Abstracts of the Ninth Asilomar FORML Conference

Asilomar Conference Center
Pacific Grove, CA
November 27 – 29, 1987

OF3210 32 Bit Forth Engine Advanced Information

C. H. Ting

OF3210 is a new generation 32 bit single chip microprocessor supporting the high level Forth Language. The multiple stack architecture provides optimal data paths inside the microprocessor as well as data communication between the microprocessor and external memory. The stacks can be also used as randomly accessible data memory, making it possible to consolidate the advantages of both the Harvard architecture and the von Neumann architecture in a single CPU. Being a 32 bit processor, it is well suited for various applications, such as signal processing, bit-mapped graphics, image processing, artificial intelligence, robotics, and number crunching problems.

32 Bit Forth Engine Simulator

C. H. Ting

In the last volume of More on NC4000 (Vol. 4, p. 52,) I took advantage of the date which was April 1, to announce a fictitious 32 bit Forth Engine and discuss many of the dreamed features of this machine. It is supposed to be a joke on Novix, Inc. and its infamous NC6000/5000 chip which has been on the same 6 week delivery schedule for the last year. Nevertheless, there are many properties in this design that warrant more in-depth study and evaluation. One way to evaluate this design without expending arms and legs is to build a software simulator of this 32 bit Forth engine and see how it behaves.

Novix Decoder

Stephen Sjolander

Forth Inc.

Why Call it a Decoder?

Forth implemented on the NC4000P combines both high-level Forth code and native machine code within colon definitions. High-level code is invoked as subroutine calls. Displaying just the address of a particular call would not make for informative decompilation. Names must be displayed for these addresses in a manner, and using a procedure, very much like conventional decompilation. In the case of native machine code there is no place to find the text to display for a given instruction. The instructions need to be translated into text resembling the original source code. This process is very much like a conventional disassembler.

Why Bother with a Decompiler?

If one lacks the source code for the nucleus, then a decompiler is indispensable. For example, I learned Forth on a PROM-based, FIG model. It came with 16K of object code and no source code. Fortunately, it did include a simple decompiler. Assuming one has the system source code, a decompiler still can be useful as an educational and debugging tool. By looking at what object code is built from the given source code, insight can be gained into how the compiler works. As a debugging tool, it can be helpful to decompile a new definition to compare the machine's ideas of the definition to your own. Lastly, in the present case I wanted

to more closely examine the results of the optimizing compiler. To fully exploit the performance of the NC4000P processor, the optimizing compilers used with it attempt to compile machine instructions to perform the work of a phrase of Forth words rather than a single word.

68000 Binary Code Translator

Michael Saari

This paper describes a Forth program which translates binary code for the 68000 family of processors into binary code for the SPARC CPU architecture.

Soft-Wired Systems

Glen B. Haydon

WISC Technologies, Inc.
Box 429 Route 2
La Honda, CA 94020

Forth is multi-dimensioned. It is a religion, a philosophy, a software kernel, a hardware processor, an assembler, an operating system and a high-level language. (Forth Dimensions, Vol. VIII, No. 3, p. 33) A guiding principle is to have complete access and control. But keep it simple.

32 Bit Forth "Can Anyone Tell the Difference?"

Michael Murdock

Cerebranet
Torrance, CA

In the early days of Forth, in the hey day of 8-bit processors, a typical address space was not to exceed 64K. Therefore setting stack sizes to 16 bit numbers was not really seen as a limitation, but as a pillar upon which to have high level definitions be transportable. As the evolution of microprocessors from the old 8 bit days and recent 16 bit chips to today's 32 bit cousins, a need is sought to use these higher performance units in a method that meets their capability rather than just port a version of Forth from its predecessor. What is proposed in this paper is to have a future standard not be pinned down on issues of stack width and address space. Applications where 32 bit chips are big are DSP (Digital Signal Processing), Graphics, Data acquisition, machine control. Forth needs to meet the needs of 32 bit users to be assured a future in the marketplace of viable languages.

Another Attempt to Tame the 8086

Michael Perry

The prevalence of MS-DOS and the IBM-PC architecture has forced me to take another look at means of implementing Forth for that environment. This paper describes the result: a curious mixture of compromises and half-baked notions.

A Multi-Media System

Guy M. Kelly

2507 Caminito La Paz
La Jolla, CA 92037

A multi-media system has been incorporated into an 83-Standard Forth system. It allows access to a variety of different storage "devices" (including MS-DOS files), each of which starts with block zero and can have up to 65,534 blocks.

Existing storage devices may be redefined and additional devices added at compile or run time. Other file systems can also be supported.

An attempt was made to make the implementation as transparent and code-size efficient as possible.

The same syntax is used with both stand-alone and MS-DOS versions of the system.

The implementation is compatible with the traditional Forth block mechanism, with Forth blocks within MS-DOS files, and with text files.

525,000 Times Faster Than a Mainframe?

Nathaniel Grossman

Department of Mathematics
University of California, Los Angeles
Los Angeles, California 90024

How would you react if I told you that I'd just carried out a calculation on my microcomputer 525,000 times faster that it had been done on a mainframe computer? Perhaps you'd flash a wry smile and snicker politely at my foolishness. More likely, you'd try to sell me a share in the bridge that you bought from that nice fellow whom you met in San Francisco while returning home from the last FORML Conference.

I have a surprise for you: my claim is true. However, your skepticism is not unwarranted. I did not arrange for my micro to run 525,000 times faster than the mainframe. That I used Forth rather than Fortran may have contributed to the gain in speed, but the choice of the programming language was not decisive or even an important factor. What made the difference was my calling upon clever, but simple and powerful, mathematical algorithms that the mainframe programs had ignored. (in fairness, he may never have heard of them; many good mathematicians have not.) A calculation that required seven hours and one *billion* terms on the unnamed mainframe needed only 1023 terms and collapsed to 0.048 *seconds* on my 10 mhz AT-clone.

In this paper, I will briefly describe the algorithms that I used and some of the problems that they can handle. Then I will present the Forth programs that I wrote to implement the algorithms. Several features there may not appear in the corpus of published Forth code. Finally, I will present some examples of the program in action, showing how to calculate π to a moderate precision from several elegant formulas that are ordinarily considered useless or nearly so for actual calculations.

We will emphasize that our goal in this paper was not to calculate π. The goal was Forth implementations of several algorithms for speeding up the convergence of slowly convergent processes.

Neighborhood Operators for Images and Automata

Robert F. Illyes

ISYS
P.O. Box 2516, Sta. A
Champaign, IL 61820

Such diverse areas as cellular automata, image processing and finite difference equations, which resemble automata and use operators like those of image processing, involve repeated operations on small neighborhoods of elements of matrices. Ordinary indexing can be used to efficiently access the elements of these neighborhoods, but the resulting code tends to be unreadable. The extensibility of Forth permits the embodiment of this efficient procedure in a simple and readable notation. The notation will be demonstrated by the solution of the finite difference wave equation of a membrane.

Portability and the Bitness of FORTH

Mike Elola

1055-102 N. Capitol Avenue
San Jose, CA 95133

Dr. Ting has said that Forth can easily support 32-bit processor just by increasing the width of a FORTH cell to 32 bits. Then, without the hardship of a new Forth that is not backward compatible, 32-bit processing can become the norm for a particular Forth implementation.

If FORTH's bit-width insensitivity (for cells) can answer the needs associated with 32 bit processors, then there ought to be a way to define data structures so that they will also be bit-width insensitive. So inspired was I by this revelation that I developed the code to support this contention.

Some Proposals for Strings in Forth

Robert L. Smith

I have been experimenting with strings and related areas in Forth, and have a few observations and suggestions. I am sure that not all of these suggestions are original. First of all, in the normal use of words like ." the delimiter of the string is a " instead of a space, as one would otherwise expect. In Forth, one quickly becomes so accustomed to the use of at least one space as a separator that comment strings and ." strings almost invariably are followed by the delimiter and then a space. For my first proposal, I suggest that this option be made a requirement. The first result is that strings, along with normal Forth words must be separated by at least one space.

Loops and Conditionals in LaForth

Robert L. Smith

This paper describes the current state of conditional and loop structures in LaForth. The most obvious addition is a simple form of the Case statement which is designed to meld in a natural way with the rest of the LaForth conditional notation. An F83 implementation of the conditionals is given so that the reader may experiment with them.

Loops and Conditionals in LaForth

Robert L. Smith
LaFarr Stuart

For many years, LaForth has been a private and highly experimental version of Forth, used and modified only by us. There have been only a very few published papers and talks about LaForth, in part because we did not wish to conflict with FIG-Forth or the standardization effort. Some of the results of LaForth have, nevertheless, had some influence on other Forth systems. We have recently agreed to release a version of LaForth for others to use. We anticipate that its main use will be for experimentation. The strength of Forth is its simplicity, and in that sense we believe that LaForth is more Forth-like than Forth! It is very easy to modify. It is an excellent test bed for new ideas. It is a "lean and mean" type of a Forth system, in contrast to some of the recent "fat" Forths which have come into vogue.

Interpreting Control Structures — The Right Way

Mitch Bradley

Bradley Forthware

A very simple modification allows the Forth interpreter to execute conditionals and loops in interpret state as well as in compile state. Interpreted loops run at the same speed as compiled loops.

Meta–Words in Forth

Tom Hand, Ph.D.

Florida Institute of Technology
Department of Computer Science
Melbourne, Florida

This paper describes a set of meta–words for the Forth environment. Included in these meta–words are a set of vocabulary management words. The vocabulary management words give the user complete control for specifying the search order through the vocabularies. The remaining meta–words are those words that should always be immediately available.

The vocabulary management words form an extension of both the Forth-79 Standard and the Forth-83 Standard. Most of the experimental proposal of Bill Ragsdale, *Search Order Specification and Control* in the Forth-83 Standard, is incorporated in this system. The meta–words in Forth give additional flexibility to the Forth environment.

The Visual Command Interface

David W. Harralson

MEPHISTOPHELES Systems Design
"Devilishly Good"
3629 Lankershim Boulevard
Hollywood, CA 90068-1217

An advanced windowed operator interface and program development environment for the IBM PC market using FORTH.

FORTH Control Structures for the Language Requirements of the 1990's

David W. Harralson

MEPHISTOPHELES Systems Design
"Devilishly Good"
3629 Lankershim Boulevard
Hollywood, CA 90068-1217

The existing Forth control structures are considered one of the weak points of the language. There have been many proposals to give one or more of the existing control structures additional capabilities or to add new control structures. These proposals add up to seven disjoint control structure types and more than 39 Forth words.

The existing set of ten words is all that is necessary in a structured programming theoretic sense.

At FORML-85, a paper was presented that defined what a completely general control structure was, and defined the same basic Forth control structure words to implement the general control structure. A limitation of this paper was that the word set defined was not FORTH-83 compatible. Interest in the basic concept was high enough that a working group on control structures was formed to remove the deficiencies.

A successor paper was presented at FORML-86 in the form of a proposed change to the FORTH-83 standard and has been accepted by the Forth standards committee as an experimental proposal.

This paper will present the following:

1. An overview of the proposed control structures rather than distributing the previous paper.
2. An implementation history.
3. Implementation cost.
4. A working implementation of the authors system.
5. Comments and reflections on the new control structures.
6. Floppy disks with the control structures in a DTC F83 system and stand-alone.

Named Local Stack Variables as Modules

Guy M. Kelly

Kelly Enterprises
2507 Caminito La Paz
La Jolla, CA 90237

There have been many papers about local stack variables. This one extends the concept to include modules. A module is created by a stack picture (external to and preceding a set of words which use the stack variables names in that picture). The module is ended by a final word which includes the same stack picture and uses the words defined between the external stack picture and that word.

Field and Record Structures

Stephen Pelc

MicroProcessing Engineering
133 Hill Lane
Southampton So1 5AF
England

This article shows how field and record structures as used in other languages can be provided in Forth. Records are used as templates, and may consist of fields, subrecords, and variant types. Extensions are demonstrated which allow for compile-time evaluation of complex field expressions. All code is in Forth-83.

Data Structure Unification

James C. Brakefield

KRUG International
Technology Services Division
406 Breesport
San Antonio, TX 78216

The theoretical framework for Forth is built on the concept of the address which is both executable and a data item. This along with the two stacks provides a foundation for the procedural aspect of the computer programming.

This paper is an attempt to provide a better theoretical foundation for the data structure aspect of computer programming. The basic approach is to embellish the data objects such that all objects are members of a single class.

This class of data objects supports a superset of the facilities found in a number of other languages.

Network of Neurons

Robert E. LaQuey

A neuron may be modeled as a threshold logic element with memory.

Networks for such elements are of interest for at least two reasons:

1. as models of brain function
2. as computing systems.

VLSI technology makes it possible to implement large networks of identical computing elements (CEs). This capability can only increase in the foreseeable future. Thus networks of computing elements may well come to be of extreme practical importance.

Neuralizing SAS: Neural Variants of Signal Space, Address Space, and Symbol Space

James C. Brakefield

KRUG International
Technology Services Division
406 Breesport
San Antonio, TX 78216

The "Brakefield" framework for computer science can be embellished to encompass the "neural" or "connectionist" approach. The principal characteristics of the "neural" approach (use of noise, local computation, relaxation, and emergent properties) are matched with corresponding features in each space.

Bayesian statistics is a possible "neural" variant of symbol space.

A modeling context is used to derive a possible "neural" variant of address space (Forth).

HyperForth = Hypertext + Forth

Robert E. LaQuey

Text is the most common form of data. Hypertext technology provides a simple but powerful means of organizing access to text. Forth is ideally suited to the implementation of Hypertext technology. Computationally intensive set operations are very useful in Hypertext technology, and the Forth chip can provide the throughput demanded by these repetitive operations. Finally Hypertext technology provides a starting point for automated acquisition of knowledge from text, thus providing a substrate on which to build a machine capable of text understanding.

Multiple Code Fields: Object–Oriented Programming in Forth

George W. Shaw
Forth Standards Team Referee
Director–elect Asilomar FORML Conference
Chairman, Silicon Valley Chapter of FIG

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

Often there is a common set of operations performed on a given group of structures. Each class of structure within the group may require its own set of manipulation operators, cluttering the language with operator names surrounding a given theme. For example, many systems have most or all of the set **@**, **2@**, **B@**, **C@**, **D@**, **N@**, **P@**, **PC@**, and **S@** as well as their corollaries **!**, **2!**, **B!**, **C!**, **D!**, **N!**, **P!**, **PC!**, and **S!**. Additionally, after almost every usage of a data structure is compiled an operator to define the action upon that structure. Considering the proliferation of operators, this is sometimes confusing, somewhat error prone, and almost always wasteful of memory.

The common uses of multiple code field words have been limited to **QUAN**s (a variable/constant type word), **LQUAN**s (a variable/constant type word in external memory) and **VECT** (to define a vectored execution variable). Other uses have been to allow unique vocabulary structures by giving each vocabulary its own search routine. All of these hardly touch upon the possibilities. The typical limiting problem has been to have a generally usable syntax for defining multiple code field words in high level and/or machine code.

The advent of such syntax allows for previously unforeseen possibilities for Forth. Generic "smart" operators can be defined which require no smarts at all because the "intelligence" is built into the data structures. Forth thus receives capabilities much like object-oriented languages, with the ability to manipulate many varied data structures (objects) with a few simple logical operators (methods). Additionally, the runtime references compile to less memory and execute faster than the previous dumb operator/reference pairs.

Bach Organ Recital on a Six Channel PC Organ

Dr. C. H. Ting

Offete Enterprises
San Mateo, CA

This program will include the following pieces:

Capriccio, On the Departure of his Beloved Brother
Wenn Wir in höchsten Nöten sind
An Wasserflüssen Babylon
Nun danket alle Gott
Komm, Gott, Schöpfer, heiliger Geist
Von Himmel hoch komm ich her

If time permits:

Three Little Suites from Anna Magdalena's Notebook

"Write Once, Read Never"

Wil Baden

Doeiz Networks, Inc.

Forth has a reputation for being difficult to understand. It has been accused of being write-only code. Of course, any programming language can be used in the write-only mode, but some languages suffer from the disease more than others, *e.g.* APL, LISP, C, and Forth.

St. Francis' Terminal Input

Wil Baden

Doeiz Networks, Inc.

Terminal input is defined to support:
Command line editing,
Instant menu selection,
Source-code line editing,
Source-code screen editing,
User-defined text macro keys, and
User-defined function macro keys.

Restarting Forth

Wil Baden

Doeiz Networks, Inc.

The Forth command line interpreter is `QUIT`. It is the default Forth application.

A Network Manager and Controller — Experience with Separate Compilation

Stephen Pelc

MicroProcessor Engineering
133 Hill Lane
Southampton SO1 5AF
England

This paper describes our experience during the design and development of a large project using a Forth system featuring separate compilation of modules as a fundamental part of the Forth programming environment. A network system is intrinsically a message passing system, and this idea is retained in the software design. We explore the design freedoms and constraints caused by these mechanisms and the benefits or otherwise.

CAD Command Language

Mitch Bradley

Bradley Forthware

We used Forth as the command and macro language for a Computer Aided Design system. This command language is based on a number of novel interpreter techniques, including a single control structure that serves both as a loop and as a conditional.

Military Forth

John D. Carpenter

The Ada language includes a base package called "Standard" [BOOCH87]. This is mandatory in all Ada development systems. There are other Ada packages that are required only if the application needs the features that are supported in any of them. The "Standard" package contains a very small number of

instructions and therefore provides the inspiration for this paper. To provide basic support for a Forth variant of words which would encompass the usefulness of the "Standard" package, one's attention is brought to the Vax Instruction Set which is the assembly language for the VAX series of computers manufactured by Digital Equipment Corporation [DEC84]. This instruction set includes facilities for word sizes up through 128-bits as well as facilities for floating point. This paper introduces for philosophical consideration and inspiration two vocabularies: **MILITARY** which is like high level Forth, but includes data typing and floating point and **INSTRUCTIONS** which is like low level Forth, but includes handlers for various word sizes as well as, again, floating point. The paper consists for the most part of a comparison of existing Ada instructions and equivalent Forth words as well as existing VAX instructions and their Forth word equivalents. The suitability of this would be in an aerospace environment which would have a Forth language that has the **INSTRUCTIONS** and **MILITARY** vocabularies plus whatever vocabularies that correspond to other Ada packages listed in the full Ada standard. It would be possible to research and explore requirements for a service or product with this Forth and then incorporate a Forth to Ada translator for the delivery of the product or service once the critical requirements have been well defined.

Milking Forth in a Conventional Software Development Environment

Andrew J. Korsak, Ph.D.

Consultant, GO FORTH>> ®
504 Lakemead Way
Redwood City, CA 94062

During the past twelve months, while working in a software development department that utilizes "orthodox" programming languages, I have managed to "milk" the capabilities of Forth whenever possible, resulting in a great time and money savings for the company that contracted me for the work. This paper describes the various ways in which I utilized Forth to expedite software development. These included using an umbilical interpreter running test code on an INTEL 8751 microcontroller interfaced by parallel I/O to a PC clone, prototyping modules on a PC that were later translated to target assembler code, and miscellaneous tasks such as stripping a "list" file back down to a source file on a PC and uploading it back to a VAX.

Chinese Limericks

Dr. C. H. Ting

Offete Enterprises
San Mateo, CA

Chinese is the best language to build limericks because words are single syllable characters, which can be easily arranged to form fixed length verses with symmetry in rhythm and in rhyme. All I need to implement Chinese limericks is a good Chinese word processor which has the character fonts and can be accessed by Forth. Such a Chinese word processor is recently available for the IBM PC. I keyed in about 1000 verses of five word poems, in the so called "old style", by Li Po, the most respected poet in Tang Dynasty. Picking verses from this data base allows be to construct limitless limericks.

The Simplest Line Drawing Routine

Dr. C. H. Ting

Offete Enterprises
San Mateo, CA

In the past ten years, I have studied and explores various ways to draw straight lines on screens, including the famous Bresenham algorithm. None of them were satisfactory. I could not accept the fact that it has to take eight screens of code to draw a straight line, which any three year old can do on a wall. This line drawing routine takes only 5 lines of Forth code! It took a little bit of insight mixed with a little bit of recursion to arrive at this very simple solution.

R&D Forth

Ray Gardiner

Ardmona Fruit Products Cooperative
Mooropna 3629
Australia

R&D Forth is an NC4016 FORTH specifically designed for writing PROM based control systems. It was written by Dean Perry and Ray Gardiner of Ardmona Fruit Products for NC4016 based hardware designed by Ray Gardiner. When completely finished, it will be public domain, as it owes much of its existence to other public domain FORTHs.

underSTANding natural language

Tom Hand, Ph.D.

Florida Institute of Technology
Department of Computer Science
Melbourne, Florida

This paper describes some aspects of STAN, a natural language system implemented in Forth, that is under development at Florida Institute of Technology. STAN is primarily a frame-based system that has roots in both conceptual dependency and case grammar theory.

"Nights on the RoundTable" or "How I spent my summer vacation"

Dennis Ruffer

Lead Sysop
Forth Interest Group
RoundTable on GEnie

For the past seven months (going on three years) I have been involved with the establishment of a Bulletin Board, dedicated to the Forth language. Now that we are finally "alive" on the GE network for Information Exchange, GEnie for short, it is probably time that I try to set down some of the history of how we got here, what we had in mind, and where we are going in the future.

Forth in a Revolving World

Tom Zimmer

How does the Forth language apply to the programming language community today, is it evolving to become competitive, or devolving to extinction? What can we do to make Forth competitive and prevent extinction?

The Forth Year in Review

Martin J. Tracy

Forth, Inc.

This last year has been a busy one for Forth. I like to believe that the "Forth year"starts and ends with FORML (FORth Modification Laboratory) at Asilomar in Northern California over the Thanksgiving weekend. Meanwhile, here's what's happened since the last FORML (86). Some of this material will appear in the *Dr. Dobb's* "Forth Column" (Feb. 88). I apologize in advance for missing anything.