has been ported to almost no new machines, and those which have been done are no longer directly related or are somehow reduced in scope. In one case, a person ported Forth to the Mac but wouldn't show the code to Perry because it had become "proprietary." From there Mike went on to discuss the public domain, and elicited a strong audience response in support of it. He especially requested vendors to look on hackers' efforts (e.g., public-domain software) as contributions to the state of the art and not as threatening in the way of competition.

The final topic for formal papers was "Local Variables," a subject which prompted Conference Director Kim Harris to remark on the proliferation of the ideas presented at FORML meetings. After thorough discussion of an idea, regardless of its degree of acceptance or rejection at the meeting, it either comes quickly into widespread use or is never heard of again. Local variables have raised some interest among FIG members lately; watching their progress or stagnation in the Forth community should be exemplary of this process.

Bob La Quey uses a transient translator which creates a small, temporary dictionary to hold values. In answer to the question, "With respect to what are the variables local?" he answers that he tries to keep them within one colon definition. His proposal tends to reflect his antipathy to stack operators.

George Shaw's approach, what he calls "Rock and Roll Programming," stores local variables under the stack with his word **ROCK** and fetches them with **ROLL**. This differs from some other solutions in that it takes values off the stack and so works like other Forth operators.

William Volk's goal was to have local variables without affecting any of his other code. For graphics solutions, he felt that he could simplify his applications which used many stack manipulations and make them run faster. His variables are kept at fixed locations in memory.

Finally, the Forth community has been far from untouched by the splash caused by the Macintosh's entry into the microcomputer pool. Charles Duff from Kriya systems, which has developed a Forth for the Macintosh (along with MicroMotion and Creative Solutions), spoke of extending Forth in new directions. The complexity of Mac's operating system really stretched Forth's capabilities, especially in the area of data structures.

During the impromptu lectures, Tom Zimmer also spoke of Forth and the Macintosh. He recommends that implementors working on that machine handle all the traps available inside Mac. Most systems do not because of the massiveness of the undertaking. Zimmer opens a MacWrite text file and enters a sequence of simple lines, e.g.:

OPEN A000 TRAPA<CR>

which can be accessed by a Forth command like

MAKETRAP OPEN

which can access the text file and find the needed parameter. This would work if the machine were fast enough to do such a search without bogging down. However, he pre-compiles this text file (which is in alphabetical order) and uses a program which can find the individual entries quickly.

That is this year's FORML conference in a very small nutshell. Complete papers of all the presentations, and some additional material, will be published in the *Proceedings of the 1984 FORML Conference.*

Part One

FORML China Tour 1984

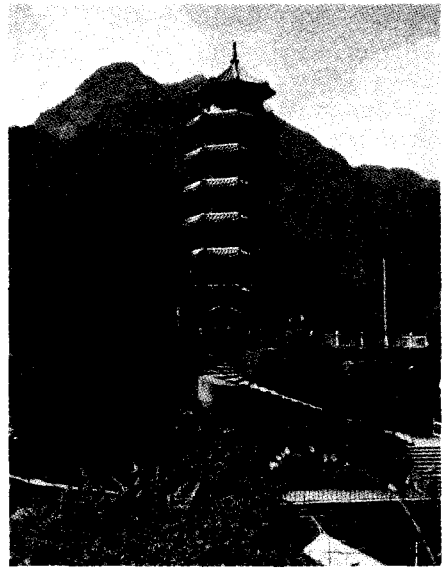
9-25: The FORML tour group met in the terminal at San Francisco's International Airport. In that lobby, we renewed old acquaintances and began new friendships with those who would be our constant companions for the next three weeks. After ritually checking pockets several times for passports, traveler's checks and plane tickets, we embarked on an itinerary that would include Taiwan (Republic of China), Hong Kong and the mainland People's Republic of China.

9-26, 9-27: International date line. By this time, our bodies had conformed perfectly to the shape of the regulation airline seats. Henry Laxen and Mike Perry, far from conferring over esoteric aspects of their Forth implementation, were engaged in the culturally appropriate game of Go. Not entirely lost in the temporal world, however, the game-side conversation soon turned to mathematical permutations of the Go board (but really, guys, Moebius Go?) and to Dr. C.H. Ting's relentless Forth version of the classical Chinese pastime.

The group changed planes outside Tokyo. We girded ourselves for an instant transition from Western to Eastern culture, but the connecting flight's announcements were in English despite the minority of Americans on board. After a short flight that brought our total flying time to about twelve hours, it was a relief to prepare for landing in Taipei. And as we tiredly hauled our carry-ons through the gate (with enough cameras to forever quell any cliche remarks about visitors to our own country), we were greeted by smiling faces and large banners prepared by members of the Taiwan FIG Chapter!

We were ushered quickly through customs and, after introductions and photographs on all sides, boarded a bus for the hour-long commute to Taipei's Lai Lai Sheraton. But not before being informed that the next day's activities would begin with an 8:30 a.m. departure!

9-27: This morning saw us visiting the Hsinchu Research Institute, whose



George W. Shaw II

Taroko Gorge was an exhilarating example of Taiwan's natural beauty.

Electronics Research and Service Organization (ERSO) brought integrated circuitry to Taiwan. They are engaged in advanced research as well as in actively implanting the results of that research into the nation's industries. It was a special pleasure to tour their facilities at the invitation of the institute's Dr. Tang and to meet with a few of their four thousand employees, because ERSO was one of the hosts of the Taipei FORML Conference.

During our tour, we had the opportunity to watch a Forth-controlled robot arm demonstration. It used video input to sort out blank metal disks from those with letters that it arranged to spell "ERSO." The machine operated under the Universal Robot Control System developed by engineers there. A meeting followed in which questions and answers were exchanged about the technology and the Forth implementation, and a formal presentation was made to that group by FIG President Robert Reiling.

It had been a busy morning, and with the prospect of several intense weeks ahead, a number of us took advantage of the lunch-time opportunity to have pizza at the hotel. An afternoon visit to the Chiang Kai Shek memorial, a truly monumental building, concluded the first day we spent on foreign soil.

9-28: The day started with a visit to the Palace Museum, where the accumulated art treasures of centuries are on display. The hours spent there were hardly sufficient to view the thousands of jade, bamboo, silk and other precious objects, which are replaced frequently by items from a seemingly endless storehouse.

The afternoon found tour group members exploring the modern, technological arts available to consumers. Shops were full of a wide variety of modern electronic products, only some of which were significantly lower priced than in the United States. More interesting to many, perhaps, were nondescript stores selling familiar-looking computers with no maker's imprint. Due to recent efforts by IBM, no versions of that company's PC were in sight; other manufacturers will have to remember that imitation is the sincerest form of flattery. The most notable examples of pricing strategy were found on the software shelves. Major applications like dBase II typically cost four dollars per diskette. Local Forth enthusiasts also have a complete selection of Forth books which have been translated into Chinese.

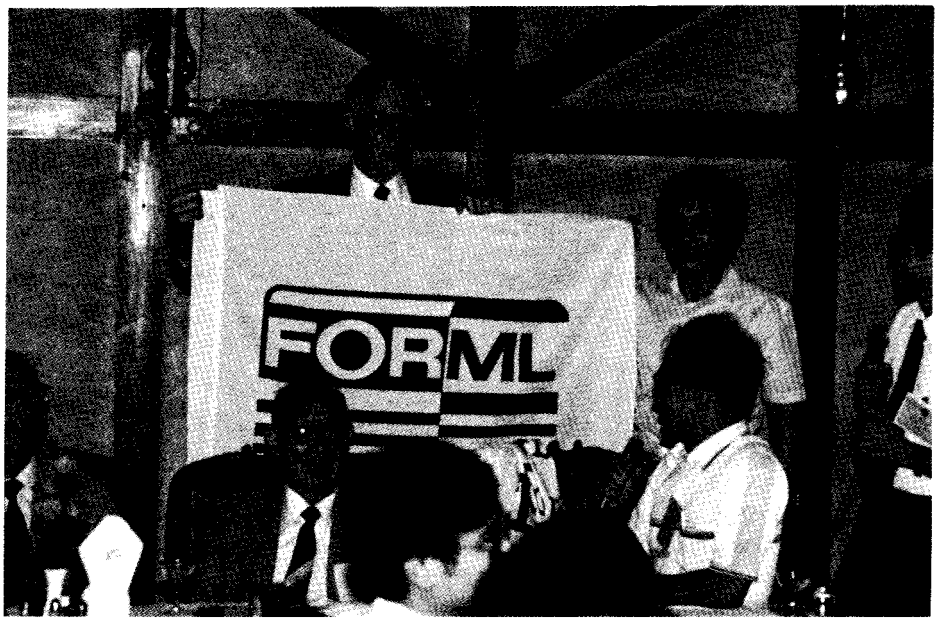
Back at the Lai Lai Sheraton, Marguerite Philips, our American tour guide, hosted a cocktail party for speakers and guests. On that occasion, the final schedule of topics and speakers was drawn up for the conference which would begin the next day.

9-29: An array of large blue and white FORML flags and a forty-foot banner greeted those who arrived at TamKang University's impressive International Conference Hall. The Taiwan FORML Conference began with presentations by the University's President Dr. Clement C.P. Chang and by its Vice-chancellor Dr. Louis R. Chow. Key organizers of the two-day event also included Dr. Timothy Huang, Dr. C.H. Ting, Dean of the Graduate Institute of Management Science Professor

Chien Lih, and Secretariat Ching-Tang Tseng, and many other distinguished planners and sponsors.

During program introductions by our hosts, Forth was touted as the best of the fourth-generation computer languages, and its importance in process control was stressed. Robert Reiling then spoke on the history and goals of the Forth Interest Group, a topic which was greatly appreciated by local FIG members, who are in the process of obtaining government recognition of their group. Bill Ragsdale next shared some non-technical ideas, such as how a piece of Forth code can reflect how the programmer thinks and solves problems, and how the language does not have the built-in limitations of other languages — instead, the limits are the mind and ambition of the programmer.

Wil Baden surprised everyone by beginning his lecture about F83 in Chinese. He discussed his translation of F83 for the Apple II and concluded by writing in Chinese characters on a



At our farewell banquet, Robert Reiling accepts a gift from the Taiwan FIG Chapter and Tam-Kang University.

— *George W. Shaw II*

transparency, drawing pleased applause from the audience. During intermissions, Wil was among those who demonstrated Forth to eager participants.

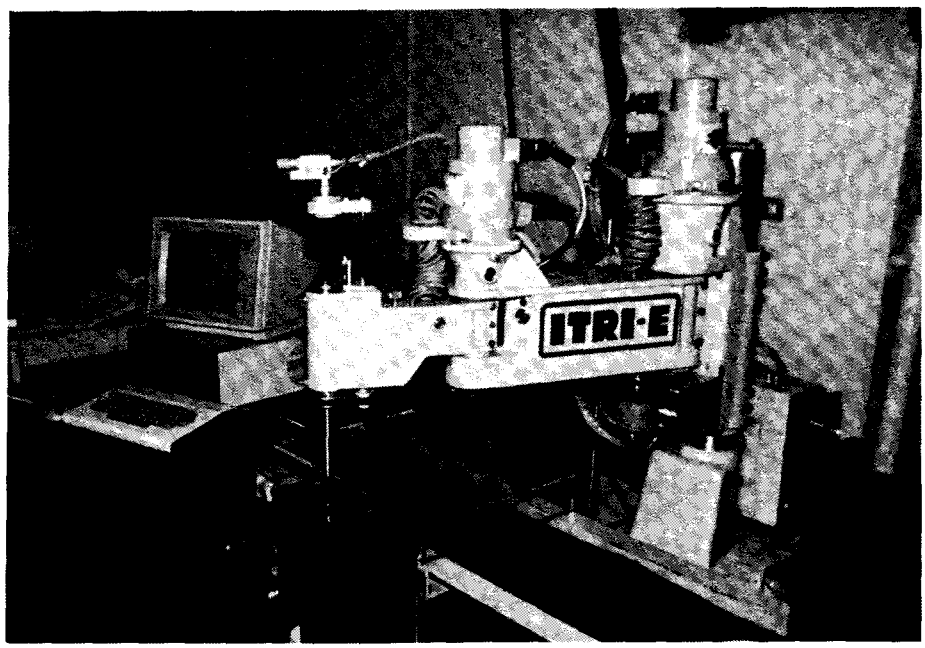
Timothy Huang proceeded to discuss how programmers whose native tongue is Chinese can program in Forth more easily than in other computer languages which are more closely aligned with English. His company has a Chinese version of Forth, and he showed some samples of its code. One definition, which he considered unnecessarily large, he compared to the very long cloths used to bind very small feet.

Ching-Tang Tseng, President of the Taiwan FIG Chapter, spoke on teaching Forth. His paper includes lists of courses which have been taught in that country, content and structure of the classes, and locally developed programs used in teaching those classes (which are available to any who might benefit from them). He also spoke on the desirability of lowering the average age of Forth students.

After some additional talks by American attendees (largely covered by the review of the 1984 FORML Asilomar meeting), H.C. Liu described a significant application he wrote in Forth for a military hospital. His program runs on an Apple II to measure a patient's pulse and correlate it with other vital signs.

9-30: Everyone was eager to return for the second day of the Taiwan FORML Conference. One of the first speakers was H.L. Lee. He had written a Forth program to solve a friend's inventory management problem. This practical system was designed for in-office use, and is forgiving of certain types of operator errors. A key element is that the user doesn't feel like he or she is learning a new computer language.

Y.M. Wu discussed "Forth Applications in Water Resources Planning." He has a simple method of using the stack to record search paths through a binary tree. Since cities are in different locations relative to water supplies, this "lowest cost flow" technique finds the least expensive flow path to get water to cities.



— George W. Shaw II

A project of great interest at ERSO is the Forth-controlled robot arm.

K. Huang presented his "6502 Cross Disassembler in Forth-79." As an addendum to his talk, he requested that all available assemblers and disassemblers be collected and published. This would save much work reinventing the same utilities, and would help to promote Forth as a development language.

After a stimulating day, the conference was officially closed with gifts of recognition to Sam Suan Chen and the Taiwan FIG Chapter officers and the conference staff; and to Robert Reiling, William Ragsdale, Charles Moore, Henry Laxen and Mike Perry. Sixty people attended a farewell banquet at which Charles Moore spoke about Forth's outstanding performance in major applications and its superiority over Prolog as the language of the fifth- and sixth-generation computers.

10-1: The FORML tour group regretfully left Taipei after a fond farewell to our friends there. Only hours later, we arrived in Hong Kong to check into our hotel in the Wanchai district. Our windows looked across the harbor into Kowloon and toward mainland China, our next destination. Most of us gathered that evening for dinner aboard a three-story floating restaurant and enjoyed a later drive up Victoria Peak for a look at the island's night lights.

10-2: Nothing was planned until later this evening, so a free day of shopping was declared by one and all. Going our separate ways, the FORML group was quickly assimilated into the international crowds. Miles of trekking through the humid heat took us past a Gucci's (next to a McDonalds), acupuncture shops (next to stereo and camera shops), high-tech malls and high-rise tenements. Enough of us wandered together into a tailor's store to create a veritable shopping frenzy amid bolts of silk and leather. At the evening dinner, we had a meeting to plan our entry into the mainland and to fill out the many required forms. Last-minute shopping and shipping (via the legendary slow boat) preceded nightcaps and a well-earned rest for our aching feet and slender wallets.

Coverage of the FORML Tour Group's activities in the People's Republic of China will be detailed in the next issue of Forth Dimensions.

Chapter News

John D. Hall
Oakland, California

We have three new chapters for a total of sixty-eight!

Orlando FIG Chapter,
Orlando, Florida

Tampa Bay FIG Chapter,
Clearwater, Florida

Central Oklahoma FIG Chapter,
Oklahoma City, Oklahoma

Atlanta FIG Chapter

August 21: The meeting was very well attended considering it is vacation season. This month the session did not feature a speaker but, as usual, lots of topics surfaced in general discussion. Forth-83 differences are causing some consternation but it was pointed out that languages of any kind have to be alive. Once in four years seems to be a reasonable revision rate. The question of Forth's readability came up. It is, perhaps, easier to write cryptic code in Forth than in other languages, but one could write Forth almost in English. It seems more a matter of style and, perhaps, more time needs to be spent in up-front work before getting down to the "fun" part of coding.

September 25: As usual, our meeting did not lack for interesting and wide-ranging discussion, including the following topics: advertising ourselves locally, Charles Moore's rumoured microprocessor, how MacForth is gaining attention, the new book *Complete Forth* by Alan Winfield, examples of shadow screens, *Micro* magazine, the use of **1 .R** to eliminate spaces in ASCII output, etc. Recall hearing about the expert system redone in Forth by GE? At a recent, well-attended presentation, the most persistent question about the system concerned the substitution of Forth for LISP. The speaker replied that Forth lent itself better to the successful implementation on a micro. Three cheers!

October 23: Ron Skelton mentioned that we now have forty-five people on the mailing list and that we have held regular meetings since July 1983. How time flies! People at MSA have been contacted and confirmed they were using Forth in their Designware products marketed by Peachtree Software. The article in *BYTE* (September 1984) about KAMAS, an outline text/data processing product using Forth, was discussed. The group also discussed John Dvorak's article in the October 29 issue of *InfoWorld* which was highly critical of Forth, but which was more sensational than substantial. Much other discussion centered on trying to implement the Laxen and Perry F83, and on understanding strange words like **DEFER**. The merits of conventional screen-organized Forth versus files of Forth code is a topic that keeps popping up. Some folks are always interested in the speed issue and were planning to try using FASTER-FORTH techniques.

—Ron Skelton

Fort Wayne Chapter

September 14: We have had three successful meetings since our last newsletter. Ed Harmon provided us with an interesting demonstration at our July meeting on the Apple II's MicroMotion Forth. Ed (at that time an admitted novice in Forth, but a competent user of Pascal and COBOL) translated a short Pascal graphics routine into Forth. There was an interesting fall-out from this exercise in view of the often-mentioned criticism of Forth (i.e., you can write it but you can't read it). Pascal was invented to enhance readability. As Ed put the two routines on the blackboard, it was indeed a surprise to witness the almost identical appearance of the two source codes. Furthermore, Ed made note of the fact that the Pascal code had unnecessary redundancy in the parameter-passing statements. Ed is enthusiastic about Forth and has purchased SUPER FORTH 64 by Persec Research for the Commodore 64.

—Blair MacDermid

Los Angeles Chapter

October 27: The morning session began with introductions, and was followed by news, rumours and then lunch. The four afternoon sessions were by Martin Tracy, who presented a three-screen implementation of floating point; Nathaniel Grossman, who continued by expanding Martin's Zen math into a Zen slide rule using the Cordic algorithm; Loring Craymer, who presented debugging help for F83 that does not require patching **NEXT** and will work with in-line **NEXT** systems; and Barry Cole, who presented some performance enhancements and a discussion of measurement tools.

—Barry Cole

Rochester Chapter

The Rochester chapter meets every other month at the University of Rochester. At the May meeting, Paul Conaway exhibited the virtues of the Rockwell F11 chipset and his New Micros "100 Squared" system. July was devoted to a demonstration of MacForth by Nick Francesco. In September, David Harper demonstrated his polyFORTH implementation of the full-screen editor described by Blakely in *Forth Dimensions* (V/2) and later expanded by Gates (VI/1). David also described and demonstrated the internals of Jack Park's Expert Systems. At each meeting, there has been discussion about what people are doing with F83. Orange County chapter handouts have also been of particular interest. Several people are looking for Forth for the PDP-8. Does anyone know of one?

—Chapter Scribe

Greater Oregon Chapter

October 13: Tim Huang reported on the Taiwan FORML Conference. Tom Almy gave a multi-tasking presentation and demonstration of the Tower of Hanoi puzzle done as several cooperative tasks rather than recursively. This involves an elegant message-passing scheme which Tom detailed with source code. Our December meeting this year will have been held at a local restaurant, including spouses and/or significant others. If successful, it will become an annual event.

—Pann McCuaig

FIG Chapters

U.S.

• ALASKA

Kodiak Area Chapter
Call Norman C. McIntosh
907/486-4843

• ARIZONA

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

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

• CALIFORNIA

Berkeley Chapter
Monthly, 2nd Sat., 1 p.m.
10 Evans Hall
University of California
Berkeley
Call Mike Perry
415/644-3421

Los Angeles Chapter
Monthly, 4th Sat., 11 a.m.
Allstate Savings
8800 So. Sepulveda Boulevard
½ mile North of LAX
Los Angeles
Call Phillip Wasson
213/649-1428

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

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

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

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

Sacramento Chapter
Monthly, 2nd Tues., 7 p.m.
170B 59th St., Room C
Call Tom Ghormley
916/444-7775

Silicon Valley Chapter
Monthly, 4th Sat., 1 p.m.
Dysan Auditorium
5201 Patrick Henry Dr.
Santa Clara
Call Glen Tenney
415/574-3420

Stockton Chapter
Call Doug Dillon
209/931-2448

• COLORADO

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

• CONNECTICUT

Central Connecticut Chapter
Call Charles Krajewski
203/344-9996

• FLORIDA

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

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

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

• GEORGIA

Atlanta Chapter
Call Ron Skelton
404/393-8764

• ILLINOIS

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

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

Rockwell Chicago Chapter
Call Gerard Kusiolek
312/885-8092

• INDIANA

Central Indiana Chapter
Monthly, 3rd Sat., 10 a.m.
Call John Oglesby
317/257-0217

Fort Wayne Chapter
Call Blair MacDermid
219/749-2042

• IOWA

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

• KANSAS

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

• LOUISIANA

New Orleans Chapter
Call Darryl C. Olivier
504/899-8933

• MASSACHUSETTS

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

• MICHIGAN

Detroit Chapter
Call Tom Chrapkiewicz
313/562-8506

• MINNESOTA

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

• MISSOURI

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

St. Louis Chapter
Monthly, 3rd Tues., 7 p.m.
Thornhill Branch of
St. Louis County Library
Call David Doudna
314/867-4482

• NEVADA

Southern Nevada Chapter
Call Gerald Hasty
702/452-3368

• NEW MEXICO

Albuquerque Chapter
Call Rick Granfield
505/296-8651

• NEW YORK

FIG, New York
Monthly, 2nd Wed., 8 p.m.
Queens College
Call Tom Jung
212/432-1414 ext. 157 days
212/261-3213 eves.

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

Syracuse Chapter
Monthly, 1st Tues., 7:30 p.m.
Call C. Richard Corner
315/456-7436

• OHIO

Athens Chapter
Call Isreal Urieli
614/594-3731

Cleveland Chapter
Call Gary Bergstrom
216/247-2492

Cincinatti Chapter
Call Douglas Bennett
513/831-0142

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

• OKLAHOMA

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

• OREGON

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

• PENNSYLVANIA

Philadelphia Chapter
Monthly, 3rd Sat.
LaSalle College, Science Bldg.
Call Lee Hustead
215/539-7989

• TENNESSEE

East Tennessee Chapter
Monthly, 2nd Tue., 7:30 p.m.
Sci. Appl. Int'l. Corp., 8th Fl.
800 Oak Ridge Turnpike, Oak Ridge
Call Richard Secrist
615/482-9031

• TEXAS

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

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

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

• VERMONT

Vermont Chapter
Monthly, 3rd Mon., 7:30 p.m.
Vergennes Union High School
Rm. 210, Monkton Rd.
Vergennes, VT
Call Hal Clark
802/877-2911 days
802/452-4442 eves.

• VIRGINIA

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

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

Richmond Forth Group
Monthly, 2nd Wed., 7 p.m.
Basement, Puryear Hall
Univ. of Richmond
Call Donald A. Full
804/739-3623

FOREIGN

• AUSTRALIA

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

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

• BELGIUM

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

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

• CANADA

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

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

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

• COLOMBIA

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

• ENGLAND

Forth Interest Group — U.K.
Monthly, 1st Thurs.,
7p.m., Rm. 408
Polytechnic of South Bank
Borough Rd., London
Contact Keith Goldie-Morrison
Bradden Old Rectory
Towchester, Northamptonshire
NN12 8ED

• FRANCE

French Language Chapter
Contact Jean-Daniel Dodin
77 Rue du Cagire
31100 Toulouse
(16-61)44-03

• GERMANY

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

• IRELAND

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

• ITALY

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

• REPUBLIC OF CHINA

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

• SWITZERLAND

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

SPECIAL GROUPS

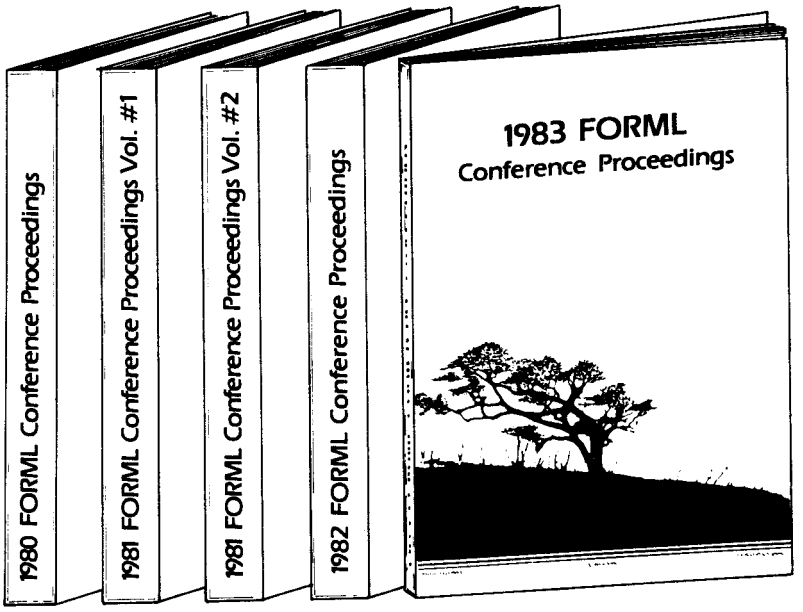
Apple Corps Forth Users Chapter
Twice Monthly, 1st &
3rd Tues., 7:30 p.m.
1515 Sloat Boulevard, #2
San Francisco, CA
Call Robert Dudley Ackerman
415/626-6295

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

FIGGRAPH
Call Howard Pearlmutter
408/425-8700

FORTH INTEREST GROUP
Winter Special

FORML
Conference
Proceedings
1980-1983



SPECIAL PRICE
Available Until April 1, 1985
Complete Order Form on Page 24

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

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

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

Address Correction Requested