Conference Abstracts

The following abstracts are from presentations made at the 1986 FORML Conference, November 28-30, 1986, Asilomar, California.

Charting Forth
Escaping Forth
Hacking Forth
Leaping Forth

Wil Baden
Costa Mesa, CA

No abstracts available.

A New Standard for Forth: Bits of History, Words of Advice

Don Colburn
Creative Solutions
Rockville, MD

This paper surveys the current trends in programming environments, the author's experiences in past Forth standardization efforts, and addresses significant issues confronting any new standardization attempt.

Hashed, Cached Buffers

Loring Cramer
Division of Biology, California Institute of Technology
Pasadena, CA

One of the most useful features of Forth is the built-in virtual memory as implemented in `BLOCK`, `BUFFER`, `UPDATE`, `FLUSH`, `SAVE-BUFFERS`, and `EMPTY-BUFFERS`. Typical buffer management implementations feature a linear search order, and buffer order rearrangement may also be a linear function of the number of buffers. The large memory space of 32 bit Forths makes it feasible to maintain a large number of buffers, but the linear search order can make it computationally expensive to do so. I have discovered an elegant approach to buffer management which features uniformly fast access to buffers and fast rearrangement time. Applications of the buffer manager to physical memory management will also be discussed.

A Simple Flexible Expert System Shell

Susan Eberlein
Division of Biology, California Institute of Technology
Pasadena, CA

The rule based expert system shell is designed to be interactive and flexible. It works by matching strings in the rule stack to those in a fact stack. Facts and rules may either be entered from the keyboard or loaded from disk. Facts may also be entered from within a colon definition. The system can ask for more information during the reasoning process, and includes the facility to change facts if contradictions are encountered. Words may be run from within rules. The rule set is dynamic, and may be changed in the midst of the reasoning process.

Turtles Explore Floored Division

Zafar Essak, M.D.

Using Turtle Geometry to draw circles illustrates aspects of division floored to negative infinity and floored to zero. This exercise suggests that both methods are required for different programming needs. Extensions to a math package are reviewed to illustrate a simple extension to FORTH-83 allowing both methods of flooring division.

A 32 Bit Processor Architecture for Direct Execution of Forth

Martin E. Fraeman, John R. Hayes, Robert L. Williams, Thomas Zaremba
Johns Hopkins University, Applied Physics Laboratory

A simple direct execution architecture for a 32 bit Forth microprocessor was developed for a JHU/APL VLSI development project. The processor can directly access over 4 gigawords of memory and includes local high speed stack memory caches. Two instruction types are defined; a subroutine call, and a user defined microcode instruction (`MICRO`). The `MICRO` instruction executes most Forth primitives in a single clock cycle. We also made measurements on existing Forth code to guide our architectural decisions.

A Usable Operating System in Forth

A. Franklin and L.M. Newell
British Telecommunications plc
S.F. Pelc and R.P. Roberts
MicroProcessor Engineering Ltd

During the autumn of 1985 a group of engineers at the British Telecom Research Laboratories identified a need for a microprocessor product for use within the control and instrumentation industry. The specification for this product was simple; it should be low cost, reliable and most importantly usable. In addition it would have multiprocessor capability and be both manufacturer and processor independent. After an exhaustive study of bus structures and application languages, the STE (IEEE P1000) bus standard was chosen to design the hardware around, while Forth was selected to be the vehicle for creating a software development environment and integrated operating system. With such a combination it would be possible to use the same hardware and software on both development and target systems. This paper tells something of the development of the operating system, its implementation and user acceptance.

Datastructure: Interpolator

Nathaniel Grossman
Department of Mathematics, UCLA
Los Angeles, CA

The barycentric interpolating formula is well-adapted for Forth implementation via defining words.

Forth Tools for Natural Language Interfaces

Tom Hand and U. Shripathi Kamath
Florida Institute of Technology

This article describes the structure of a set of software tools for developing natural language interfaces. The system includes a lexicon builder, a parser that uses Earley's algorithm, and a semantic analyzer that utilizes Fillmore's case grammars. The implementation is totally written in high level FORTH and was developed on the Motorola 68000 architecture.

APE: A Forth Automatic Programming Environment

Tom Hand and Rakesh Mahajan
Florida Institute of Technology

This article describes the implementation of APE, an automatic programming environment. It is an interactive environment that uses the very high level language (HLL) approach to automatic programming. This implementation of APE is completely written in high level FORTH and was developed on the Motorola 68000 architecture. APE produces executable FORTH code based on the specifications as obtained by the user.

A FORTH Standards Proposal: Extended FORTH Control Structures for the Language Requirements of the 1990's.

David W. Harralson
Straightforward
Hollywood, CA

No abstract available.

The Forth Operating System

Glen B. Haydon
La Honda, CA

No abstract available.

An Interpreter and Object Code Optimizer for a 32 bit Forth Chip

John R. Hayes
Johns Hopkins University, Applied Physics Laboratory

A Forth outer interpreter and object code optimizer have been written for the JHU/APL 32 bit Forth microprocessor chip. To support the direct execution nature of this processor, the interpreter allows in line code expansion. An unconventional dictionary structure was also designed for the interpreter. A peephole optimizer that attempts to pack sequences of Forth primitives into a single instruction was written. The table driven optimizer performs instruction sequence reordering and data path and flow control optimizations.

Subroutine Threading

Robert F. Illyes
ISYS
Champaign, IL

A FORTH implementation that fully exploits the advantages of subrouting threading must deal with a number of new problems. These problems have several solutions, not all of which are consistent with FORTH-83. The problems and solutions are largely applicable to both the Novix FORTH and to a conventional processor.

How and Why: Multiple Inheritance Object Systems

Stephen D. Lindner
Insite Computing

Forth is an excellent language for artificial intelligence research and development. This is especially true of ForthTalk, an object-based extension to Forth. ForthTalk's value will be demonstrated by using its multiple inheritance ability to animate an existing picture by simply "mixing in" an animation "Flavor". Source code examples will then be used to further demonstrate programming simplification using ForthTalk.

Prolog at 20,000 LIPS on the Novix?

L.L. Odette
Applied Expert Systems, Inc.
W.H. Wilkinson
Boston, MA

We describe experiments on the Novix with a Prolog that compiles to Forth. On the prototype Novix chip at 4 MHz we have achieved 6K LIPS with virtually a straight port from MasterForth to cmForth. Given bug fixes in the production Novix, optimization of the Prolog run-time system for the Novix instruction set, and a 10 MHz clock rate, we expect that 20K LIPS is possible.

Modular Forth—Import, Export, and Linking

Stephen Pelc and Neil Smith
Microprocessor Engineering
Shirley, England

In a normal Forth environment, a small change in the early part of a program forces the whole application above that point to be recompiled. As applications become larger, the time wasted in recompiling becomes larger, and both vendors and programmers seek ways round the problem. This paper presents one solution, which permits separate compilation of modules, with a choice of early or late linkage. This module system is implemented in a product called "Modular Forth" for MSDOS systems.

Proposed Standard Changes

Stephen Pelc
Microprocessor Engineering
Shirley, England

As a result of recent work we would like to see changes to the number conversion words, and to the vocabulary mechanisms. Our recent work has involved designing Forth systems for newcomers to Forth, or for sale through dealers unfamiliar with Forth. To gain wider acceptance, Forth must become more consistent; to use another buzz-word, it must become more regular. The learning curve for Forth is long enough as it is, particularly for those used to other languages.

The Forth "Standards"

Joel V. Petersen
Nicolet Instrument Corporation
Madison, WI

The Forth-79 and Forth-83 Standards have long been the bane of Forth programmers who have been fortunate not to work on processors with only 16-bit addressing of 8-bit bytes and 16-bit wide data buses. A love/hate relationship has developed between the author and the Forth Standards. Nonstandard Forth on the nonstandard Nicolet 1280 spectral processor is revisited.

The Design Language

Dennis A. Ruffer
ALLEN Group, Test Products Division
Kalamazoo, MI

The polyFORTH Data Base Management System was used to create a way to organize a project's design. This paper discusses this method of documentation, its successes and failures. The software itself is not discussed in depth, as the design of the system, included as an example, is adequate to explain the system. The system, which runs on MSDOS F83, is offered into the public domain, excluding the polyFORTH Data Base Management System.

A Diagnostic Expert System in polyFORTH

Dean Sanderson and Adam Shackleford
FORTH, Inc.

Our client was developing a complex piece of autonomous chemical analysis equipment. Thirty of the custom-built units were to be deployed around the world to measure the accumulation of a toxic chemical in seawater. The analyzer was to perform unattended what had required a testing laboratory and a human technician. It would analyze dozens of samples a day, while the lab had been limited to two or three. After the units were built and deployed, they would require maintenance and service. Personally diagnosing and fixing problems was out of the question; the units were very complex and there would be too many for one person to handle. The other alternative, training technicians, writing manuals, etc., would be time-consuming and expensive. The client needed to be in thirty places at once. Of course, a third alternative occured to our client—an expert system—and that system is the subject of this paper.

Portability: 16 to 32 Bits

Stephen Sjolander and Jon Waterman
FORTH, Inc.

Portability is an important attribute of well written software. The process of porting an application from one hardware environment to another can be viewed as simply a special set of modifications to that application, with an eye toward factoring out those parts of an application most likely to change. The structure of the Forth language lends itself to this modification.

We will describe the process of taking a small application running in one hardware environment and moving it to a new hardware environment. Nearly identical Forth dialects and operating systems are used in both environments. However, the source hardware is a 16-bit processor, while the destination hardware is a 32-bit processor.

The application to be ported is a simple decompiler. Decompiling involves examining the Forth object code found in the dictionary and displaying it in a manner resembling the original source code. Because decompilation is so dependent upon the Forth implementation, this is porting under worst-case conditions. Our goal is to uncover the difficulties involved and to demonstrate techniques to overcome these difficulties.

ZAPPING the F83 Dictionary

C. H. Ting
Offete Enterprises
San Mateo, CA

Most of the words in the `FORTH` vocabulary of the F83 system are useless in normal programming, and they only serve very specific system functions. These words are best placed in the `HIDDEN` vocabulary for safe storage. A word `ZAP` is defined which moves a word in the context vocabulary to the current vocabulary. Using `ZAP`, we can remove all the system words from the `FORTH` vocabulary and relink them into the `HIDDEN` vocabulary. The result is a much leaner and cleaner `FORTH` vocabulary which is more useful to a regular programmer and friendlier to a new user.

A Simple Forth Inference Engine

Martin Tracy
FORTH, Inc.

This paper is the third in a series on extending Forth for artificial intelligence. The first paper presented list-handling extensions (FORML, 1985) and the second extended it to match arbitrary patterns of lists (Rochester, 1986). This paper presents a simple inference engine based on the previous two extensions. The list handler is available from the Forth Model Library, volume 1. The pattern matcher and inference engine appear in the Forth Model Library, volume 4. Both of these volumes are available from the Forth Interest Group.

The Development of a VLSI Forth Microprocessor

Robert L. Williams, Martin E. Fraemen, John R. Hayes, Thomas Zaremba
Johns Hopkins University, Applied Physics Laboratory

A high performance 32-bit Forth language directed microprocessor using 4 micron single level metalization Silicon-On-Sapphire technology has been designed and tested. The VLSI chip contains over 18,000 transistors on a die 73mm^2. This paper describes the details of the processor's design and its implementation.

The 1986 FORML Proceedings is published by the Forth Interest Group, P.O. Box 8231, San Jose, CA 95155.