Conference Abstracts

The following abstracts are from presentations to be made at the 1985 Rochester Forth Conference, held June 12-15th at the University of Rochester, Rochester, New York. These presentations will appear as papers in Volume 3 of the Journal.

Forth's Linguistic Advantages for Intelligent Systems

David James Barnett
Winter Haven, FL

Aside from the obvious systemic advantages of speed and low memory demand, Forth possesses linguistic attributes which make it the ideal medium for developing intelligent systems, allowing for facile implementation of modular description schemes and cognitive templates. 'Words' may embody knowledge about their own interrelatedness forming semantic networks, with the conceptual proximity of the combined elements embodying added knowledge as well. Furthermore, the systemic advantages are preserved as each conceptual metalevel is generated.

This ability of a language to propogate coherently through search and solution space while remaining descriptively concise is virtually the definition of mental process, and from it intelligent machine behavior is ultimately derived.

FORTH Implementation in a High-Level Language

James C. Bender
The BDM Corporation
Columbia, MD

A prototype FORTH interpreter has been implemented in Ada as a research project. The FORTH interpreter is written so as to take advantage of Ada features. The core vocabulary consists of a set of Ada procedures which correspond to single intermediate language codes. The intermediate code instruction set is, therefore, very large. Efficiency is improved with a higher level instruction set, rather than a primitive instruction set.

The components of the interpreter are contained in a set of Ada packages. The central core consists of a package which defines the dictionary, a set of registers, and some basic operations such as pushing and popping from the operand and return stacks, and pushing an item to the dictionary. A hashed table is used for storing word names and pointers into the dictionary. A table lookup is performed rather than a search through a linked list in the dictionary when a word name is to be checked.

A special feature of the interpreter allows the entire dictionary and register environment to be saved and reloaded by name. A word is defined which, followed by a file name, saves the dictionary, name table, and registers to the named file. Another word followed by the file name will cause the environment previously saved to be reloaded.

The key to the functioning of the interpreter is the word definition format in the dictionary and the inner interpreter algorithm. The word definition contains a flag indicating that the word is a primary or secondary word definition, and another flag indicating that the word is an immediate word or a normal word. The inner interpreter is called with a pointer to the word to be executed. If the word is a primary, the procedure which executes intermediate words is called, else the procedure which executes secondary words is called. The procedure which executes secondary words loops until an end-of-word tag is found, recursively calling the procedure to execute words. There is an additional problem to handle, since some words reset the pointer to

the current word so that it points past data imbedded in secondary word definitions. This is taken care of by setting and resetting the instruction pointer in the environment definition package.

Work is presently proceeding on modifying the interpreter design to take advantage of the capabilities of Ada for defining abstract data types and procedures for operating on them. Additionally, an implementation of the interpreter in Modula 2 is in progress. The Modula 2 version is designed from the start to take advantage of Modula 2's capability for defining abstract data types.

On Interfacing MC68010 fig-Forth to Unix

P. R. Blake and N. Solntseff
Department of Computer Science and Systems
McMaster University, Hamilton, Ontario L8S 4K1

This paper describes an implementation of fig-Forth on an MC68010 computer operating under Unix. The starting point of the implementation was a standalone version of fig-Forth written in macro-assembler by C.P. Kukulies (1).

In order to be able to operate under the Unix system, Forth needs extensions to introduce words for performing I/O operations and accessing the filing system. These will be described, as well as the words needed for a printer interface and a "script facility" to provide a complete record of an interactive Forth session.

The paper discusses the problems encountered because of the inherent 16-bit nature of the fig-Forth model and the need to use relocatable rather than absolute machine code. The changes needed to accommodate this, including the modifications made to the inner interpreter, are described. In the current version, the filing-system extensions are coded as C-language routines. The main difficulty encountered is the mismatch between 32-bit C and the 16-bit Forth model. This has been solved through the use of separate C-language and Forth stacks. Because of this, the addition of new C-coded primitives is not as easy as it might seem at first sight and requires further work.

The effectiveness of our solutions to the above problems are discussed and directions for further work to improve the implementation are described. In addition to transporting the fig-Forth to a CT Megaframe and a DEC VAX-11, it is planned to use the same interfacing techniques to provide a version of FORTH-83 on all departmental machines.

Reference

(1) Christoph P. Kukulies, Heider Hof Weg 15, Aachen, 5100 West Germany.

Remote Monitoring and Display of On-Site Energy Conversion System Performance – A Forth Application

Henry C. Botkin, Research Engineer
Science Applications International Corp.
P O Box 2351, 1200 Prospect Street
La Jolla, CA 92038

The remote monitoring and Display system was developed for the Brooklin Union Gas Company (BUG) by Science Applications International Corporation as an integrated hardware and software package. The system provides for collection, storage and display of power plant performance data gathered using existing dedicated data acquisition equipment. The system also provides the ability to remotely access and display this data.

The basic system consists of two major subsystems, the data collection and display system located at the field test site, and the remote access and display system located the BUG general office. Both systems are based on enhanced IBM-XT microcomputers providing high-resolution color graphic display (640x400) and hard copy, as well as automated data manipulation using a simple, function key driven user interface. A DOS version of FIG Forth was chosen as the primary implementation language due to the speed, flexibility and compactness of the language required for the serial data handling to be performed. The actual graphics were produced using compiled Fortran so that the special graphics device drivers required for the output devices would not have to be developed in Forth. However, the Fortran was executed automatically by the controlling Forth code, transparent from the user.

The paper further describes system design constraints, the hardware selected, software structure and language interaction, and future plans for system expansion to other energy conversion systems.

Improving the Understandability of Forth Code

John Bowling
Starlight Forth Systems
15247 North 35th Street, Phoenix, AZ 85032

One of the most often vocalized complaints about Forth is that it is unreadable, and therefore unmanageable because no one but the original author understands it. In keeping with the concept of the Conference, this paper will show one method of improving the readability, and thereby the understanding of program flow.

Samples of structured pseudo-code in three generations will show improved readability as each step is accomplished:

1. Multiple Conditional operations
2. Multiple CASE: operations
3. State Machine operations

DOD Network Protocols in Forth

Mitch Bradley
Sun Microsystems
2550 Garcia Avenue
Mountain View, CA 94043

The Department of Defense family of network protocols (commonly called TCP/IP) allows computers to communicate over a network using different machines and different underlying data transport mechanisms. This paper describes a Forth implementation of a portion of this protocol family, which allows a stand-alone Forth system to communicate over an Ethernet with other systems running Unix 4.2BSD.

A Proposed Forth Symbol Table Structure

James C. Brakefield
Technology, Inc.
Breesport, TX

The data structure used for definitions and the word search of Forth can facilitate various utilizations of the same. My goal is completeness and efficiency.

Constants or literals are stored separately as globals. This is so they can be hashed. The concept of constant is enlarged so that the code string is a constant in the same sense as a text string. A constant is something which is self-identifying and unchanging at the textual level.

A colon definition is then a pairing of a code string with a name. Only pointers to the two "constants" is kept in the definition entry (a pointer to the name string and a pointer to the code string).

The various constant and word definition records are implemented with tagged fields. This allows inspection of the entire record by any routine possessing a pointer into any field of the record. (Decompiling is made straightforward.)

Optional fields are: Reference count, Comment pointer, Parameter or input type list pointer, Result type list pointer, etc.

Conversion of a Token Threaded Language to an Address Threaded Language

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

After becoming spoiled by almost four years of experience with a token threaded language. I find it difficult to imagine ever going back to an address threaded language. Token threading gives me the flexibility of a meta-compiler without the complexity. All code is relocatable. I can modify or delete words at the beginning of the dictionary without recompiling. I can eliminate the words not needed for an application before burning the application in ROM, and since garbage collection becomes a trivial problem, I can even eliminate the need for screens by decompiling object code and doing all my editing from RAM.

Many of the tools which I have come to depend on would be difficult or impossible to adapt to an address threaded language. However, address threaded languages do have a speed advantage over token threaded

languages for most applications. A compromise has been achieved which allows me to do all my development work in a token threaded language and then use self modifying code to convert the language into an address threaded language for increased speed when the project is completed.

A Direct Threaded Code TTL Control Unit

Jack Calderon
Bowling Green, OH

A hardware design for a control unit is presented which utilizes TTL logic and fast CMOS Static RAM to implement a very fast version of DTC. The control unit is capable of sequencing the operations required for NEXT in one memory cycle and for DOCOL and SEMI in two memory cycles. A prototype of the control unit is currently being tested for performance evaluation.

A FORTH-Based Object File Format and Relocating Loader Used to Bootstrap Portable Standard Lisp

Harold Carr and Robert R. Kessler
Utah Portable Artificial Intelligence Support Systems Project
Department of Computer Science, University of Utah
Salt Lake City, UT 84112

For a quarter of a century much Artificial Intelligence research has been accomplished using Lisp as the implementation language. Portable Standard Lisp (PSL) was created to support research activities on a wide variety of processors and operating systems. To aid in porting PSL we developed an ASCII object file format whose relocation directives are essentially FORTH language statements. A Portable Linker is then used to directly link compiled Lisp code normally dependent on the Lisp runtime system. Finally, since the object file contains FORTH statements to handle final relocation of segments, we wrote a relocating loader in FORTH to load and execute these "exported" Lisp programs. The PSL runtime system is ported by processing it with this system as just another exported program. This paper discusses the FORTH-based object file format (which is general enough to handle other languages besides Lisp) and the FORTH-based relocating loader.

Work supported in part by the Burroughs Corporation, the Hewlett Packard Company, the International Business Machines Corporation, the National Science Foundation under Grant Numbers MCS81-21750 and MCS82-04247, and the Defense Advanced Research Projects Agency under contract number DAAK11-84-K-0017.

A Multiprocessing Computer System Composed of Loosely-Coupled FORTH Micro-chip Modules

Gabriel Castelino and Richard Haskell
School of Engineering and Computer Science, Oakland University
Rochester, MI 48063

A computer system containing multiple FORTH micro-chip computer modules that communicate over an 8-bit parallel bus is described. The modules are loosely coupled and any module can invoke routines that reside in any other module using a Remote Procedure Call mechanism. Arbitration for the bus is handled by a central arbitrator which treats all modules equally. The communication link consists of 8 data lines and 6 control lines. The communication protocols are layered following the guidelines of the OSI Reference Model.

The individual modules may have different native instruction sets and can operate at different speeds. The FORTH based Remote Procedure Call handler is identical for all types of modules except for some low-level primitives that are machine-dependent.

A Microcoded Machine Simulator and Microcode Assembler in a FORTH Environment

A. Cotterman, R. Grewe, R.D. Dixon
Department of Computer Science
Wright State University

A FORTH program which provides a design tool for systems which contain a microcoded component was implemented and used in a computer architecture laboratory. The declaration of standard components such as

registers, ALUs, busses, memories, and the connections is required. A sequencer and timing signals are implicit in the implementation. The microcode is written in a FORTH-like language which can be executed directly as a simulation or interpreted to produce a fixed horizontal microcode bit pattern for generating ROMs.

The direct execution of the microcode commands (rather than producing bit patterns and interpreting those instructions) gives a simpler, faster implementation. Further, the designer may concentrate on developing the design at a block level without considering some of the implementation details (such as microcode fields) which might change several times during the design cycle. However, the design is close enough to the hardware to be readily translated. Finally, the fact that the same code used for simulation may be used for assembly of the microcode instructions (after the field patterns have been specified) saves time and reduces errors.

Micro-Based Sonar Operator Training Device

R.H. Davis
3400 Greencastle Road
Burtonsville, MD 20866

The author's group at the Naval Surface Weapon Center is engaged in the development of a low cost desktop trainer for NAVY ASW Acoustic Sensor Station Operators. Each device is based on a Zenith 150 Microcomputer, with the software being developed using ROHDA-FORTH, the author's personal implementation of the FORTH language. Brief descriptions of the system being simulated and the FORTH-based simulation are given. The paper then focuses on the productivity and software management impact of using FORTH in the development effort. Particular areas discussed are 1) the value of very high level coding (using the ROHDA-FORTH Data Structures and Symbolic Function compilers) to expedite development of all coding for an initial pass, 2) the coordinated use of various trace and debug facilities (including the MSDOS DEBUG for assembly words) to achieve extremely rapid code development, and 3) an attempt to maintain and disseminate the Word glossary for the project as the primary coordination working paper. Selected examples of coding from the project will be used to illustrate the use of the high level programming constructs and the trace and debug facilities used in the software development.

A Machine to Implement a Generalized FORTH Environment

R. Dixon, R. Grewe, T. Rocheleau, A. Cotterman
Department of Computer Science
Wright State University

A microcoded design of a machine to implement a 32-bit FORTH has been simulated. The addressing is segmented so that storage allocation and deallocation is quite general. Garbage collection and compactification of memory are supported. The design of the instruction set is in the spirit of the RISC (Reduced Instruction Set) machines, allowing stack operations and indexed access directly into the stack to produce a frame or register-like environment.

The aim of the project is to produce a fast machine, which while supporting the traditional FORTH environment, will allow more traditional parameter passage and access; it will also support integrated packages of languages and applications.

A slightly higher level implementation of a similar FORTH kernel has been implemented by another group on a VAX and that implementation is available.

Optimizing a DOS-Based Forth for Use on Large, Multi-Programmer Projects

Thomas B. Dowling
Miller Microcomputer Services
61 Lake Shore Road, Natick, MA 01760-2099

This paper discusses the features which make a Forth implementation suitable for use on large, multi-programmer projects. Some of these features are fast compilation, temporary compilation (e.g., the Assembler), multi-file loading, library, compiled overlay, fast string, and long memory address support. Utilities for multi-file cross-reference, multi-file search and replace, and file compare are required. These utilities are necessary for tracking the changes made to different files by various programmers and the management of global definitions, data space, and documentation.

A FORTH Implementation of the Heap Data Structure

W.B. Dress
Instrumentation and Controls Division
Oak Ridge National Laboratory, Oak Ridge, TN 37831

The heap data structure is a versatile tool for the FORTH programmer. Its use speeds program development at both the conceptual and implementation stages by allowing dynamic arrays, enhancing garbage collection, and simplifying overlay manipulations. An examination of the high-level code leads naturally to examples involving dynamic arrays, possible use in sorting problems, maintenance of linked-list data structures, and the use of overlays for both data and executable FORTH code. Extensions of this simple heap manager to more versatile but complex situations are suggested.

The implementation of heap presented here has been used for some time in our FORTH-based version of the expert-systems language OPS5. This has resulted in faster extensions to the language, easier maintenance of the multitude of list data structures, and a much faster and simpler means of garbage collection.

Real-Time State Machine Implementation Programming Techniques

Randy M. Dumse
New Micros, Inc.
808 Dalworth, Grand Prairie, TX 75050

Experienced system designers readily admit that state diagrams are more useful in describing real-time automina than flowcharts. This is particularly true during the system definitions, when the actions of the machine are described in terms of the inputs and outputs and time conditioned responses. These same designers would be hard pressed to name even one source describing any method for generate programs from state diagrams. State diagrams have historically been the tools of the hardware designer, which may explain why published materials on conversion methods from state diagrams to computer programs (flow chart, pseudo-code or high level language routine) are not widely available. This paper will attempt to identify why high level language programmers have lined up behind the flow chart (or pseudo-code equivalent) and left state diagrams to hardware engineers and a few stubborn assembly language programmers. It will also explain at least one version of the rules used to design using state diagrams, and how to program from these drawings.

A Forth-Based Iconic Interface

M. Erradi and C. Frasson
Department d'informatique et recherche operationnelle
Universite de Montreal, C.P. 6128 Montreal, Quebec, H3C-3J7 Canada

This paper highlights the power of MacForth as a high performance interactive programming environment for the Macintosh computer, in developing an interactive iconic interface for integrated databases.

Recent improvements both in technology and in user-oriented software have shown the feasibility of a new type of languages. These languages are based on a set of commands which allow accessing and manipulating data without any knowledge of their physical or logical organization. They use high level operators capable of manipulating entire sets as single objects instead of being restricted to one record at a time. In certain cases they span a wide range of users so that they concern both professional and end users' requirements. In such a system an end user can immediately define, modify and manipulate data with sample commands.

There is an acute need for increasing user interfaces. The technology of multiple-window browsing, pop up menu and drawing graphics facilities enhances the capability of the end user and the programmer. MacForth has been specifically tailored to put these functions at their disposal and to equip them with the necessary tools to develop software which fits comfortably within the Macintosh environment.

We present the main specifications and some implemented commands of a command language based on icons. Objects to be manipulated are represented by icons. Properties are attached to each object and are also represented by icons.

The user can associate a property with an object using a mouse. Predefined menus perform different functions (retrieval, update, highlighting, etc.). The system is implemented in MacForth on a Macintosh computer.

Interactive Videodisc Control and Computer-Based Training on the Apple Macintosh

Nick Francesco
J.A.M., Inc.
300 Main Street, East Rochester, NY 14445

This paper will cover the primary reasons for choosing Forth (in particular, MacForth from Creative Solutions, Inc.) as the programming language to accomplish control of interactive videodisc players and to create computer-based instruction.

Methods of interactive videodisc control, and problems associated with that control will be examined. How Forth makes the programming of various computer-based training techniques quick and easy will also be discussed. Special consideration will be given to the innovative features of the Macintosh, and how MacForth makes accessing those features so simple.

Included will be some graphics from the Macintosh that will help to illustrate the concepts discussed. Some examples of code will also be given.

Microcomputer Control of a Machine Tool

D. French, R. Rier and B. Hughes
The University of Waterloo, Ontario, Canada

A Machine Tool Microcomputer Controller has been designed to control a 4-axis milling machine. The microcomputer used was the IBM-PC, using P/C Forth.

The controller conforms to the E.I.A. standards for data input and has provision for Continuous Contouring and Point to Point operations using "canned cycles." For contouring applications the system has full linear interpolation in all 4 axes, switchable circular interpolation in 2 axes, and linear and circular interpolation may be combined.

The normal functions such as "jogging" and data input are made directly from the keyboard. In addition machine programs can be stored on disk, and these programs can then be used to operate the machine tool.

A graphics function has been implemented to enable the cutter path to be checked prior to the machining operation, as the graphics mode is generated from the same program which generates the drive pulse trains for the machine tool, a true cutter path is shown graphically.

In order to obtain the maximum feedrate, machine code was used for generating the output pulse trains. The output to the motor drives was taken directly from the parallel port.

The use of Forth for this application proved to be an efficient method of software control and the ease of embedding machine language within the program enhanced its use. The paper discusses the problems, solutions in writing the control program, and enhancements that were introduced.

The Design of a Low Cost CAD/CAM System

D. French and B. Hughes
The University of Waterloo, Ontario, Canada

There are a number of CAD/CAM systems available to industry in the $20,000 to $300,000 range. The small companies which make up the majority of industry cannot, in general, afford these systems. The system discussed in this paper has been designed to be a low cost system, having all the necessary functions required by the small companies.

The program is written in P/C Forth using the IBM-XT microcomputer.

The design of the program raises two types of problems which must be solved to ensure the system will be efficient, and meet the requirements of industry.

The first problem was to design a system which would not only be easy to use, but would also be self checking to ensure that the correct type of data was input to the system.

The second problem was in using Forth in this type of environment. For example, floating point data input is essential; also, using the 8087 microprocessor which requires scientific notation, requires manipulation of data to overcome such problems. Additional problems were also encountered as the whole system had to be operated in the graphics mode.

The paper discusses these two types of problems and their method of solutions using a number of examples encountered whilst writing this program.

Run-Time Error Handling in FORTH Using SETJMP and LNGJMP for Execution Control

Jay S. Friedland, Robert J. Paul, Jeremy E. Sagan
Turning Point Software, Inc.
11A Main Street, Watertown, MA 02172

This paper discusses and provides several FORTH implementations of the SETJMP/LNGJMP concept for error handling in applications which have words nested many layers deep. SETJMP and LNGJMP may be familiar to programmers who have worked with UNIX/C environments. A SETJMP is paired with a LNGJMP, which when executed later will return control to the code following the SETJMP, passing an appropriate value on the stack. In most FORTH applications a word at each level returns an error or success code which is processed before continuing. The use of SETJMP/LNGJMP eliminates the nesting of IF ... ELSE ... THEN or BEGIN ... UNTIL statements by transferring control to a level where all error conditions may be handled. SETJMP and LNGJMP may be nested to handle local as well as global situations. These constructs reduce code overhead and produce code which is easier to follow. SETJMP and LNGJMP are ideal programming constructs for error handling since they allow control flow to be optimized.

A FORTH-Based Process Control Language for Personal Computers: DPCLC

Peter L. Ginn
301 Sycamore, Lake Jackson, TX 77566

This paper introduces a process control language (specifically for computer-based control in chemical plants) which has several unique features, many stemming from its FORTH antecedents. The language is sufficiently compact that it can be edited, tested, assembled and executed on stand-alone personal computers, but is hardware-independent. It can be extended by the user defining his own new elements. It makes possible very high loop-closing productivity. It features conditional execution of every instruction in the language, which results in a very compact source code. But the most advantageous feature of this language is that the full range of options, any of which can be included in the program for the process control computer, can be perceived quickly by those who must write the programs. At the same time, such options as are included in any one program are presented so clearly that the actions taken by the process control computer are understood easily by those who must monitor the computer's performance.

DEBUGGER: The Design of a Test Aid to Support the Development of an Embedded Computer System

Harvey Glass
University of South Florida
Tampa, Florida

The paper describes the design of a Forth-based debugging aid that has proven useful in the development of a sophisticated real-time microprocessor based controller. The controller provides an interface between a militarized intelligent terminal and 1) an auxiliary bubble memory, and 2) a local area network. The computer programs supporting the controller operate in an interrupt driven multi-tasking environment. The test support facility was implemented to assist in identifying and correcting programming errors in the controller software. We will discuss our experiences in the design of this tool and how it provides a programmer with a means of gaining visibility into the product under development.

Macro/Subroutine Threading and Forth Machines

Ronald L. Greene
Department of Physics, University of New Orleans
New Orleans, LA 70148

With the advent of microprocessors designed to directly implement most Forth primitives in hardware, the use of direct and indirect threading for Forth must be re-evaluated. In this paper I compare a very efficient direct threading technique suggested recently by Alexander Burger with macro/subroutine threading. The latter method is shown to be the more efficient for low to intermediate level Forth programming, while the former is more suited to high level programming. I discuss a method of utilizing the two complementary methods within a single system, resulting in efficient threading over virtually the entire range of Forth programming.

Exception Handling in Forth

Clifton Guy and Terry Rayburn
Bremson Data Systems, Inc.
11691 West 85th Street, Lenexa, KS 66214

Forth relies on the discipline of the programmer to provide the benefits of structured languages: readability, predictability and modularity. A difficult class of problem for structured software is the handling of exceptions to the control flow, including errors. Even in properly designed software, a low level module may detect a condition whose proper resolution resides on a higher level. We describe a general implementation for exception handling in Forth that hides the inherent structure violation within a readable control structure which allows return from any nesting depth of Forth words and control structures. Further, this structure is itself nestable to any arbitrary depth.

Modification of FORTH Control Structures

David Harralson
Mephistopheles Systems Design
5619 Via Inez, Yorba Linda, CA 92686

This paper proposes that the FORTH control structures be modified to provide a more powerful and cohesive set.

Why do we need these changes? As a long term user of FORTH, I have always suffered from the inadequacies in FORTH control structures and early on modified my system as documented in this paper in order to even be able to write application code (well, at least write it easily).

Are the new structures useable? Not only that, but they make programs more readable. The FORTH nucleus is a model of very low complexity, where you wouldn't expect to see the benefit of the new control structures. The implementation code shows some examples.

Local Variables

John R. Hart and John Perona
Hartronix Inc.
1201 North Stadem Drive, Tempe, AZ 85202

There are some cases where the number of variables inside a procedure exceed the maximum that can be easily accessed with simple stack operators. Local variables are proposed to free programs from stack thrashing and spaghetti-like manipulation that can occur in programs with a large number of stack variables.

Standard FORTH can easily access the top three elements on the stack. The Return stack can contain a fourth variable. Programs with more than four variables require a careful arrangement of the stack order and execution sequence. Sometimes there is no reasonable solution. These problems make it difficult to justify using FORTH for tasks that require a large number of variables.

There are several ways to implement local variables. One is to use a standard variable and save its contents at the start of the procedure and restore at the end. Other methods would be to assign space in memory or on one of the stacks and develop a means of addressing the space.

Forth Automates Intrinsic Viscosity Determinations

D. J. Hooley
The Standard Oil Company, Warrensville Laboratory
Warrensville, OH

An automated system for determination of the intrinsic viscosity of polymer samples has been implemented in the Analytical Laboratory at SOHIO.

Multi-tasking FORTH facilitated the implementation of the project and simplified software by enabling independent operation of the tasks. Conventional terminal mode tasks were used for the status display, operator interface, bar code printer and data storage on a remote computer system. Four additional tasks were responsible for control of, and data acquisition from, four viscometer tubes. All of these tasks share a single copy of the control and data acquisition software. Individual data buffers are used in each task to hold task specific information.

Timing for each channel was done in software rather than by individual real time clocks. The 1 KHz interrupt rate of the system's fixed period clock provided more than the required accuracy of 10 ms.

Report generation, calculations, plotting of data and data acquistion were accomplished with memory resident software on a 60 KB LSI-11/2 processor with a total hardware cost of $10,000. Vendor supplied operating systems would have required at least an LSI-11/23, 256 KB memory and a 20 MB disk system. These systems would have also required individual real time clocks for each channel to achieve the necessary timing accuracy. The cost of such a system would have been several times that of the FORTH based system.

System Development Via Prototyping in the Analytical Laboratory

D. J. Hooley
The Standard Oil Company, Warrensville Laboratory
Warrensville, OH

In the appropriate situation, the prototyping approach to system development offers lower risks to the software development manager than the traditional structured approach because it is more likely to produce a system meeting client's needs. (1) The iterative process of prototyping will be examined using the system for determination of intrinsic viscosity of ploymers, implemented in the Analytical Laboratory at SOHIO, as an example. The system evolved into the current form through three major prototypes. Although a number of details still need revision and enhancement, the major system features are working well. The current prototype meets the user's needs, has been in routine since August 1984, and has resulted in a doubling of the analyst's throughput on this test. For prototyping to be practical in a real time data acquisition environment, it is necessary to have versatile development tools. FORTH provides this environment.

Reference

(1) Boar, B. H., Application Prototyping, New York, John Wiley & Sons, 1984.

A Forth-Enhanced Real-Time Control System

Shih-Jiang Hwang
ERSO/ITRI
Taiwan

With compact threaded code structure and interaction capacity, FORTH is considered as an appropriate programming development tool in the laboratory, but its low execution speed restricts it to use in real-time control systems. We solve this problem by dividing the system into two parts: run-time control and user-interface. The former is implemented in conventional compiler or assembly language and the latter in threaded-code structures. They are all implemented as tasks under a multi-tasking real-time operating system. Tasks in the run-time control part perform specific process control such as vision, motion and teaching. The user-interface part supports a flexibly interactive way to let the user define FORTH words.

Cooperation and synchronization among tasks is via system calls and a real-time executive, a system routine to supervise all system external conditions.

In addition, we change the conventional FORTH structure to improve its extensibility. Our FORTH system contains one data segment and one stack segment. The data segment, shared by 9 tasks, contains a dictionary, user areas and buffers. The stack segment is partitioned as 9 parts, with each partition assigned to one task. Thus we implement a multi-tasking FORTH in 128K RAM. Besides a return and data stack in each partition of the stack segment, we add a value stack for vectoring, transformation operation and floating-point calculation with a numerical processor.

The fast speed of conventional languages and the compactness of FORTH have been combined.

What's Wrong With Forth?

John S. James
Communitree Group
1150 Bryant Street, San Francisco, CA 94103

Forth offers unusual control over the entire hardware and software environment, allowing design efficiencies far more important than its speed and memory performance. But some traditional coding practices have used this flexibiilty in ways which impede the development and maintenance of large projects.

And easy access to system facilities has too often allowed vendors to get away without providing complete application support for any particular purpose.

The keys to improvement are modular software design, information hiding, and closure. We will have succeeded when programmers new to a project can quickly come up to speed and contribute to any single area of a complex system.

Numbers and Other Pseudo-Vocabularies

George Kaplan
Design Ware Inc.
185 Berry Street
San Francisco, CA 94107

Conceptually, a number is a constant. For example, "100" is the "name" of a "word" that pushes the value 100 to the stack. Because it is impractical to define a constant for each number, Forth systems handle numbers as a special case. Typically, the text interpreter will attempt to make a word into a number after FIND fails to find the word in the vocabulary search order.

This paper shows that the special handling is unnecessary: numbers can be treated as words in a pseudo-vocabulary in the normal search order. This makes the text interpreter simpler and more flexible. An application could, for example, specify a vocabulary to be searched after the attempted number conversion. Other types of pseudo-vocabulary are possible. A syntax for specifying the "FIND-time" behavior of a vocabulary is proposed and examples of possible usage are given.

Some Problems in Implementations of the Forth Standards

Mahlon G. Kelly
268 Turkey Ridge Road, Charlottesville, VA 22901
and
Nicholas Spies
313 Grace Street, Pittsburgh, PA 15211-1503

While writing a text on FORTH we have examined in detail the FORTH-79 and FORTH-83 standards and 12 implementations. Although the goal of the standards is transportability, few programs are transportable. The reasons include: 1) standard words are so few that most implementations double or triple the length of the dictionary and most programs use system-specific words. 2) The differences between FORTH-79 and FORTH-83 are slight enough that many vendors have not converted to FORTH-83; two standards reduce portability below what it would be if one existed. 3) Subtle re-definition of words in FORTH-83 (e.g. PICK , / , LOOP , FIND) with no explicit documentation of the changes has made it difficult to convert programs. 4) Some specifications are unrealistic and universally ignored (e.g. EMIT should work only with a 7-bit character). 5) Important topics that must be addressed by implementations are not addressed by the standards (e.g. operating-system files, extended memory, floating point, strings, graphics). 6) Ambiguous specification in both FORTH-79 and FORTH-83 have resulted in different implementations of each standard.

A Tokenized Rule Processing System

Steven M. Lewis and Mike Fisher
U.S.C. Department of Bioengineering and Danforth Corporation
Los Angeles, CA

In examining rule processing in LISP we were struck with the crudeness with which text was treated. We have developed a rule processor in FORTH in which the rules are strings of FORTH words. Undefined words are automatically entered into a text vocabulary. As unknown words are entered into the vocabulary they are tested for the presence of special characters and English constructs such as ending with s. Words with appropriate construction are designated as one of 255 special classes. Clauses are compared on a word-by-word basis, each word being tested not only for equality but for synonyms. The processor will automatically equate regular plural and singular words and expand regular possessive forms. Special words allow rules to remember indeterminate antecedents and retain them during the testing of a rule. Articles and similar words are included in the clauses but not tested in matching clauses. Incorporating the syntactic processor into an available inference machine (Park, Mountain View Press) gives a flexible powerful knowledge system.

Investigation of Interrupt Response Times of PDP 11/44 and PDP 11/23 Computers Programmed in FORTH for CAMAC Interfaces

T.S. Lund, J.R. Birkelund, J.A. Abate
Eastman Kodak Company, Rochester, New York

Comparison of interrupt response times for PDP 11/44 and PDP 11/23ı machines, using the FORTH programming language, is presented. The interrupt response of the machines with FORTH implemented as a stand alone operating system is compared with an implementation of FORTH running under the RSX11M operating system. The comparisons have been made on systems operating CAMAC parallel interface busses, both as single bus controllers (IEEE 583), and as parallel highways (IEEE 596), which require a branch driver interface. Some comparisons to the same functions, achieved under the RT11 operating system, are also presented.

From WFF to Proof: Forth and Logical Program Construction

Alexander Luoma
Tallahassee, FL

A decade after the "structured revolution," all is not well in the software world. The writing and revision of programs remains the foremost expense in computing, despite new languages, theories, tools, and work stations for the programmer. All agree that the production of programs is holding back the full use of computers, but disagree whether the problem is programmers, languages, or the natuare of the programming task itself. Currently a number of mathematicians and methodologists (Dijkstra, Hoare, Orr, Gries and others) have launched a strong attack to advance mathematical method as a means to move programming from an intuitive or learned craft, to a verifiable scientific discipline.

FORTH, as a language created by programmers specifically for programming, is in a unique position to expose the "cracks" causing problems in software composition. FORTH has a close logical form in the sense of not only being a virtual machine, but also a virtual logic. The manipulation of the stack leads to WFFs (well-formed formulas), which we call FORTH words or programs.

This paper will discuss Wff 'N Proof by Layman E. Allen, a 1960's board game of propositional calculus using Lukasiewicz notation. Using various FORTH words to demonstrate WFFs, the argument will be made that such WFFs can act as autoclitic frames (described by B. F. Skinner) to achieve what math and grammar cannot do alone, to create further WFFs that extend our ability to create good programs.

References:

Brodie, Leo, *Thinking Forth.*
(Various Authors), *Forth Dimensions.*
Allen, Layman, *WFF 'N Proof, the Game of Modern Logic.*
Skinner, B. F., *Verbal Behavior.*

Concrete Suggestions for a Forth Floating-Point Standard

Ferren MacIntyre
Center for Atmospheric-Chemistry Studies
Graduate School of Oceanography
University of Rhode Island, Narragansett, RI 02882-1197

Many serious programmers dismiss FORTH as unsuitable for professional work because it lacks standard floating-point (FP) Capability. Proposed here is a word set based upon two years of intensive use of the MMSFORTH words (including FQUANs and "long-addressed" variants) with the 8087 numerical coprocessor, with input from several other approaches.

Most operations are the familiar Forth words preceded by an F, or a C for complex arithmetic. Short words are strongly preferred because of the natural length of FP definitions. Unusual words include -F for negation; -F- and F \ for FSWAP F- and FSWAP F/, which are single instructions on the 8087; and FRT2 for FROT FROT, symmetrical with FROT and used about as often.

Left open is the choice of FP number indication: a European-style comma, as 123, or 123,4 (which might be confused with the compiling comma); or an embedded period, as 123.0 (which requires that double-length numbers end with a period, as 123.).

A carefully designed FRAND random-number generator is needed. For compatibility, software-FP packages require an 8-deep FP stack. Index registers seem desirable for array manipulations several words deep from a calling loop.

Plotter Drivers as an Exercise in Forth Wordset Design

A. Richard Miller and Thomas B. Dowling
Miller Microcomputer Services
61 Lake Shore Road, Natick, MA 01760-2099

Three levels of driver design are discussed for a typical X-Y plotter. A minimum driver merely permits FORTH to mimic the plotter's internal BASIC command set. A more elegant one transforms the same commands into a more FORTH-like and more English-like wordset. Finally, the command set is modified and vectored into the existing MMSFORTH TGRAPH (Turtle Graphics) utility. This transforms the plotter into an optional video screen to which one can output TGRAPH graphics routines without any modification.

Demonstrations will be given using the inexpensive Radio Shack FP-215 Flatbed Plotter. Source code for a simple FORTH implementation is included, and can be adapted to many similar plotters.

Editors and Editing Technique: A Review of Fancy and Practical Features in the Evolution of the MMSFORTH Screen Editor

A. Richard Miller and Jill A. Miller
Miller Microcomputer Services
61 Lake Shore Road, Natick, MA 01760

The editor is the ultimate utility for the FORTH Programmer, as it is the one with which he or she works most completely. And since it is apt to be written in FORTH, it is easily adapted to new ideas. Thus the important and malleable FORTH editor receives great attention and is a logical focus for objective and subjective design considerations. Since 1978, MMSFORTH Editor has evolved through many standard and special versions. Some of the more interesting features and user techniques, from our earliest line editor through the sophisticated screen editor in our new MMSFORTH V2.4, are discussed.

Forth in the Computer Numerical Control Environment

John Mullen
Department of Industrial Engineering
Iowa State University
Ames, IA 50011

In order to help students grasp the concepts of computer-aided manufacture, a small Computer Numerical Control (CNC) workstation was built. It consists of a microcomputer, a vertical mill, a plotter, a printer and necessary interface electronics. The workstation may be controlled by means of either a FORTH-based control language or a CNC language closely following the standards of EIA RS-358-B.

Although the capacity of the mill is small and the resolution of the system is not consistent with industry standards, the workstation provides students with the desired experiences and also serves as a physical demonstration of the potential of FORTH.

The development of this workstation was greatly facilitated by the use of FORTH. FORTH-79 was used to test electronic driver and interface circuits, control the plotter and mill, form a basis for the FORTH-like control language, and implement the EIA standard CNC language. In addition, since the CNC languages are imbedded in FORTH, the support features of the FORTH system are available to the user.

The inherent transportability of FORTH together with a hierarchial modular design result in an implementation that is almost totally hardware independent and very flexible. As a result, this project serves as an extensive demonstration of the utility of the integrated FORTH environment in CNC applications.

Forth as a Multimicroprocessor Control System

C. A. Meyerholtz, A. J. Schubert, M. J. Kristo, and C. G. Enke
Department of Chemistry, Michigan State University
East Lansing, MI 48824

In our laboratory, we have sought to place a triple quadrupole mass spectrometer (TQMS) under complete computer control. We decided early in the project that a distributed-processing system of several microcomputers would be needed to control the more than 30 instrumental parameters, acquire data and perform peak-finding in real-time. The hardware, which consists of four 8088-based microcomputers, one master and three slaves, was designed and built using the Newcome-Enke Bus. (1)

FORTH was chosen as the programming language for the multimicroprocessor system because its speed, low memory requirements, modularity and extensibility lead to greater ease in programming and testing. We made several modifications to the basic FORTH system to support distributed-processing. The slaves' interpreters were changed to accept commands from the master. The target-compiler was modified to compile code directly into slave RAM. This allows testing of the code on the master beforehand and easy modification of code later. All software was written so that the programmer interacts with the system as a unit, rather than as a series of isolated microprocessors. The programmer has access to all words defined for all processors; processor linkage at execution time is transparent to the programmer.

References:

(1) B. H. Newcome and C. G. Enke, A Modular Twin-Bus Microcomputer System for Laboratory Automation, *Review of Sci. Instruments,* in review.

A Versatile Software Control System for a Triple Quadrupole Mass Spectrometer

C. A. Myerholtz, A. J. Schubert, M. J. Kristo and C. G. Enke
Department of Chemistry, Michigan State University
East Lansing, MI 48824

In the pursuit of developing a computerized triple quadrupole mass spectrometer (TQMS), our laboratory has developed a software control system that allows the user convenient and flexible control over the instrument. The TQMS has 30 parameters and 5 different scan modes. A convenient command structure was needed to allow the operators to take easy advantage of the flexibility provided by the high level of computer control of the instrument. The user can now control all parameters and scan modes through high-level commands. For instance, to scan quadrupole three, the user merely types in 3SCAN.

To change parameters, one can input them manually into the parameter editor or use a set of "softknobs." The "softknobs" are optical rotary encoders, which allow the user the familiar feel of an analog dial, with the convenience of digital control. Up to 5 windows can be used to monitor instrument performance in multiple regions simultaneously. Both softknobs and split screens allow easier and more accurate manual tuning.

The TQMS control system has the further capabilities to scan any system parameter and generate a linear or logarithmic display of the result, to automatically sequence experimental modes, and to maintain a data file using FORTH database management.

Implementing Forth for the Multics Operating System

Michael A. Pandolf
Cambridge Information Systems Laboratory
Multics Development Center, Honeywell Information Systems
4 Cambridge Center, Cambridge, MA 01242

This paper discusses the issues involved in creating a closed FORTH environment that executes entirely with the Multics System. A primary design issue addresses the rule that within Multics executable code is intended to not modify itself. Other than requiring individual copies of the dictionary for each user, this impacts the ability of the FORTH environment to make use of Multics supervisor support, which has the potential to prevent FORTH from performing any I/O. To a lesser extent, it is important to allow FORTH to fit comfortably within the segmented addressing architecture of Multics. Explored are some of the tradeoffs between requiring on the one hand that FORTH take advantage of segmentation and on the other hand that FORTH not require

cognizance of a specific addressing architecture. Finally, there is the desire to allow the FORTH environment to execute as efficiently as possible while utilizing the services of an operating system optimized for the PL/I language.

A Process Monitor

Hemant S. Patel
E. I. duPont deNemours & Co.
LaPorte, TX

This paper discusses a FORTH implementation for monitoring, control and optimization of a medium-sized chemical process. The 68000 based program routinely collects on-line field data, performs absolute and deviation limit checks and stores data on a hard disk for future retrieval in graphic or tabular form. Routines for lab analysis and product quality management via CUSUM technique are provided. Both field and lab data are used to optimize conversions and yields. To quickly isolate and respond to plant upsets, field variables are organized and displayed as trend charts in groups containing up to eight variables. A top-down tree structure is used so that each variable in a group above is casually related to another group below. FORTH screens are utilized as fill-in shells for customer reports and help files.

Time critical modules are written in assembler. Two byte integer format is used for mass storage with four byte integer arithmetic used for computational accuracy. The resultant package is extremely fast and allows features such as scrolling of trend charts. While non-FORTH commercial programs offer a wide range of features, all of them are hardware specific, with customizing being difficult and laborious. Because of FORTH's transportability, adaptations to other computers are relatively easy. Furthermore, FORTH's modularity makes customizing and "add-on's" quite simple.

Object Oriented Extensions to Forth

Richard Pountain
London, England

The paper will describe two iterations in the evolution of a set of object oriented extensions to FORTH-79, based loosely on the syntax and semantics of Smalltalk-80. The initial inspiration for the project was a desire to create a structured data facility in FORTH, equivalent to records in Pascal (as an illustration for a book chapter on Data Structures in FORTH). The initial idea was to use the existing CREATE....DOES> mechanism for this purpose. This end was easily achieved by writing a defining word which produces "smart" identifiers with which to address the fields of a record. Having accomplished this, the author noticed that in fact CREATE...DOES> contains the seeds of a far more powerful data abstraction mechanism, namely *classes* as in Simula or Smalltalk. It is equivalent to a class constructor without any inheritance mechanism.

State Sequence Handlers

Edward B. Rawson
Rawson Engineering
Moccasin Hill RFD 3, Lincoln, MA 01773

We describe a new type of sequence control structure which allows an ordinary : word to behave as a small state machine. The control structures may be freely mixed with BEGIN, UNTIL, etc., and with IF, ELSE, and THEN, with conventional nesting restrictions. The new structures include analogs of BEGIN, UNTIL, etc., as well as several other words which support terse, readable, state machine code.

The normal state variable is replaced by an entry pointer, but its handling is largely hidden in the source code, and application code never need supply a pointer value. Definitions may be understood as standard FORTH procedures, with the understanding that execution may "hang up" in one of the internal loops or called procedures until it is proper to proceed. Repeated calls to the main procedure result in repeated execution of the loop code or the called procedure until continuation is appropriate.

The sequence control structures encourage structured code, make state entry and exit natural, and produce fast, compact, object code. They eliminate the need to assign and manage state numbers. The structures and several applications are described.

An Idealized Game Interaction with a Knowledge Engineered Environment in Forth

Dana Redington
Sleep Research Center
Department of Psychiatry and Behavioral Science
Stanford University School of Medicine

A rapidly evolving segment of Artificial Intelligence – the field of Knowledge Engineering – is transforming computers into machines that can exhibit human-like intelligence. The illusion is that machines can now "behave" and "think" as humans do. This result dramatically impacts the quality of communication between human and machine intelligences.

Probably the most widespread example of human and machine communication is found in game interaction – a place where players accept implicit or explicit rules of behavior and act accordingly. Components of game interaction can be found in other applications. For example, a primitive level of game interaction is found in word processing where there are implicit rules of data entry, editing, and printing; an example of one of the highest levels of game interaction is found in fledgling applications of expert systems on personal computers, such as problem solvers where rules and data are defined explicitly.

A number of languages can be used to embody intelligence into user interfaces and develop high level game interactions, such as Basic, Pascal, C, and more profoundly Lisp, Prolog, and Smalltalk. However, FORTH provides a unique opportunity to transform an already extensible environment into a system that can manipulate objects, exhibit intelligence, and communicate with a human in a natural way, yet remain extensible.

This paper describes a novel game interaction exhibiting machine intelligence. The classic game of ANIMALS which has been used in the field of Artificial Intelligence and written in a variety of languages, including FORTH, is presented from a different perspective. The example illustrates the power of a knowledge engineered environment in FORTH that manipulates objects.

From Bubble to Quick: A Forth Comparison of Sorting Algorithms

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

This paper compares generalized implementations of half a dozen sort algorithms in FORTH. Examples of sorting integers and strings are used to illustrate the different methods. Even the faster sorts are coded compactly in FORTH. The sorting algorithms take item numbers as arguments, allowing users to define their own EXCHANGE and COMPARE words. Informal guidelines are given for choosing an appropriate algorithm for your application.

ROBOFORTH II: An Easy to Use Language for Programming Robot Arms Written in Forth

David N. Sands
22 Cheddars Lane, Cambridge, England

ROBOFORTH II is a robot control language written in FORTH. By making maximum use of FORTH's features the result is compact, fast, and very easy to use. It can be easily adapted to any robot arm or any 8-bit microcomputer.

As a language it is a means of communication between the human user and the robot arm and should ideally resemble a natural language. FORTH makes this possible. As with a natural language a novice can learn the easy words first and still get results. Yet the vocabulary is extensive enough to permit more complex requirements to be expressed. To the basic vocabulary of FORTH-79 (or FIGFORTH), I have added another 80 odd words which are concerned only with robotics. Other list-oriented languages like LISP and LOGO may appear to offer similar capabilities, with even some advantages over FORTH, but being bulky interpreters they are slower and available on fewer computers.

The Need for a Standard Test-Suite

Nicholas Spies
313 Grace Street, Pittsburgh, PA 15211-1503
and
Mahlon G. Kelly
268 Turkey Ridge Road, Charlottesville, VA 22901

A test-suite is a collection of programs used to find whether an implementation of a language complies with a standard. It tests whether the outcome of operations meets specifications. Ambiguities and poor wording in the FORTH-79 and FORTH-83 standards allow misinterpretation so as to cause differences in implementations that are supposed to be "standard." This undermines the usefulness of the standards. Test-suites would assure compliance with the standards (both for system authors and users). Producing the test-suites would resolve ambiguities in the phraseology of the standards and force better wording. And the code of the test-suites would provide examples to help programmers learn and interpret the standards. Although test-suites would be easier to write for FORTH than for other languages, if written now they would be subject to the same ambiguities as have created differences in implementations. Thus test-suites must be written and approved by the standards team, presumably accompanied by clarification of the standards.

NAPLPS Decoding in Forth

Dr. Billie Goldstein Stevens
CBS Technology Center
227 High Ridge Road, Stamford, CT 06905

The North American Presentation Level Protocol Syntax* (NAPLPS) is a standard for encoding both textual and graphic information for videotex. Given a personal computer with hardware (either built in or added on) that can produce a display conforming to the standard, the actual decoding of NAPLPS-encoded data can be done in software. Such software decoders have been written both experimentally and as commercial products (primarily in Canada, where consumer videotex is more prevalent than in the U.S.). NAPLPS decoding is similar to recursive descent parsing, but NAPLPS code does not always have the clean structure of a context-free language. There are awkward jumps and unexpected escapes possible. Forth lends itself readily to the task of NAPLPS decoding, gracefully handling both the straightforward parsing and the sudden shifts. An experimental NAPLPS decoder in Forth has been developed at the CBS Technology Center. We briefly describe the NAPLPS coding techniques, then explain the design and implementation of the decoder. The suitability of Forth for this task is illustrated and emphasized.

* Videotex/Teletext Presentation Level Protocol Synyax: North American PLPS. Published jointly by American National Standards Institute, Inc., New York as ANS X3.110-1983, and by Canadian Standards Association, Rexdale (Toronto) Canada as CSA T500-1983, December 1983.

Controlling an Integrated Automatic Parametric Characterization System with Forth

Jonathan Sun and Barry Mason
GTE Communication Systems
2000 West 14th Street, Tempe, AZ 85281

The modeling of semiconductor characteristics requires empirical correlation with theory. The complexity of the theory and processing variations require very large amounts of data to be gathered, which if performed manually would be very time consuming and the results prone to error. An automated system has been developed that can be universally applied to measure any desired quantity or parameters of a device (or circuit). The configuration of the system consists of an automatic wafer prober, instruments such as voltage, current and capacitance meters for measurements, an HP9836 microcomputer for system controller and the use of a VAX 11/780 for postprocessing and data storage. FORTH is used on the HP9836 for controlling the system, sending the measured data back to the VAX 11/780 for postprocessing, modeling parameters calculation and data storage. It is my understanding that this is the first time that FORTH has been used for this application. Most of the presently available parameter extraction systems are implemented in BASIC. Several unique aspects due to the selection of FORTH will be discussed, such as the easiness of the programming, maintenance and enhancement. The trade off between FORTH and BASIC will also be described.

REvised REcursive AND? 'REPTIL :IS

Israel Urieli
Ohio University, Athens, OH 45701

Since its initial presentation and demonstration in 1984, REPTIL has undergone a complete revision. It now includes many new features, such as an intrinsic recursion capability, and decompiler and editing of the object code in a structured manner without any need for source code storage. All of this magic is due to the adoption of the basic infrastructure of Token and Status Threaded Code of Buege's RTL, presented at the 1984 Rochester Forth Conference.

The emphasis of REPTIL for usage in the education field is somewhat different to that of its predecessor RTL, which was conceived as a practical high performance relocatable language for control applications. The system vocabulary is extremely readable, consistent and self-explanatory, as is exemplified by the operating system (outer interpreter) verb RUN, defined as an infinite recursive loop as follows:

```
'RUN DO:
    R.RESET / Initialize the Return Stack pointer
    READ-LINE
    DO-LINE
    RUN
:END
```

Four fundamental constructs are defined, being the deferred execution pair ( ), defining verb pair DO: :END, a powerful ?THEN ?ELSEIF ?ELSE ?END conditional structure set and a REPEAT[?EXIT]END loop structure set.

Technical Management of Large Forth Projects

William Volk
Aegis Development Inc.

What are the considerations in management of a large (over 5 megabytes source code) Forth programming project? The author, a former manager for several large Forth projects, talks about the pitfalls and perils in a large development situation.

Often technical management papers will present a series of completed, successful projects. This paper will talk about projects ranging from successes to near failures. The goal is to learn from past mistakes and to spark discussion on programmer/project management.

The role of Forth in the development environment is also important in a large project. Is an extensible language ill-suited for these projects? Should Forth be limited to small/real-time programs as its inventor suggests?

What extensions to Forth can aid in the large multi-person environment? Programming standards, extensions (local variables, objects, debuggers, and source control systems) and strong design tools may make the difference in a very large project.

New directions in the Forth development environment will also be introduced. An example of a Forth system that seems to offer excellent environment for large application development (NEON) will be presented.

Standardization: Management's Dream or Productivity's Doom

Terry T. West
Seattle, Washington

The idea that a standard brings a higher degree of "manageability" to any area of application may be a seductive myth. Standardization helpful in hardware realms does not automatically carry over into symbolic uses. This is often especially true with a language like FORTH. A standard may be positive from the implementor's standpoint but negative from that of the user. Project management can benefit from well designed, UPWARD COMPATIBLE standards. When they are not, as in the case of FORTH-79 and FORTH-83, such standards can be counterproductive.

We should note that FORTH itself was born in large part due to the strictures placed upon the user by such "standardized" languages as FORTRAN and COBOL. When standards are not compatible, we must ask what distinguishes such standards from dialects? What is the functional difference? The existence of standards implies

that a common ground exists among the standard's users, yet FORTH was invented as a tool to simplify software development in cases of EXTREME CUSTOMIZATION. In other languages standardization has occurred over decades, the standardization of FORTH could well be premature!

FORTH as an AI Language – A Challenge to LISP

Terry T. West
Seattle, Washington

LISP has been the Lingua Franca of Artificial Intelligence for over twenty-five years. LISP resources have evolved over this time into powerful tools. LISP's power is centralized upon formalization and generalization. FORTH offers structures more akin to human language and communication forms. The future will bring highly advanced peripherals and human interfaced control technology; FORTH will provide superior resources here.

FORTH also offers superior potential in efficient and dynamic data representation. LISP recursive power is convenient in many applications but is inefficient. FORTH's extreme structural definition flexibility and extensibility will prove superior over a wider class of AI applications.

We must distinguish between AI and Robotics, both of which are closely related. AI focuses upon generic behavior forms and elicited responses. Robotics centers upon specific behavior and directed responses. LISP implements these through highly evolved EVAL procedures. FORTH can substitute Both exotic vectored execution and compilation under user control. In LISP, the programmer is a user of LISP's resources, in FORTH, the programmer is a participant in the conceptual implementation.

Forth-79 Profiled

Alan Winfield, Ron Goodman, Richard Pountain
Metaforth, Ltd. Hull, England

The technique of 'profiling' an applications program, that is obtaining statistical data on the length of time that the program spends in each of its component sections, is well known and allows time critical parts to be identified and thus optimized.

Less obvious is the possibility that an entire operating system might be profiled. This is hardly practical with the majority of operating systems (even less, compiler), but the simplicity and virtual machine concept of a Forth system does lend itself to the in-depth analysis that profiling will allow.

This paper describes such an analysis for a 79-standard Forth system. Results are summarized which detail from high-level words through to the simplest code primitives, including NEXT, a number of different activities of the system. In particular, interpreting, and compiling are profiled. Finally, an analysis which combines the separate results is given showing the overall average activity of the Forth system. Some surprising results are indicated; in particular the amount of time that the system spends in the inner interpreter. Also a number of code primitives turn out to be executed unexpectedly often, while others are executed much less frequently than might be expected. These results have proved to be particularly useful in our quest for a Forth machine architecture, by indicating precisely which aspects of the hardware design need to be given special attention.