Conference Abstracts

The following abstracts are from presentations made at the 1984 FORML Conference, November 23-25, 1984; Proceedings published by the Forth Interest Group.

NONCE Defining Words

Wil Baden

The problem of "self-defining words" occurs again and again in Forth but is often not recognized for what it is. It is a paradox of Standard Forth that you can define a word <namex> which you can use to define a word <name>, and never use <namex> again, but you cannot define <name> directly.

Development of a Tightly Coupled Forth to Operating System Environment

Peter K. Blaser
TELELOGIC INC.
Cambridge, MA 02139
Lawrence P. Forsley
University of Rochester
Laboratory for Laser Energetics
Rochester, NY 14623-1299

Most Forth systems either ignore or replace the operating system which invoked them. If the operating system is well written and/or offers many desirable features, then replacing or ignoring it may be a mistake. In addition, the interactive nature of Forth and its extensibility tend to enhance a good operating system and may well make a poor operating system tolerable.

This talk will describe the continuing evolution of a Forth written under contract to Digital Equipment Corporation for the VAX architecture under the VMS operating system. Some goals of this project have been to develop an interactive interface to VAX/VMS system services and Record Management Services (RMS), [1] implement VAX/VMS Help for Forth, replace the VAX/VMS default Command Language Interpreter (DCL) with Forth, and develop a run-time loader in Forth.

Our implementation began with the Kitt Peak version of Forth [2], which we significantly modified to: reduce the kernel size, use hardware error detection schemes rather than costly software error detection, and allow the user an interactive interface to VAX/VMS system services including RMS.

With the exception of its 32 bit architecture, this is nominally a Forth-83 standard system. Future work will include separated headers from code, a vocabulary structure similar to "ONLY", a run-time loader, a method to produce VAX/VMS images from Forth code, novel code threading techniques [3], and multi-tasking.

We invite other universities to participate in this project with us. System copies will soon be made available for media cost and the Institute for Applied Forth Research, Inc. will coordinate work in many areas including software tool development and programmer productivity.

References:

[1] David Forster, "Operating System Services in Forth for VAX/VMS" Proceedings of the 1984 Rochester Forth Conference. pp. 108-114.

[2] Richard Stevens, "Forth 11 Reference Manual", Kitt Peak National Observatory, November 1982.
[3] William Sebok, "A VAX Implementation of the Forth-79 Standard", presented at the 1984 Rochester Forth Conference, and submitted to the Journal of Forth Application and Research.

An Improvement Proposal for DO . . . +LOOP

John Bowling
Starlight FORTH Systems
15247 N. 35th St.
Phoenix, AZ 85032

DO . . . +LOOP has an inherent speed disadvantage and a programming problem that can be overcome for most fixed increment +LOOP functions. The IDO . . . ILOOP proposal allows the incremental value to be specified when IDO is executed, and keeps the increment on the return stack. A word with a loop can now have a different increment every time it is executed, without fancy stack adjustments.

Object Oriented Programming

Ronald D. Braithwaite
Rising Star Industries
P.O. Box 3063
Idyllwild, CA 92349

Structured programming in the '70s is giving way to object oriented programming in the '80s. Following an object oriented approach involves more than mouthing slogans, but involves issues such as re-entrancy and the minimal use of impure data structures. Difficulties in maintaining data objects are discussed, with the speed/complexity tradeoffs examined. Techniques for avoiding "smashing" a data object are presented and object oriented programming in a multi-tasking environment is covered.

Disk I/O Under Operating Systems

Ronald D. Braithwaite
Language Group Leader
P.O. Box 3063
Idyllwild, CA 92349

An area of FORTH desperately needing standarization in some form is that of using FORTH under an operating system. Faced with the necessity of porting a series of major applications (about 15,000 screens of source and another 5,000 shadow screens) to several operating systems, the Rising Star Industries Language Group developed a standard physical disk interface. This method uses a dynamic memory allocator in order to allow multiple files to be open, each with a different block size, a modified LRU mechanism that allows for locked blocks, and extensions that allow for byte stream disk I/O, as well as the existing FORTH standard mass storage words.

The Development of a Graduate Course on Microprocessors in Product Design for Mechanical Engineers

Charles E. Buckley
Stanford University
Palo Alto VA Hospital

The Smart Product Design Laboratory in the Design Division of the Mechanical Engineering Department at Stanford University and its associated curriculum were organized seven years ago. At that time, this facility represented a unique resource for introducing mechanical engineers to a topic of which they had heretofore been expected to remain ignorant — the use of microprocessors in the design of electromechanical products.

Since that time, the level of sophistication in laboratory capabilities and curriculum content has grown steadily. One manifestation of the facility's developing maturity has been a shift towards the use of the Forth programming language for practically all development work done there.

In this paper, the results which this shift has brought about are described. Following a brief

introductory discussion of the facility and curriculum, a focus will be made on two particular issues. The first of these concerns changes which have been made to the curriculum to accommodate and/or take advantage of Forth. The second deals with the changes which have been experienced in the scope of projects which may be reasonably undertaken by students during the course of an academic year.

A Decompiler Design

Bob Buege
RTL Programming Aids
10844 Deerwood SE
Lowell, MI 49331

Although FORTH was originally developed for control applications, its dependence on screens makes it difficult to use without a disk drive. By using a high quality decompiler, it is possible to regenerate source code from object code and edit any high level word directly from RAM. If this capability is added to a token threaded language which also allows deletion of unused words with garbage collection, the need for screens is eliminated, allowing the language to be totally independent of a disk drive. The decompiler described in this paper has been used as the basis for such a system for over a year and the resulting system has been found to be much easier to use than conventional screen based systems.

Status Threaded Code

Bob Buege
RTL Programming Aids
10844 Deerwood SE
Lowell, MI 49331

RTL is a Relocatable Threaded Language which I developed to overcome some of the weaknesses of FORTH. Because of the requirements which I demanded from RTL, I was forced to invent a new class of threaded languages which is more flexible than languages based on either direct threaded code or indirect threaded code. The purpose of this paper is to describe this new type of threaded code which I call status threaded code and show how it was used to advantage in the creation of RTL.

The In-word Parameter Words

Sam Suan Chen
Institute of Nuclear Energy Research
P.O. Box 3-3
Lung Tan, Taiwan 325
Republic of China

Another implementation of in-word parameter structure over Huang's scheme (*FORTH DIMENSIONS* NO. 3, 1983) is presented to assure a simpler form and the correct functioning in colon definition as well as in interpretive mode.

Another Look at DTC

Dr. Loring Craymer
Division of Biology
California Institute of Technology
Pasadena, CA 91125

A straightforward form of direct threaded code has been implemented on a Z80-based microcomputer. Instead of containing pointers to machine code, the compilation fields of all non-CODE words take the form CALL <routine>: W is passed on the hardware stack rather than through a dedicated register. This costs slightly in terms of space, but average execution time decreases by 18–20%. NEXT becomes a 7 byte routine, so that an in-line NEXT costs only four bytes per CODE word, for a total of about 350 bytes for the Laxen-Perry F83 model; this cost is somewhat offset by the two bytes per word saved by DTC in comparison to ITC. Having NEXT in-line results in a 9% further improvement in execution speed. The DE register pair was dedicated to hold the top of the parameter

stack: execution speed improves by a few per cent. The alternate register pairs were dedicated to hold the current loop index' and limit' values. The net result of these changes is a nearly 35% faster implementation at a cost of only a 3% increase in memory usage.

This form of direct threading should be applicable to all conventional microprocessors. A model for implementing faster 32-bit FORTH systems on 16-bit machines will also be discussed.

Extensions to Forth in the DOS Environment

Thomas B. Dowling
Miller Microcomputer Services
61 Lake Shore Road
Natick, MA 01760

Extensions to the Forth interpreter can make improved use of the file system available in DOS. The Forth interpreter has an additional rule added to try interpreting a word as a file name if it is not in the dictionary and is not a number. The file, if it exists, is then loaded. This can be used to load libraries or utilities. By adding some conventions to the file source code, nothing happens for an already-loaded library, and an already-loaded utility is executed without loading.

Experience using these modifications to MMSFORTH has shown their value. Some points on using a file system for the control of source code in the development of a large project with many programmers will also be discussed.

NEON — Extending Forth in New Directions

Charles B. Duff
Kriya Systems, Inc.
505 N. Lakeshore, Suite 5510
Chicago, IL 60611

NEON is a threaded language that arose out of a need for a powerful development system on the Macintosh. NEON is an object-oriented language that has inherited characteristics from other extensible languages, including Forth, Smalltalk, Logo and Lisp. This paper uses NEON as a forum for discussing some areas in which Forth might be improved, and suggests models for certain extensions. Areas covered include: Named parameters, local variables, defining words, vectoring, multiple-code field data types, data structures, and forward referencing. An early version of NEON was described in the paper, "Forth Meets Smalltalk", presented at the 1984 Rochester Forth Applications Conference.

A Critical Evaluation of Threaded Interpretive Systems

Harvey Glass
College of Engineering
University of S. Florida
Tampa, FL 33620

While languages based upon threaded interpretive systems have been used for a variety of applications, these systems have been generally ignored by serious students of programming languages. We propose to critically examine — to gain insight, and to investigate the suitability of these systems for implementing a programming environment — specifically an environment to support programming in a functional style.

We hypothesize that threaded interpretive systems may have merit as the basis for more ambitious language implementations that have yet been attempted — and that such languages may offer a reasonable compromise between the flexibility of more interpretive systems and the efficiency of native code compilers.

We describe extensions to a threaded language which provides the kernel of a functional style language. We propose to examine the promise of such a language and to compare it to the language LISP. The goal is to gain insight into the real and apparent capabilities of threaded languages and to evaluate the potential of such systems for support of functional programming environments.

...FORTH, FIFTH, And BEYOND

Andreas Goppold
2000 Hamburg 26
West Germany

An essay into the time-frame network of instrumental language.

This is not really one coherent essay. Rather it is a collection of several threads of thought. On these threads there are what I call beads of ideas that are loosely stringed together. In between are a few knots, centers where many thoughts cluster to a complex similar to a normal essay. Before I start putting out my web of ideas, I want to give some sort of outline, or abstract: I am talking about FORTH as a subject but it is not only FORTH. What I want to do is to connect the development of one particular instrumental (or here: computer-) language which we call FORTH to the fabric of human cultural development.

The Paradigm of Interactive Programming

Andreas Goppold
2000 Hamburg 26
West Germany

Computing is not yet a truly academic science. That is, the decision about what is considered a factual truth as opposed to conceptual heresy is not yet vested in a social body whose language structure in essence pre-forms and pre-determines the kinds of thoughts that can be thought of at all. (See Wittgentstein: Tractatus Logico-Philosophicus.)

What Thomas Kuhn calls a paradigm of science is in my view this thought structure which forms a meta structure with respect to the theories that are formed by a science. With meta structures, FORTH programmers should be familiar, and even though I am using the term in a somewhat more general sense, I think I will hit a familiar vein.

Douglas Hofstadter has written a book: "Goedel, Escher, Bach", his main theme is Self-Reference. This concept is Hofstadter's name to something applied onto a version of itself. The computer industry is an industry applied to itself. Only through computers in CAD/CAM were microprocessors possible. Modern Computer-assembly lines like that for the MacIntosh can be considered as automated automata factories. This way the computer industry is a prime example of a self referent system. Self reference is closely rleated to the halting theorem of the TURING MACHINE which says that nothing can be said scientifically about the outcome of such a process. Computing, far from becoming a subject of science, seems to swallow up science. Expert systems are the symptom for this. The accounts of scientists who gave their knowledge to the system, and their feeling of depression when the system re-presented their own knowledge to them in much more complete and refined form speaks for some really deep impending changes in much that has been taken for the core of science and which may become as defunct as the weaver sitting at his loom.

A Forth Slide Rule

Nathaniel Grossman
Department of Mathematics
UCLA
Los Angeles, CA 90024

Many users of floating point arithmetic will not have need for a fully-developed, comprehensive floating point package, yet they may find applications for a "slide rule". Such a smaller computational package would — like a slide rule — carry out low precision floating point arithmetic (including addition and subtraction) and include a broad selection of the common higher mathematical functions. As with the slide rule, functional domains would be limited, requiring user foresight. A Forth slide rule implementation will be presented, with a concise four-function arithmetic implementation by Martin Tracy as basis and with all the higher functions derived from the unified CORDIC algorithm.

A Simple Metacompiler

Guy T. Grotke and Guy M. Kelly
San Diego, CA

By accepting a set of reasonable limitations, it is possible to construct an interactive, incremental metacompiler which is simple to understand and simple to use.

Freedom in Programming Languages

John Hall
Oakland, CA

The freedom to invent, experiment, design, and to code, that are so much a part of Forth, are missing in other computer languages today. In these languages, artistic ability is stifled and professional excellence is too narrowly channeled, leading only to inhibited, mundane and gray products. Like the freedoms in our personal lives, these freedoms must be zealously guarded and their preservation defended from authoritarian tendencies.

FORTH Readability

Tom Hand
Department of Computer Science
Florida Institute of Technology
Melbourne, Florida

This paper is concerned with two issues. First, the utilization of pseudocode can aid both as a design tool and as a documentation tool. Second, the style in which FORTH code is presented can simplify understanding.

The 32-Bit Gambit

John F. Healy

The most natural and obvious way to extend FORTH to a 32-bit virtual machine is to treat stack items as 32-bit entities and to rewrite the entire system accordingly. New microprocessors currently being introduced, such as the Motorola 68020, would make this a simple task from the implementor's point of view. While such implementations do not offer direct compatibility between 16-bit and 32-bit systems, they have the virtue of preserving the existing FORTH-83 Standard in almost every detail except for data and address width specifications, thus leaving the programmers of 32-bit systems in a familiar environment. In addition, this way of extending FORTH would lend itself to the complex, math-intensive laboratory and scientific process control applications for which the new 32-bit microprocessors are most adept.

Toward Standardized Modem Words

John S. James
CommuniTree Group
P.O. Box 486
Santa Cruz, CA 95031

A unified approach to personal-computer modem support allows developers to write most applications in a modem-independent way, despite the great variety of modems which now exist or are likely to be developed.

This paper presents a glossary of the current working draft of a general modem word set suitable for unattended operation, and includes an example of implementation. It discusses some of the issues and problems of general-purpose modem support.

Modular Forth

George Kaplan
DesignWare Inc.
183 Berry Street
San Francisco, CA 94107

Most Forth systems compile applications to fixed addresses in the dictionary. Applications can have modular designs, but the modules are "linked" by compiling them together. Applications are generally limited to the size of available memory.

This paper discusses a method for dividing an application into separately compiled, relocatable modules. All words within a module are hidden except those explicitly declared as external. At run-time, an application's modules are loaded as they are needed. A set of words to control compiling and linking of modules is proposed.

A FORTH Development Environment

H. H. Koller
Group for Measurement Techniques and Laboratory Automation
Central Research Units
F. Hoffmann-La Roche & Co. Ltd.
Ch-4002 Basel/Switzerland

For the program development for small standalone systems with programs usually in proms a FORTH system was developed. The hardware consists of a 8085 microcomputer system on single Eurocards based on the Siemens-SMP-Bus with two floppy drives. A simple 8085 Emulator connects the target system to the development system.

FORTH was first implemented using the FIG listing for the 8080 microprocessor.

The following significant modifications to the original code were made:

(1) Change to "subroutine threaded code", which runs faster; allows a mix of assembly and highlevel-code in the same word; allows easy use of interrupts.
(2) Implementation of interactive, incremental target compilation for headerless, romable code.
(3) Possibility to select code and data segments at addresses specified by the user.
(4) Inclusion of simple, fast realtime multitasking.

The resulting FORTH is mainly compatible with 83 Standard. Deviations include the definition of defining words using the constructs:

"DOES> . . . . ;"
"BUILDS> name . . . ;"
(Ragsdale: FD Vol. II/5 Jan/Feb 1981)
and " :[" for the definition of immediate words.

Who's How Dumb in Telecommunications

Philip La Quey

In this paper I present a way of controlling telecommunications using a technique referred to as key capturing. Key capturing can be a very powerful technique especially in telecommunication networks. The source code for several useful words, such as a dumb terminal program, is included.

Local Variables

Robert E. La Quey

Explicit use of stack manipulation operators (DUP, SWAP, ROT, OVER etc.) often makes FORTH difficult both to read and to write, thus wasting a precious resource, the programmer's time. A technique for compiling the stack picture into local variables is demonstrated. Using the technique a FORTH-like language can be defined which uses reverse polish algebra and the stack but not explicit stack manipulation. The tradeoffs involved are discussed as are implications for the FORTH virtual machine.

Reverse Polish Translation

Robert La Quey

A translator that generates explicit stack operations (DUP, SWAP, ROT, etc.) from reverse polish algebraic equations is presented.

Arrays and Stack Variables

George S. Levy
5737 Menorca Drive
San Diego, CA 92124

A set of new data structures is proposed which improves FORTH readability by eliminating memory reference words (such as @ and !) and stack reference words (such as SWAP, DUP, and ROT). These structures all obey the same syntactic rules for fetching or storing data and are of two types. The first are memory based such as VARs, ARRAYs, and CARRAYs and the second are stack based such as SVARs and SARRAYs. The latter allow the naming of stack locations and must have their stack position "set" by a sequencing statement similar to the conventional stack picture use in documentation (i.e., input stack — output stack).

High Level and Code Level FORTH Multitasker

George S. Levy
5737 Menorca Drive
San Diego, CA 92124

A high level multitasker has been developed that uses the inner interpreter to go around the round-robin multitasking loop. It is simple, transportable, provides a model which a code level multitasker may easily follow, and is significantly faster than other high level multitaskers. It is designed to work in a memory management environment in which all "jobs" are not visible by all tasks. Task commands can be issued by any task and sent to any other task. The concept of resource has been generalized to include not only memory area, I/O ports, and disk, but also tasks themselves.

Doubling the Speed of Indefinite Loops

Michael McNeil
1271 Lost Acre Drive
Felton, CA 95018

A new indefinite-loop control structure — known as START ENTER UNTIL — is proposed. The new structure is logically equivalent to BEGIN WHILE REPEAT, but in the majority of looping problems its run-time looping mechanism will execute up to twice as fast. This improvement is achievable not only in high-level Forth control structures, but also in the Forth assembler's equivalent control structures.

A Language for Digital Design

Chuck Moore
410 Star Hill Road
Woodside, California

FORTH is a programming language with extraordinary versatility. I was reminded of this while designing the FORTH microprocessor. It has instruction decode based on random logic and I needed a way to document and verify this design.

```
 0   ( Digital logic simulator)
 1 : S. ( n)     DUP 32768 AND IF ." +" THEN 32767 AND . ;
 2 : _ ( n - n)     32768 XOR ;
 3 : and ( n n - n)     OVER 32767 AND   OVER 32767 AND MAX
 4   ROT 32768 AND   ROT 32768 AND   AND + ;
 5 : or ( n n - n)     _  SWAP _  and _ ;
 6
 7   ( Technology)
 8 : 2AND_ ( n n - n)     and _  9 + ;
 9 : 3OR ( n n n - n)     or or   30 + ;
10 : 2XOR ( n n - n)     OVER _ OVER and >R  _ and R> or   35 + ;
11
12   ( Logic equations)
13 0 CONSTANT A          10 CONSTANT B          10 _ CONSTANT C
14 : ENB_ ( - n)  A B 2XOR ; ( 45)
15 : XY ( - n)  ENB_ C 2AND_  A B 30R ; ( +84)
```

This block of FORTH code is the preposterously simple solution. At the top is a digital simulator that can combine signals and determine worst-case timing. In the middle is a description of the logic elements (technology) to be used. At the bottom an example of two output signals determined from three inputs.

A Consequent-Reasoning Inference Engine for Microcomputers

Jack Park
Box 326
Brownsville, California 94919

Application of operational artificial intelligence techniques such as knowledge-based expert systems to the desk-top microcomputer marketplace requires consideration of two limiting design factors: available memory, and system speed. System speed is governed by clock frequency of the host microprocessor, bus width (typically 8 or 16 bits) , and level of program abstraction. For any given microprocessor system, level of program abstraction becomes the governing criteria in system speed. Availability of high-speed working memory is the principal limiting factor, once a microcomputer has been selected to host a program. This paper describes a consequent-reasoning inference engine and rule compiler written to explore the issues of limited memory and program abstraction using an Apple II computer with 48k of working memory. Two design criteria were involved: virtual memory was not to be required for run-time operation of an expert program, and semantic simplicity for the rule set was deemed useful in eventual consumer acceptance. The program described explores static memory allocation as a technique for application of working expert systems to desk-top microcomputers.

Outline of a Forth Oriented Real-Time Expert System for Sleep Staging: a FORTES Polysomnographer.

Dana Redington
Sleep Research Center
Stanford University School of Medicine

Knowledge Engineering — a rapidly growing segment of Artificial Intelligence — is transforming the way computers interact with the world. Machines are mimicking highly trained specialists in

various fields, hence the designation *Expert Systems.* The greatest proliferation of machine intelligence will be seen in the development of microprocessor based knowledge systems or personal expert consultants and operators. This paper focuses on personal expert operators; it briefly introduces microprocessor based expert systems and describes how these systems can be made to *think* and *behave* in real-time; language environments that promote knowledge engineering are also discussed. A cornerstone problem in the field of Sleep Research helps illustrate the construction of a microprocessor based real-time expert system using Forth.

Yet Another CASE

John Rible
Miller Microcomputer Services
61 Lake Shore Road
Natick, MA 01760-2099

MMSFORTH, having been developed independently of Forth, Inc. and FIG Forths, contains unique implementations of some Forth words. This paper describes MMSFORTH's character CASE statement, which has been used by thousands of programmers since 1979. It is compared to the versions of CASE described in Forth Dimensions' CASE contest. Extensions of the CASE statement are explored.

32-Bit FORTH On IBM Mainframes

John Rojewski
Transaction Services — Systems
Western Regional Operations Center
American Express Company
Phoenix, AZ 85027

The purpose of this paper is to share the philosophy used in designing a 32-bit FORTH implementation for an IBM 370/30xx large-scale system. This implementation provides source-level compatibility with most 16-bit systems, and provides the additional expandability available by using 32-bit stacks and Dictionary elements.

Topics include modifications required to run FORTH under the control of other Operating System(s) (VM/SP or ACP/TPF) and File System (CMS), along with other utilities developed for testing and problem determination. Terminology is FORTH, IBM, and Operating System dependent, and mixed thoroughly.

Toward a Standard Computer

Dennis Ruffer
ALLEN GROUP
Testproducts Division
2101 N. Pitcher St.
Kalamazoo, MI 49007

Standards in the computer industry are an elusive dream of every programmer. To not have to worry about what type of computer, terminal, printer, plotter, or disk you are talking to would be truly amazing. To be able to transfer a document or file to any computer, terminal, etc. anywhere in the world would be simply unbelievable. The International Standards Organization is trying to make this a reality. They are proposing a 7 Layer Model, that I will try to describe in this paper. It creates a new operating environment in which all computers can find compatibility. The problem is, the Model is an entire operating system, and like all standards, manufacturers must agree to support it. Here is perhaps, the challenge, and why I am presenting this paper.

Error Trapping and Local Variables — One Year Later

Klaus Schleisiek
POB 202 264
2000 Hamburg 20
West Germany

A year ago I started to "misuse" the return stack for information other than return addresses or loop parameters. As it turned out, the return stack is the proper place to store information which has a limited lifetime corresponding to a certain execution level.

Storage and retrieval of this kind of information is implemented by building "frames" on the return stack which are linked to each other and do have a minimal common format such that the system will be reset to a known state on ABORT.

This kind of mechanism is the backbone of languages of the PasGol (PAScal alGOL) type and their notion of the scope of names and local data types. But Forth goes beyond these languages from the early days of computing — namely it goes beyond the paradigm of the sufficiency of one stack, the latest and hopefully last example of which is called NS 320XX.

References

[dc0] Colburn, D. "User Specified Error Recovery" FORML '83
[ks0] Schleisiek, K. "Error trapping" FORML '83

ROCK and ROLL Programming: An Innovative Approach to Local Variables

George W. Shaw
Shaw Laboratories, Limited
P.O. Box 3471
Hayward, California 94540-3471

To date, the published Forth implementations of local variables have met with limited success. The implementations have been "expensive" or have had problems related to either memory allocation or stack congestion. An "inexpensive" approach which dynamically allocates the memory, removes the local variable values from the stack and supports local names would be ideal. This paper solves the first two problems. The third problem is very system-specific and at least one adequate simple solution has already been found ("Adding MODULEs to FORTH", D. Vewey Val Schorre, 1980 FORML Conference Proceedings, p. 71).

Algebraic Parsing Techniques in FORTH

Terri Sutton
Rising Star Industries
P.O. Box 616
Silverado, CA 92676-0616

Presented with the need for an algebraic tokenizer and detokenizer, and faced with both size and speed constaints, research was done into classical computer science literature. Numerous algorithms were discovered. One was selected and implemented in FORTH, with significant additions. This method for parsing algebraic equations is presented, with examples of both left-to-right and right-to-left algorithmic state machines.

Zen Floating Point

Martin Tracy
Micro Motion
Los Angeles, CA

This minimal floating-point package was written to meet three design criteria:

1. It must be small.
2. It must be in harmony with Forth.
3. It must be fun.

It is indeed small — three screens of source code compiling to much less than 1K of object code. And, thanks to the magic of a good editor, you will find a (somewhat less readable) one-screen version on screen six. All four floating-point arithmetic primitives as well as input/output conversion are included.

A Buffer Allocation System for Forth

William Volk and *Ron Braithwaite*

The need for contiguous variable length buffers is established. Buffer space is allocated from a pool, and buffers are maintained using a doubly linked list. Methods for maintaining re-entrancy are discussed. Both memory and disk based systems are described.

A Portable Graphics Wordset in Forth

William Volk

Portability in graphics is defined. A minimal overhead (space/speed) system is presented, tradeoffs discussed, and implementation considered. The benefits of Forth as a graphics language and some of its impact on graphics programming are described. Examples include: windowing, turtle-graphics, 3D perspective, and computer aided design.

Named Local Variables in Forth

William Volk

Recursion, re-entrancy, and readability provide examples where named local variables are useful. Various methods for providing this are presented, with the size/speed tradeoffs discussed. A method for using existing variables through the technique of shallow binding is presented. An implementation is described that uses existing data structures. Future speed/space enhancements will be shown, as will the use of multiple CFA's to improve performance. Some sample programs will illustrate where use of Locals is appropriate.

Valdraw, A Drafting Program

William Volk and *Rick Sanger*

A "user-friendly" drafting program is presented. Hands-on demonstration and examples of output will be shown. Coding techniques for graphics programming, in Forth, are discussed.

Meta-compilation of High Level Forth Code

Harry Roy Wilker
Sentient Software, Inc.

A technique will be discussed that can be used to meta-compile high level application code on top of existing off the shelf Forth kernals in an interactive environment. A host system is used which controls the compilation environment of the target machine, while allowing interactive development on the target machine. It has proven especially useful in those situations where large scale applications must be written for machines with limited resources.

BCD Floating Point Stack + 20 Bit Addresses = 20 Bit Forth

R. Lee Woodriff

The RPN calculator style floating point environment within this FORTH implementation will be discussed along with the parsing routine VAL which turns a string representation of an algebraic formula into a floating point stack entry. Applications of this word to general forth programming problems will be presented. The set of necessary FORTH words for floating point stack manipulations will also be reviewed, with comments regarding the HP implementation. Specialized bit operations for a 20 bit FORTH environment are of particular interest as address space of modern processors increases.

The 83 Standard VS HP71B Forth Environment

R. Lee Woodriff

This ROM based forth language follows the 83 standard to a large extent. A sorted system dictionary accessed by word length as a hashing is used to speed word search. Most 83 standard words refer to 20 bit cells in this implementation. A very nice set of nyble handling words is included for ease of BCD floating point manipulation.

The Future of Forth: Is This the Shape of Things to Come?

R. Lee Woodriff

Compromises for portability are minimized when FORTH is the system language choice for very small systems. A case will be made using the HP71B example that FORTH is an ideal language for portable systems. Program length and speed comparisons will be presented.

This implementation of the FORTH idea will be used to ponder future directions forth should probably take especially in terms of user accessibility and functionality without compromise on power, speed, and compactness of compiled code.

The HP 71B Forth/Assembler ROM: Hardware Environment

R. Lee Woodriff
Menlo College
Menlo Park, CA

Fundamental architecture and description of the device as a forth system will be presented. IO handling and file/buffer words will be covered briefly, particular attention will be given to new FORTH ideas which are potential candidates for inclusion in any new forth standard.

The Macintosh and Forth

Thomas J. Zimmer

Is there life for Forth programmers after Macintosh? I believe there is, although it will never be quite the same, and maybe that is just as well.

When I first received my MAC, I could visualize great wonders in Forth occurring on its screen, flashy windows, mouse controlled menus, you name it and I knew Forth could do it. Well here it is 8 months later, and I do have a crude version of Forth running on MAC, and I may even be able to see the light at the end of the tunnel, but this is one of the longest races I have been in, in a long time. You see MAC isn't like other computers, other computers have ROM monitors providing Character I/O, MAC has a ROM monitor providing over 500 system calls, only one of which prints a single character on the screen, and it forgets to do a CR at the edge of the screen. There is no linefeed of carriage return function, no simple scrolling, even character intput requires an EVENT array, and EVENT sorting to find the key pressed.

Now it may sound like I am complaining, but that's not really the case, you see I don't mind the inconveniences of MAC, because I get power, performance and a user interface equaled by NONE.