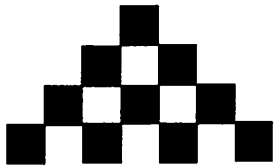
Definitions

The Extensible Languages Publication

Volume I, No. 2
November/December 1993



SILICON COMPOSERS INC

The SC32 Line-Up

SC32™ 32 bit Forth Microprocessor

- 8 or 10 MHz operation.
- 1-cycle instruction execution.
- Multiple Forth words per clock cycle.
- Contiguous 16 GB data and 2 GB code space.
- Stack depths limited only by available memory.

SBC32 (Single Board Computer32)

- 64 KB to 512 KB 0-wait-state static RAM.
- 100mm x 160mm Eurocard size board.
- SC/Forth32 Forth-83 based system included.

PCS32 (Parallel Coprocessor System32)

- Full-length PC/XT/AT plug-in board.
- 64 KB to 1 MB 0-wait-state static RAM.
- SC/Forth32 Forth-83 based system included.

DRAMIO32 (DRAM, SCSI, Serial, Parallel)

- Eurocard size, plugs into SBC32 or PCS32.
- Holds up to 16MB on-board DRAM.
- 16-bit parallel, 4 serial ports, time/date.
- Interfaces up to 7 SCSI devices.
- Software drivers in source code form.

FAD32 (A/D Converter, WatchDog Timer)

- Eurocard size, plugs onto SBC32 or PCS32.
- 16 channels, 12 msec, 12 bit plus sign A/D.
- Voltage generation, wdog timer, proto area.
- Software drivers in source code form.

655 W. Evelyn Ave #7, Mtn View, CA 94041 (415) 961-8778

ENCODER PRODUCTS COMPANY

Synergy Plus - An Intelligent Distributed Control System

+ Hardware

AIO 1218 - 8 inputs, 2 outputs, 16 digital I/O, PID routine
DIO 1248 - 48 digital I/O, +5 vdc logic
CTR 1224 - 2 high speed 24 bit counters, 16 digital I/O, 3 modes
SIO 1232 - 2 serial I/O, 16 digital I/O, RS-232/RS-485 options
RCM 1244 - Byte FIFO interface to IBM (Forth, Lotus, Windows)
* MCB 1290 - 2 axes of servo control, slaving, fractional ratioing

* MCB 1290 plugs directly to any baseboard module via SBX
All modules include SBX interface, Bitbus communications, Diagonostic LEDs, Watchdog, and many more standard features

+ Software

FREE!

Multiasking operating system (DCX pre-emptive)
Resident interactive programming kernel (Forth-83)
Resident development interface and editor
Disc-based library of application utilities
- Floating point arithmetic
- dynamic memory management
- state language programming
- message passing
- object oriented programming
- many more available for your unique application

+ Integration Services - Turn-Key Systems

System specification
- System design
- Programming
- Installation

1601B Highway #2 • Sandpoint, ID 83864

Call 800-366-5412

VENDOR'S COLUMN

New 68HC11 Interactive Cross Compiler

FORTH, Inc.

FORTH, Inc. announced a new release of its chipFORTH interactive cross-compiler for the Motorola 68HC11 family of processors. The new release incudes extensive optimizations and improvements, including a 32-bit "big model" host to support large target programs. chipFORTH is widely used for instrumentation, process control, industrial sensors and other applications.

Elzabeth Rather, President of FORTH, Inc. notes:

Perhaps the best-known is Federal Express' Supertracker, a hand-held device used by over 50,000 FedEx agents world wide as the primary input device for their award-winning package tracking system.

chipFORTH allows an engineer to interactively develop and test high-performance embedded applications for the 68HC11 using an IBM PC as a host. chipFORTH connects to the target 68HC11 through the chip's serial port to provide interactive, incremental programming without an in-circuit emulator or other extra hardware and software. The package includes the pF/x multitasking executive and polyFORTH development environment running under MS-DOS on a PC host, plus all the tools necessary to develop and test applications for the target. These tools include a high-level FORTH cross-compiler, cross-assembler and a complete target nucleus with hundred of primitives. All of these, plus the pF/x executive are also supplied in source form.

The runtime library includes integer and fixed-point fraction math, which offers floating point-like capability without the speed penalty; string handling; clock and calendar; and input and output number conversion. chipFORTH utilities and programming aids include a string and full screen editors and PROM programming support.

The product includes a development board that provides 8K each ROM and RAM and dual RS232 ports. chipFORTH is configured to download and run a test application on this board; a simple procedure is provided to let the user reconfigure chipFORTH for custom hardware and target address space. The 68HC11 chipFORTH package is offered at $1,995. for software, hardware, documentation and telephone hot-line support. Versions of chipFORTH are also available for the 8031/51, 80x96, and 68xxx families of microcontrollers.

For more information, please contact:

FORTH, Inc.
111 N. Sepulveda Blvd.
Manhattan Beach, CA 90266
(800) 55-FORTH or 310 372-8493.

Definitions *Volume 1 No. 2*
November/December 1993

Definitions is published 6 times a year

Table of Contents

What is Forth technology?
How do I program my laser printer in PostScript?
What language really works for rapid prototyping?
How do I find consultants or programmers?
Who really understands hardware and software?
How can I make Microsoft Windows work for me?
What works in real time for embedded systems?

These and more questions are answered in **Definitions.** Subscribe Today!

Subscriptions are
$25/year in North America, and
$35/year outside North America

Definitions is distributed bimonthly by the Institute for Applied Forth Research, Inc., 70 Elmwood Avenue, Rochester, NY 14611. Definitions is published and edited by Lawrence P.G. Forsley. Circulation and advertising is handled by Brenda J.G. Forsley. Send inquires to us at that address, or at our phone: (716) 235-0168.

Advertising in this issue

1994
Rochester Forth Conference
on
Rapid Prototyping
June 22nd - 25th, 1994
at the
University of Rochester

Call for Papers

There is a call for papers on all aspects of Forth technology, its application and implementation, but especially as relates to **rapid prototyping**. Rapid prototyping allows fast turnaround from product conception through design into manufacturing and through maintenance: the complete product cycle. It has become increasingly important with ever shortening product life cycles and design times. Interactive, extensible languages offer the most potent, and cost effective, rapid prototyping tools available.

Other sessions will cover standards, related languages, applications and new processors. Standards include the ANS X3/J14 as well as up and coming standards like Open Boot. Related languages include Keithley's ASYST and Adobe's PostScript.

Submit a 100 word abstract by April 15th and a final 5 page paper by June 1st. Type should be no smaller than 8 point. Author's kits will be sent indicating preferred format. Longer papers will be considered for submission to the refereed Journal of Forth Application and Research.

The Conference

Vendor Exhibits
Training seminars, beginner through advanced
Poster sessions
Programming in PostScript
X3/J14 ANS Forth Standard
Forth vs. C vs. C++
Forth under Windows
Object oriented technologies

The fourteenth annual Rochester Forth Conference will be held at the University of Rochester in Rochester, New York. Registration, training seminars and vendors booths will begin the afternoon of Wednesday, June 22nd. On campus housing is available. **Attendance will be limited.** Register early.

For More Information, contact:

Lawrence P. G. Forsley
Conference Chairman

Forth Institute
70 Elmwood Avenue
Rochester, NY 14611 USA
(716)-235-0168 • (716)-328-6426 *fax*
EMail: Genie.....L.Forsley
Internet.....72050.2111@compuserve.com

with the same stack diagram. It was suggested that TypedForth could be a higher level subset of Forth allowing developers to drop into low level Forth and use words returning variable numbers of arguments as long as the end result was type safe.

Other safety issues were raised as well. It was generally conceded that denying programmers access to the internals made C and C++ safer languages than Forth. In an era where the developer is increasingly the limiting factor, safety is more of an issue. It was suggested that use of the return stack for DO LOOPs and intermediate storage were inherently unsafe. A safer, higher level Forth which denied access to the return stack would help to discourage common errors. A separate DO LOOP and local storage stacks would allow the return stack to be used by source level debuggers. A case can be made for a higher level Forth in which all variables are named local variables and no stack manipulation is allowed. As above, words in this language can be written in low level Forth with full access to the stack.

Memory management is another issue. While the standard provides for the standard C memory management words: malloc and free, these are in the optional word set. Current Forth memory management using ALLOT and FORGET is extremely limited since the only way to free memory is to free all memory allocated after a particular point in the dictionary. In programs using dynamic creation and destruction of objects, this cripples any effective memory management. In general it is unclear when or if memory allocated from a system will be released. The standard Forth scheme of allocating space in the dictionary is useless in a system where frequent memory allocation random release are used.

There are ways that C and Forth could co-exist in the same program. A hybrid program might take advantage of the speed, safety and close ties with the operating system available with C while using the interactive and extensible nature of Forth to accomplish tasks that neither language could achieve on their own. Currently there are several versions of Forth implemented in C where the Forth systems executes Forth within a C shell which manages interaction with the operating system. It as also suggested that a Forth which understood the mechanisms of dynamic linking could act as an integrator for C routines available in dynamic libraries.

Cooperative hybrid programs raise several issues. Most important is who is in charge? Does the C program use Forth for well defined tasks while retaining control or does Forth retain control but execute some or most of its words as C routines? Development of good hybrid systems require programmers familiar with the details of implementing both C and Forth applications. It will be a major challenge. It is clear that the C community will not be interested in exploring such solutions until the Forth community demonstrates the utility of the approach.

Several issues are raised in this discussion relative to some of the basic tenets of Forth. How important is it that words be allowed to return a variable number of arguments? Could a Forth be built where all arguments were accessed through local variables? What would the effect of making the dictionary dynamic so that words might be randomly undefined? This would be possible in a tokenized Forth. What would be the effect of making redefinition of a word global rather than incremental?

In the third and concluding part of this article we will look at systems where this has been done, including hybrid Forth and C systems.

[1]Excerpted from "Working Group on C and C++", Proceedings of the 1992 Rochester Forth Conference on Biomedical Applications, pp 113-115. Chaired by Dr. Kent Brothers and written by Horace Simmons.

Forth, C and C++[1]

This is part 2 of a 3 part article examining Forth, C and C++

While C compilers can produce code that is faster than Forth this was not seen as a critical advantage. Very little of the code in a large program is executed often enough to make significant differences in speed. Both C and Forth allow the developer to optimize critical sections of their code. For the vast majority of large programs today's processors provide adequate speed. In the future, speed will become even less of an issue.

Forth and C
Part 2 of 3

C has a well established and very uniform collection of libraries for many functions developers might wish to perform. Because C and Unix developed together, almost all C compilers offer emulation of library functions available under Unix. These functions include routines for managing files, manipulating strings and managing memory along with hundreds of other functions. Individually, many of these library routines would be simple for a developer to code. Together they form a vast array of working, tested functionality on which the developer can rely. The C libraries illustrate the down side of the common characterization of Forth: 'The good news is you can write anything you want; the bad news is you have to". The new ANSI standard proposes many of the C library functions as optional routines. This is a step in the right direction, but C programmers can count on these libraries to be available on any system. Development and publication of code implementing a number of standard library functions would be a major contribution.

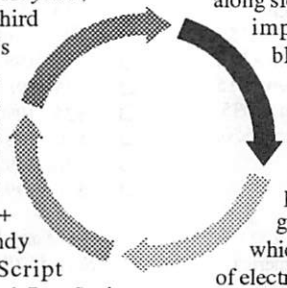
Another advantage of C++ and modern C is the use of prototypes and local variables. The most common errors in Forth involve mismanagement of the stack. Such errors come in two forms. First, programmers misuse the stack by failure to adequately track the location of items on the stack. A Forth word might start out with A B C on the stack. At some point later, the developer believes that the stack holds: B A A whereas in fact it holds B C A. or B A. A second, more subtle error involves misunderstanding not the location of items on the stack but their type. Imagine the effects on a program running on a 32 bit system if the programmer believes that a variable on the stack is an address when it is a number or the programmer believes the stack holds a number when it holds a 32 bit float. Errors of this type were seen to represent a significant majority of errors found in Forth code. In modern C and Pascal programs such errors cannot occur. Developers access the stack only through named local variables and thus cannot make errors in stack access. Function prototypes causes the compiler to test the type of variables passed to each function. As a result, the compiler will catch errors in argument type. These features mean that a C compiler will catch most errors that a Forth programmer will have to test at run time.

In a strongly typed Forth system, stack comments would be active and typed. It would require all words to supply the compiler with a description of the number and type of stack arguments. The compiler would check to make sure that type and number of arguments were preserved. This system would require eliminating all words which return a variable number of arguments such as ?DUP, substituting ?IF. It would also require that all branched IF statements return

Definitions
the extensible languages publication

What's Up?

Definitions is well on its way with its second issue. *Observation: Factoring and Complexity* looks at a different way of programming, and thinking: even beyond the bounds of object oriented programming. We have three Vendor's Columns this issue: one following Randy Dumse's latest 68HC16 based-board at *New Micros, Inc.*, an *Interview with Dr. Glen Haydon, Mountain View Press*, and a third regarding FORTH, Inc.'s chipFORTH for the 68HC11. The continuing work at the John's Hopkins Applied Physics Laboratory by Marty Fraeman, John Hayes and others is documented in a *33+ MHz Forth Processor*. Randy Dumse ponders PostScript Programming with an actual PostScript program in *Postscript vs. Forth*. Dr. Nick Solnsteff completes his series on the history and background of extensible languages with *Palaeography of Extensible Languages*. Finally, Horace Simmons and Dr. Kent Brothers continue their discussion from last issue with *Forth, C and C++*.



The 1994 Rochester Forth Conference is coming up in June at the University of Rochester. This year's theme is Rapid Prototyping. Anyone trying to bring a product to market in the rapidly changing, global market knows how important shortened product life cycles have become. Extensible languages, with resident, interactive operating systems on embedded products as well as interactive languages under host operating systems provide a rapid way to market.

For many of you this will be your last issue, *unless you subscribe now!* In the next issue we will look at Open Boot, interview Chuck Moore and discuss his latest microprocessor developed in conjunction with Dr. Ting, continue PostScript programming, observe Forth on Trial, watch Forth in near space and more. If you have comments, please write. If you are interested in our Vendor's Column or Consultant's Spotlight, call us. We're here to support you!

As editor of this publication I have found that although we are primarily for the extensible language community, we run right along side hardware. This is unique and important. There is a gradual blending of the interface between hardware and software, and no languages better support that crucial interface than the extensible languages. Definitions is proud to be in the forefront of a growing technological wave. One which will impact not only the realm of electronics, but may effect every area of human endeavor. Perhaps, with the right push, we'll see software and hardware replaced by smartware!

Join us!

Subscriptions are $25/year in North America and $35./year outside North America.

Name: _____

Address: _____

_____

Phone: _____

Payment: ☐ check

☐ MC ☐ VISA

ACCT# _____

Expires _____

Journal of Forth Application and Research *(JFAR)*

JFAR is the only refereed journal devoted to Forth-like languages. Special topic issues have included object oriented programming and real-time expert systems. *Call for a complimentary copy!*

JFAR Issue	#1	#2	#3	#4
Volume 2		❑	❑	❑
Volume 3	❑	○	❑	❑
Volume 4	❑	○	❑	❑
Volume 5	○	❑	❑	❑

❑ = $15.00 , ○ = $20.00 (Conf Proc)

Full Sets:

Volume 1		❑ $30.
Volume 2	(3 issues)	❑ $40. *Save* $5.
Volume 3	(4 issues)	❑ $50. *Save* $15.
Volume 4	(4 issues)	❑ $50. *Save* $15.
Volume 5	(4 issues)	❑ $50. *Save* $15.
Volumes 1-4		❑ $170. *Save* $35.
Volumes 1-5		❑ $220. *Save* $50.

JFAR Subscriptions:

Volume 6
- ❑ $60.00 Individual, USA
- ❑ $65.00 Individual, North America
- ❑ $75.00 Individual, Europe/Asia
- ❑ $145.00 Corporate, North America
- ❑ $160.00 Corporate, Europe/Asia

Note: No additional shipping charges on Subscriptions.

Order Form:

Name_____

Address:_____

_____

City:_____

State:_____ZIP: _____

Phone Work:_____
Home: _____
FAX: _____

Rochester Forth Conference Proceedings

The international Rochester Forth Applications Conference has been held at the University of Rochester for the past 13 years and has been attended by as many as 175 people from around the world. Most Conference Proceedings consist of long papers by invited speakers addressing the Conference theme and as many as 50 shorter papers on all topics of Forth application and implementation.

- ❑ $25. 1981 ***Forth Standards***
- ❑ $25. 1982 ***Databases & Process Control***
- ❑ $25. 1984 ***Real Time Systems***
- ❑ $20. (Vol.3,#2) 1985 ***Software Productivity***
- ❑ $20. (Vol.4,#2) 1986 ***Real Time AI***
- ❑ $20. (Vol.5,#1) 1987 ***Comp Architecture***
- ❑ $25. 1988 ***Prog Environments***
- ❑ $25. 1989 ***Industrial Automation***
- ❑ $25. 1990 ***Embedded System***
- ❑ $30. 1991 ***Automated Instruments***
- ❑ $30. 1992 ***Biomedical Applications***
- ❑ $30. 1993 ***Process Control (fall '93)***

Set of any 4 or more Proceedings
Save $5./Proc., up to $60.!

- ❑ $20.00 Forth Ref Bibliography, 3rd Ed
- ❑ $25.00 1988 ASYST Conference Proc
- ❑ $62.00 *Stack Computers: the New Wave*
  By: Dr. Philip J. Koopman, Jr.
- ❑ $50.00 *Scientific Forth*
  By: Dr. Julian Noble(with software on disk)

Purchase Subtotal: _____
Shipping Subtotal: _____
($5./book,15 Max. North America)

Grand Total: _____
❑ MC ❑ VISA ❑ Check
Card #_____
Expiration Date: _____

Institute for Applied Forth Research, Inc.
70 Elmwood Avenue
Rochester, NY 14611
(716)-235-0168 voice (716)-328-6426 fax
72050.2111@compuserve.com email

the interpreter is distributed throughout the object space! Data objects are instantiated by means of *defining words* which are the analogues of type identifiers in conventional languages such as C or Pascal.

In common with other extensible languages, there is no rigid distinction between compile-time and run-time phases. As can be seen from Fig. 2, processing of Forth statements can move arbitrarily from one to the other and back again. The symmetry of the diagram is very striking.

DEFINITION OF NEW DEFINING WORDS

New data structures can be easily defined in Forth with the help of the **create ... does>** mechanism. As there is no syntax to consider and the semantics (the individualized new part of the interpreter) can be specified in terms of Forth itself, the data abstraction mechanism is both very elegant and simple.

Forth programming thus entails the creation of a language for the application (building of *objects* to represent the problem environment and the means to make them *perform* as required) and this language becomes the application.

The creation of new control structures which proved to be a major stumbling block in the case of extensible languages, can be easily done in Forth using the standard word-definition mechanism. It is only necessary to make the new word **immediate**, i.e., executable even during the compilation phase.

CONCLUSION

Forth is an extensible language par excellence and has found a niche in the evolutionary schema of programming languages. Like the coelacanth, it will probably outlive a lot of its contemporaries!

— Nicholas Solntseff
McMaster University

(THE END)

[1] W.I. van der Poel and I. Maarsen (eds.), *Machine Oriented Higher Level Languages*, North Holland Publishing Co., (1974), 535 pp.

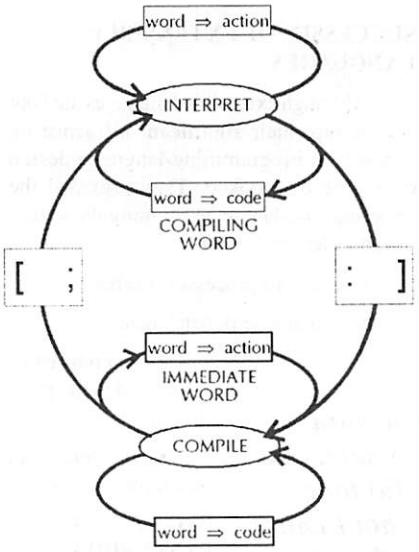


Figure 2. Forth's Compile/Interpret Cycle.

FORTH

Forth as a programming language has all the features of an extensible system. In this, it is aided by the complete lack of syntax – a Forth program is a sequence of *words* which are compiled into a (threaded) sequence of *procedure addresses* (or code addresses). The Forth symbol table or the *dictionary* is available to the user at all times which is one of the distinguishing features of extensible languages.

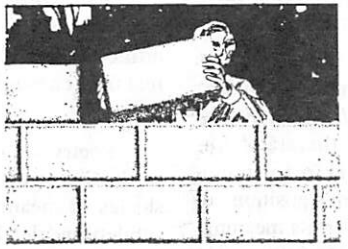
The Forth code-address or "inner" interpreter is *very small* and *efficient* as threading leads to "direct" execution of procedure code. In the case of some architectures, the inner interpreter can be coded into a single machine instruction!

What makes extensibility almost trivial in Forth is that every data object *carries its own interpreter* with it as every word contains a field which contains the address of the code that provides the semantics. In other words,

OBSERVATION:

Factoring and Complexity[1]

Imagine, if you will, a brick: a single brick. Hold it in your mind's eye. Add a second brick. What is its relationship to the first? Add a third brick. Keep them all equally in your mind's eye. How do you arrange them? Add a fourth brick. Don't let the edges become fuzzy. Keep your focus on all four bricks equally. Has their orientation changed as you've added the fourth? Add a fifth brick. Are they becoming harder to keep equally in focus? Add a sixth brick. What is the new orientation? Is your focus shifting from one to another? Are you losing the individuality of each brick? Add a seventh brick.



Most people find this exercise difficult by five or six bricks, regardless of how they arrange them. A psychologist, Dr. Miller[2], conjectured that people can keep seven, plus or minus two, independent objects in mind at the same time before they need to be categorized, or *chunked*. At seven or so bricks, most of us fall back on a brick wall, or a pile of bricks, where any number of bricks can be represented or queried individually, but we can't have the bricks and their individuality simultaneously.

This is how people naturally categorize the world around them. This isn't how programming languages are intended to be used, nor, how code is organized. Programs written in languages, like C and FORTRAN, are thought of in terms of lines or pages of code. There is no correspondence between how we organize items and the organization most programming languages force upon us.

Forth is an exception. Most good Forth code fits on a single line. Curiously, a line holds five to nine words between the onset of a colon definition and its ending semicolon. There is a good impedance match between human thinking and Forth's organizational strategy.

Forth's interactiveness allows, even demands, play. One can play with Forth words. Our attention spans are limited, and although the human mind is no longer stretched by hours or days awaiting a FORTRAN mainframe edit, compile, link and test, even 30 seconds is too long. Forth's immediate responsiveness matches our limited attention span.

Given the ability to organize our thoughts in a computer similar to how we think, we can interact with, and edit, our programs as if they were thoughts.

The French mathematician, Descartes, wrote a long letter to a friend, at the end of which he apologized for having not had the time to write a shorter letter. We rarely have time to write a proper program, let alone the luxury of rewriting one, save to fix bugs. Yet, what is a program in Forth? A collection of words: a collection we constantly name, rename and rewrite. We give old words new meaning.

Forth encourages the discipline of rewriting, and gives us the time to do so.

Dr. C.H. Ting pointed out that talk show host Johnny Carson often had a pencil in his hand. A pencil with an eraser on either end. As Ting noted, "the important part of a pencil is the eraser".[3] What you take away is more important than what you leave. Similarly, Dr. Alan Furman stated the aphorism[4]:

To gain knowledge, add something every day. To gain wisdom, remove

something every day.

With each rewriting our understanding of our words grows before us as we hone them further, like tools. With each honing the tool becomes more precise, sharper, and the code often smaller. Our knowledge of our application gained by working with that application becomes wisdom.

Dr. Kent Brothers, one of the authors of VP-Planner, a successful, until legally challenged, clone of Lotus 1-2-3, described at the 1990 Rochester Forth Conference[5] how they developed VP-Planner. He pointed out that each time Lotus came out with a new release, so would they. With each release Lotus became larger. With each release the code for VP-Planner got *smaller*.

The parameter stack is the primary method of parameter passing in Forth. Dr. Glen Haydon, author of *All About Forth*, observed that the use of the stack for unnamed parameters is akin to using pronouns in English. An item's position on the stack is but a place holder: its meaning is determined by the context of the word, much as the referent for a pronoun is found in the context of a sentence. Similarly, multiple pronouns and their indirection are confusing in a sentence, just as stack depths greater than three are unwieldy in Forth.

Few speak of reading computer programs. We speak of writing, debugging, documenting or maintaining code: but rarely reading. We don't think of computer code as for people: its for computers: but computers don't think, people think. Ultimately code is for people.

I have read a great deal of Forth code during the past nineteen years, some of it mine, but the majority written by others. I have been often struck by the programmer's signature: not the one left by his or her initials, but by the way in which their thoughts are expressed: particularly in Forth. Some write code which is very monolithic. It consists of a few, large definitions which do everything. Others write code made up of small well-factored pieces.

Charles Moore. the first discoverer of Forth[6], writes spare, well-honed code. His definitions are rarely more than one line long. His programs are rarely larger than a few pages[7]. Moore wrote a BASIC compiler[8] in 1981, which was elaborated on by Michael Perry.[9] Although it imposes a few restrictions, such as separating keywords with spaces and requiring integer arithmetic,

the BASIC compiler source is less than 100 lines long and compiles to 1550 bytes.

This style is reminiscent of Haiku poetry, where the constraints of form are as artificial as Haiku itself, but as real as the real time constraints the program must live within.

Poetry isn't easy to read. It requires work. Good poetry has subtlety, depth and shades of meaning. Good Forth code has subtlety and depth, although plays on words may be lost on the *in silico* interpreter.

When I read Forth code I look *at* structure *for* structure. The first is form, the second content. Form shows as tight or rambling code, lengths of words, and naming of expressions. Content appears as what the code does, and how it does it. In Forth, Form and Content are curiously intertwined. Is a compiler directive or a defining word strictly either? They are an example of Content directing Form directing Content.

As with all good writing, good Forth code shows clarity of thought through lucidity of expression.

Einstein once observed, "Things should be made as simple as possible, but no simpler", which is to say just complex enough. Several metrics for measuring software complexity have been developed, including Tom McCabe's cyclomatic metric index, known as McCabe's Metric, which

Palæography of Extensible Languages (Part 2)

WHY DID EXTENSIBLE LANGUAGES BECOME EXTINCT?

The development of extensible languages was motivated by the desire of their creators to avoid the large and cumbersome programming languages such as PL/I which tried to provide in one package all of the features of ALGOL, FORTRAN, and COBOL. The proponents of extensible languages pointed out that generality easily slips into inefficiency. To overcome this tendency, they advocated that expensive features should be made available only to those that need them.

"When I use a word," Humpty Dumpty said, in a rather scornful tone, "it means just what I chose it to mean – neither more nor less."

Lewis Carrol (1896)
Through the Looking Glass and What Alice Found There.

It is good to remember that a programming-language designer cannot foresee all of the potential users' needs.

Why did extensible languages die out?

There were many reasons for this, the most important of which are:

- Language extension via a theoretically sound metalanguage is a non-trivial operation;
- Portability could not be achieved because of the lack of a common platform;
- Well-designed programming languages, such as Pascal, provided good data-definition facilities;
- Machine-oriented languages[1] for systems programming provided access to machine features.

SUCCESSES OF EXTENSIBLE LANGUAGES

Although extensible languages died out as a group, their significant influence on subsequent programming-language design cannot be overlooked. They provided the programming-language community with a better understanding of

- macro processor mechanisms
- machine dependencies
- problems associated with portability.
- abstract data structures.

FAILURES OF EXTENSIBLE LANGUAGES

Attempts to create a widely usable extensible language ended in failure for a number of both experimental and theoretical reasons. The three most important of these are:

- Inability to provide user-level extensibility;
- Inability to describe control-structure extensibility;
- Total absence of compiler and prog-ram exchange.

Programming-language theory was still in its early stages of development in the late sixties and this led to the first two failures. At the same time, computing was totally dependent on the main frame and the programming language community was divided into a number of camps that could not exchange programs or systems. Hence, the signal lack of portability.

D: Tell us a little more about the WISC/16.

G: *The WISC/16 is about as simple as it can be and still have the necessary functions available. Everything is connected to a single 16-bit bus. The source and destination codes are chosen before enabling the particular functions. A 32-bit microcode is stored in RAM. Two stacks of 512 words each are included. The machine is 16-bit word addressed. Software utilities allow a byte swap within a 16-bit word. We have found that wasting one byte for ASCII characters is a small price to pay for the added total space available. The clock speed is derived from the host. The limiting speed is the ALU chip, the 74181. Everything is common TTL. The total chip count is 84 dips of the 7400 series and no MSI. The WISC/16 provides an ideal tool to learn how to connect your processor and application with Forth software as the glue. Half or more of the WISC/16 sales are overseas.*

D: What else do you have to offer your customers?

G: *MVP is one of the few places where a knowledgeable Forth user answers the phone. I am the author of the latest edition of All About Forth, which includes a functional definition for 7 public domain implementations and the actual source code for four implementations. The appendices in All About Forth include the Installation Manual, 79-Standard and 83-Standard documents. The combination makes the book an ideal general reference for all Forth users.*

Soon after the ANS Forth Standard is adopted, common functions will be added to All About Forth along with an implementation. It will be a 32-bit model using DPMI (DOS Protected Mode Interface). This implementation has been used to address as much as a 28 megabytes of linear address space on a standard Intel based system. The 64K and segment considerations are no longer a problem. There is minimal perceptible time penalty for this implementation. With the DPMI on the Intel machine one does not have to go to a Motorola 68K or other processor to get a 32-bit linear address space.

I am also aware of many fine proprietary Forth implementations and can put many of them into perspective for use with your applications We welcome inquires. We also offer technical typesetting and other help in preparing manuscripts, articles and books.

Glen can be reached: 1-(415) 747-0760 voice (if he can get it in 4 rings) or voice mail, fax, and BBS.

measures control structures. Dr. Tom Hand, while a professor at Florida Institute of Technology, took John Cassidy's assembler written in Forth for the Intel 8080[10] microprocessor and applied McCabe's metric. Hand noted that an 8080 assembler written in Pascal or C typically has a value far larger than 50, whereas:

Cassady's assembler has a metric of 0: there were no control structures.

Hand found if he applied other conventional measures of complexity, such as the number of variables, Cassady's assembler still had a metric of 0.

An assembler written in Forth has no complexity!

Forth allows us to write programs with no complexity: yet they still get the job done.

When I was teaching at the University of Rochester I told my students the best code was no code. If there was no code, there was nothing to write, nothing to debug, nothing to document, nothing to maintain. Though they complained, I maintained that if they solved the problem before them without code they would get an A; but for the meantime they could approach it asymptotically.

How is this possible in Forth? The key is proper problem factoring and using Forth's inherent properties: the text or outer interpreter for parsing and organizing items in a dictionary; the stack for parameter passing; and, most importantly, defining and compiling words to create structures which make structures.

The best code, is no code!

—L. P. G. Forsley

[1] This article is based upon a presentation by Lawrence P. G. Forsley, "Rhyme, Reason and the Tao of Forth", *Proc. of the 1992 Rochester Forth Conference on Biomedical Applications*, pp. 34-40.

[2] Miller, G. *Psychology of Communication: Seven Essays*, Basic Books, New York: 1967.

[3] C.H. Ting in the question period following the talk on "Rhyme, Reason and the Tao of Forth".

[4] Alan Furman in the same question period.

[5] K. M. Brothers, "The Forth System Behind VP-Planner: Designing for Efficiency in the Face of Complexity", *Proc. of the 1990 Rochester Forth Conference on Embedded Systems*, pp 5-12.

[6] Moore, C., "FORTH: A New Way to Program a Minicomputer" *Journal of Astronomy and Astrophysics*, Supplement 15:1974. pp. 497-511.

[7] Personal conversation with Charles Moore on December 2, 1993. He noted that one exception was a was a large database package he wrote while at FORTH, Inc.

[8] Moore, C., "Programming a BASIC Compiler in FORTH", *1981 FORML PROC, Vol 2*, Forth Interest Group, San Jose, CA.: 1981. pp 513-519.

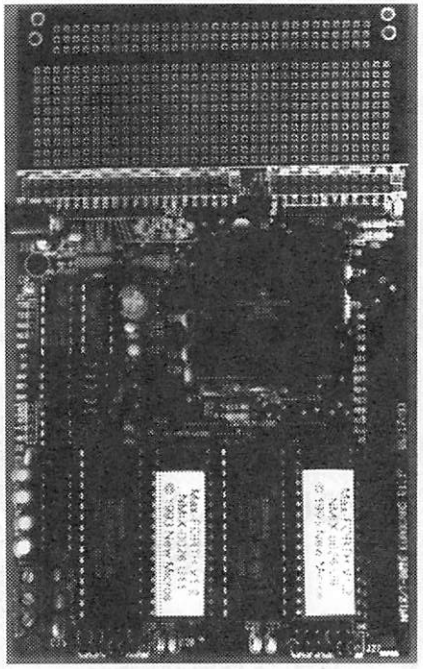
[9] Perry, M., "Charles Moore's BASIC Compiler Revisited", *Forth Dimensions*, Vol III No. 6, Forth Interest Group, San Jose, CA.: 1982. pp 175-178.

[10] Hand, T, "Software Metrics for Forth", *Proc. of the 1988 Rochester Forth Conference on Programming Environments*. pp. 67-68.

VENDOR'S COLUMN

New Micros, Inc.

New Micros has released its first revision 68HC16 board, the NMIX-0026. The NMIX-0026 is a highly featured CPU board in the NMI Eurocard board series. The NMIX-0026 has hardware feature of the 68000 single-chip computers while maintaining some software compatibility with earlier 68HC11 designs. The 68HC16 CPU is (largely) source compatible with the 'HC11. The 'HC16 is a 16-bit CPU while the 'HC11 is an 8-bit design.



Software throughput is enhanced, not only by the wider data-path, but also by the higher clock speeds, up to 16.78 MHz operation from 32 KHz osc. Address and data space are also larger in the 'HC16. Each can be as large as 1 Megabyte.

The 'HC16 chip also provides Programmable Chip Selects, which were not available on the 'HC11 (at least the original 68HC11A8 and E9 variants). The Watchdog Timer and Clock Monitor features have been supplemented with a Bus Monitor as well.

Like the 'HC11, the 'HC16 has 8 A/D channels. However, the 'HC16 offers 8 or 10 bit accuracy. The 'HC16 has the same compliment of serial channels as the 'HC11: one Asynchronous Serial channel and one Synchronous Serial Channel. Twice as many free running 16-bit timers are in the newer chip. The Input Captures, Output Compare and Pulse Accumulators are provided in the same quantity on both chips. Two Pulse Width Modulated outputs are provided on the 'HC16. These make possible a "low cost" D/A-like output. Many applications with slow response times will see this signal as an analog level. A simple filter can be used to "round off the edges" for systems with faster responses. A host of parallel lines are also available on the new part. Since the 'HC16 is packaged in a 132-pin PQFP, many more connection points are available which could not be accommodated on the original 'HC11 52-pin PLCC.

The NMIX-0026 has a small 1"x3.5" prototype area on the end of the 100x160mm Eurocard format. Connectors and interface circuitry can be added there. This is about half the prototyping area which was available on the NMIX-0021, 'HC11 based boards. The larger processor socket and additional memory requirements required enlarging the active component area of the board.

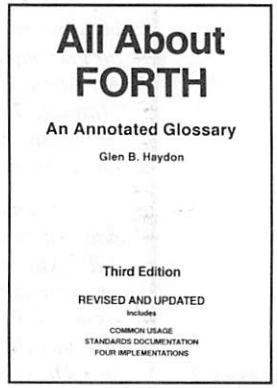
The NMIX-0026 still maintains a good deal of compatibility with the New Micros NMIT and NMIS series computers. The 32-pin JEDEC memory sockets have the same pinouts on all versions, and the jumpers for the sockets are the same as found on other boards with 32-pin sockets.

VENDOR'S COLUMN

Interview with Dr. Glen Haydon

Mountain View Press

I caught Glen Haydon, owner of Mountain View Press on a late October morning. I interviewed him at his Shangri-La West, overlooking the Pacific ocean while he was resting in his hot tub. Over the years Glen has built a glass and redwood home at the end of a long dirt road in the hills behind Stanford University.



Jon Ross, the first managing editor of the Journal of Forth Applications and Research, lived with Glen for several years while working on his MBA. During this time he used his considerable construction skills on Glen's home.

Glen's formal training was in physics, biology and medicine. He has experience in programming and hardware design. He is also an amateur astronomer with a 12 inch reflector telescope under a converted grain silo dome. He usually answers the phone himself and can handle inquiries in both English and German.

Mountain View Press, known simply as MVP, has been around for over 12 years. Glen took it over from Roy Martens about 3 years ago and made it a division of his company, Epsilon Lyra. WISC Technologies and Haydon Enterprises are also division of Epsilon Lyra.

D: What does MVP do?

G: *We publish and sell hardcopy documentation for Forth implementations and applications. We feature the MVP Forth implementation designed for beginners. The MVP Forth kernel has been stable since 1983. Some more recent applications have been added in the current distribution. We also handle books from other publishers and a number of other implementations: FPC, fig-Forth, eForth from Ting, Pygmy from Sargent, Yerkes (which is a public domain version of Neon for the Macintosh) and Rick vanNorman's 32-bit F32. These implementations and others are available on our bulletin board. We use the simple Procomm bulletin board software. Contact us for more information.*

D: What about hardware?

G: *We distribute the WISC Technology products. The WISC/16 is available in several forms including printed circuit boards. The WISC/32 is available with the Harris chip on a single board for evaluation. We hold patents on both designs. The WISC/16 is a good teaching tool.*

D: Who is using the WISC?

G: *Several schools are using the WISC/16 in their undergraduate and graduate engineering classes. Some have bought the bar circuit boards and populated them while other have bought the assembled and tested boards. A few people have found wire wrapping their own boards rewarding. It usually takes 20 to 30 hours to wire wrap. By adding hardware functions one at a time, each function can be tested as you progress. The basic Forth concept of factoring and testing each function is done in hardware. We use a PC as a host for the development hardware and use a little bit of glue to allow the host to access a single central bus. All functions are accessed from the central bus.*

there was a good chance the path might be too complicated for some PostScript versions. In the end I settled for moving it outside the **repeat.** I simply stroked the whole path at once. (Once for horizontal lines, once for vertical.) I saw no quick means of changing the structure which would leave such an easily understandable example intact.

The first printed page (which was not solid white) was almost solid black. At that time I had not added the **setlinewidth** command. I initially set it to .01, which was a very fine line. Later I decided .1 was more aesthetically pleasing. When the line width was handled, I focused on the problem of the page being only 2/3's drawn. I had reversed the order on the X and Y inputs to the **rlineto** and **rmoveto** commands.

Finally, I had used **.1 setgray** and had very dark lines. I changed the command to **.9 setgray** and achieved the desired results.

For someone regularly using PostScript, with an interactive environment, the whole procedure should have only taken 10 minutes. Such is the nature of programming, it is not for the faint hearted or those without the necessary tools at hand.

In the next issue, I will discuss a few better development tools operations, and where these tools can be obtained.

—R. Dumse

Since the memory map of the 'HC16 is 16-bit, the minimum configuration of sockets would require four, two for RAM and two for ROM. Four 32-pin JEDEC memory sockets are provided on the board. Any 28-32-pin JEDEC device can be used in any sockets. In fact, 1 Megabytes of Pseudo Static RAMs (PSRAM) can be installed in the two RAM sockets, each with 512 Kbytes.

The previous 8-bit boards had three memory sockets by design. This allowed one RAM for variable storage, etc.; one ROM for language monitor or operating system; and one open socket intended for the target code. This could be RAM during development and then EPROMed for production. The three sockets fit neatly on the chosen Eurocard format.

This is not possible with the 'HC16 and its 16-bit memory access. The power supply section of our previous designs was deleted to make room for four sockets. This allows two 8-bit RAMs and two 8-bit ROMs to be present in memory at one time. The increased size of the usable memories, however, may relieve some of this lack of sockets, as a large, battery-backed RAM can simulate both program space and data space. The programmable chip selects make moving memory devices possible without actually moving them from their sockets.

A new scheme was needed for the bus expansion cards. The old 34-pin Vertical Stacking Connectors (VSC-34) so familiar on the 8-bit cards was expanded to a 60-pin JEDSTACK Vertical Stacking Connector (VSC-60). It was done in such a way 8-bit peripherals will still work in the lower 34 pins of the VSC-60, yet new memory cards can be added which will take advantage of the entire 16-bit data-bus and address bus.

Previously, the best sellers in the New Micros series of single board computers have been boards based on the 8-bit F68HC11. As mentioned, the 68HC16 is a faster, 16-bit version of the popular 68HC11. The transition from 8-bit to 16-bit was not a smooth one. This was not an easy board for NMI to make. They had planned to come out with it much earlier, but the greatest part of the delay turned out to be something beyond their control.

Randy Dumse, president of New Micros, Inc. found:

The prototype house who made our fast turn protoboards made several defective PCB's in a row. We spent valuable resources trying to figure out why the design didn't work. The software never seemed "to boot". However, it turned out the PCB was full of shorts.

Shorts on the data-bus caused the chip to go into odd operating modes on power-up. It was a ticklish problem to isolate, since the board is seldom questioned first. Finally, the software development continued without the prototype hardware using a Motorola EVM board. A close implementation of the F68HC11 MAX-FORTH was ported to the 'HC16.

The lead spacing on the 132-pin PQFP is .05" and is made for surface mount attachment. NMI found a socket made by AMP which allows several-time replacement of the IC despite the delicate leads. Successful insertion and removal should be possible about 20 times. Although this is far less than other devices, it is better than having a Single Board Computer in which the CPU could never be removed.

The biggest problem with the design was accommodating the chip selects. On-chip programmable chip-selects should make the ultimate in flexibility. For the most part this is true. Unfortunately, the architecture, had one drawback: mixed 8-bit and 16-bit selects. There are no provisions

for priority with the internal chip selects. An 8-bit access occurring at the same memory location as a 16-bit access, causes a full 16 bit transfer. Since the board can carry the full addressable space of 2M RAM, if the chip selects are set for full access to memory, there is no space in the memory map not covered by a 16-bit access chip select. Therefore, addressing 8-bit peripherals is also difficult.

As Randy points out:

The design assumption made by NMI was that anyone willing to pay the cost differential to step up from the 'HC11 to the 'HC16 needed additional processing speed.

Processing speed was preserved by adopting a 16-bit bus despite the problem of overwriting "paired" bytes on 8-bit operations. This was fixed in software for NMI's 'HC16 FORTH and will be in hardware on a second generation of the board which will use 8-bit chip-selects. All memory will be accessed as 8-bit, which will remove boundary access and single byte write problems. The 16-bit bus speed will be maintained for on-boundary accesses, by having two 8-bit memory accesses in parallel. Although this will use two more of the processor chip select pins otherwise available for the user, it preserves performance.

In addition, the programmable chip selects don't easily allow full access to the 2M space. Since the 'HC16 does not have 24 address lines, but its 68000 oriented hardware does, the upper address lines were "faked" by copying the top address line the 'HC16 does have. Unfortunately, this means the largest possible chip select is half-a-Meg. FORTH tests which half of 1 Meg memory space is being accessed and automatically adjusts the chip select.

Randy notes:

There is still more to be learned by chip manufacturers about combining hardware with software. These chip selects prove that. While well thought out, not every possible use is obvious.

Versions of the board are also planned, which will accommodate the new 68HC16 parts, such as the MC68HC16Y1 with the Time Processing Unit (TPU). The MC68HC16Y1 comes in a 160-pin PQFP. The TPU is a small "co-processor" which manages additional timers. The RAM in the MC68HC16Y1 can be used as emulation control stores to develop new "micro-code" for the TPU. Developing a FORTH interface to maximize this unique feature of this part is a challenge, yet to be realized. Still, the development of this new line of processors continues at NMI, having far reaching consequences for future 16 and 32-bit boards from the company.

Next issue we will talk to NMI about their newly proposed Easy-A Multidrop Protocol standard.

For More Information:

New Micros, Inc.
1601 Chalk Hill Road
Dallas, TX 75212-3804
(214) 339-2204

output. I wrote the text in an editor and downloaded it to the printer. I was "blind" to any error messages. I tried to change from the parallel port to the serial port, so I could see the error messages. The last time I did that was on an NEC890 printer, but now I use an NEC95 printer. The setup was different, and I decided it would take half an hour to figure out the "secret" difference which alluded me on the first try. In the end I settled for the tried and true, trial and error method of debugging. This method will always produce results - slowly - but faithfully.

My first indication of a problem was when the download ended with a "WAITING" message on the printer front panel, followed by a several minute delayed return to "READY PS". There was no output. A rereading of all the commands used did not reveal any anomalies. By commenting out essentially all the program (using **%**) except the **showpage** at the end, I was able to spit out a blank page. Good. That proved the transfer method through the parallel port was working. Next, individual lines were uncommented and clean pages were occasionally received. This produced about 6 pages of blank paper. To my surprise, the code I most suspected was giving no problem. The simplest code was the problem. The procedure being passed to the **repeat** command would not interpret without error. Originally, I had a **stroke** command in the **repeat** loop between the **rlineto** and the **rmoveto.** Only one path was to be generated at a time. It turns out, since **stroke** uses up the path, it essentially eradicates the current point, so the **rmoveto** did not know where the starting point was. This error prevented output.

It is not a good idea to let the paths become too complicated in PostScript. I used the **stroke** in the procedure to be sure the path was only one segment long. By removing it, outside the **repeat** phrase, I felt

after the scaling. Making the line width 1/10" of the quadrille cell size was an arbitrary choice. For a sharper line on the page, a smaller line width can be chosen, according to the minimum resolution of the printer being used.

With these preliminaries out of the way, the actual lines can be drawn. PostScript has some of the feel of the old turtle graphics. To establish a first position and origin, **0 0 moveto** can be used.

From this known position, lines can be traced out in relative position. The **moveto** command uses absolute coordinates. There are also a whole set of relative commands. The **rlineto** command uses the current position and the supplied x and y values to calculate a new position, relative to the first. In turtle graphics terms, the pen is down during this move. There is a noticeable relationship between the turtle graphics approach and the operation of PostScript. The **rlineto** describes a path, but does not actually draw any "dark bits" in the buffer being readied for the paper. This will be done by the **stroke** command after the path is described. The PostScript "turtle" only draws potential for color, not the actual color itself.

After the **rlineto** command, a return to the next line's position is used. It is important not to create a path with this move, which would be darkened with the stroke. The command to do this is **rmoveto.** It is the relative counterpart to the absolute **moveto** command. In this case, the **rmoveto** both repositions the "turtle" back to the left of the screen and moves one-unit-step up the page. In turtle graphics terms, the pen is up in this move.

The **repeat** command takes two parameters, a count and a procedure. The 45 horizontal lines on the page can be added to the path using **repeat.**

Finally, the stroke turns the path into drawn lines in the memory image which will be the dark traces on the paper. As mentioned, this is a different step from the turtle graphics concepts. The path created with **rlineto** (or a host of other drawing commands available in PostScript) need either to be stroked or filled before the bits in the output buffer are set dark (according to the setgray level). Stroking traces the path with a line whose width is controlled by **setlinewidth.** In this case, the paths are meant to be stroked with the **stroke** command.

The same procedure is used to set the vertical rules. Finally, after the drawing is done, the original graphic state is restored, with the **grestore** command, and the output buffer transferred to the paper in the printing process with the **showpage** command.

The complete program follows:

```
gsave
.9 setgray
18 18 scale
.1 setlinewidth
0 0 moveto
45 { 36 0 rlineto -36 1 rmoveto } repeat
stroke
0 0 moveto
37 { 0 44 rlineto 1 -44 rmoveto } repeat
stroke
grestore
showpage
```

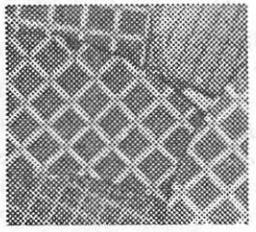
Well, was it worth the effort? This program took about an hour to write and an hour and a half to debug. It's really questionable if it wouldn't have been better to get the straightedge out and do the work by hand. Why did it take two and a half hours? First, since I don't program in PostScript very often, I had to look up almost every command. For instance, I couldn't remember which end of the gray scale **.1 setgrey** produced. I thought it would be very light and it was actually very dark.

The second reason is that I didn't have an on screen viewer for the PostScript

33+ MHz Forth Processor

Native Forth instruction set processors provide a high level language with efficient instruction execution. For example, complex instruction set computers, *CISC*, like the Motorola 68000, take tens of clock cycles/ instruction. Even reduced instruction set computers, *RISC*, like the SUN SPARC, average one or two clock cycles/instruction. By comparison, multiple Forth instructions can be executed per clock cycle, some reporting an *average of 1.5 instructions/clock cycle!*

The Johns Hopkins FRISC3 was licensed by Silicon Composers and is commercially available as the SC32. FRISC3 was produced in 1987, and first silicon was available in 1988. It is a 32 bit Forth processor with single cycle subroutine calls and the ability to embed the subroutine return in some operations giving a free subroutine return. It has two on chip stack buffers, each of which is 16 elements deep with overflow into memory. Considerable simulation of Forth Programs show that a register bank of 16 is optimal. Dynamic studies of stack nesting depth by Koopman[1] and Hayes[2] indicate that the execution, or return stack, rarely exceeds a depth of 16.



Marty Fraeman and John Hayes of the John Hopkins Applied Physics Laboratory have been using the Chipcrafter silicon compiler (now known as EPOCH) from Cascade Design Automation of Seattle, Washington to develop a new version of the SC32, referred to here as the FRISC4. It will be like the SC32, with similar instruction set but a higher level of integration. Silicon Composers will fabricate the part for their customers and the Applied Physics Laboratory.

The FRISC 4 instruction set includes a **DOES>** instruction, which is a data subroutine call which pushes the return address on the parameter stack in one cycle. There are four branch instructions including **DOES>** and subroutine call. This is a byte addressed machine, with load and store byte instructions.

There is better math support including a barrel shifter and a leading zero counter for floating point normalization and image compression. For example, one lossy algorithm does log compression and saves the exponent. There is better hardware for multiply and divide. Since a multiply can be done 2 bits/clock, a 32 x 32 multiply with a 64 bit product will take 16 clock cycles. The divide step is 1 bit/clock, or 32 clock cycles for a full register divide of 64 bits by 32.

In modifying the original architecture, John Hayes noted:

To first order, the 16 deep stacks capture most data accesses. The stack buffers got us down to 1 cycle/instruction. But, you now have a memory access problem, where you access memory every processor cycle. Our Forth program measurements show 10% of the memory accesses are data and no programs had greater than 20% data accesses. Writes are rarer (a few percent) This is old information but we're exploiting it now.

Forth programs run in simulation showed combined instruction/data caches were better than instruction only caches. Currently there is an instruction data cache of 4 kbytes. For all of the benchmarks there was a hit rate of over 90%, although we prefer to use an effective cycle time measurement which takes into account the miss penalty and hit ratio.

Cache could be a big win in space because there are very few space-rated

memory parts. For example, United Technologies Microelectronics Corporation (UTMC) sells 55 nsec 32K x 8, radiation hard memories. However, the FRISC4 on-chip cache is at least twice as fast as the external memory part, and may use less power.

Processor cost is partly a function of die size. The larger the die size, the lower the yield, and the higher the cost. The FRISC4 die will be smaller than the SC32 die which is about 10 mm on a side. There may be a cache-less version for both radiation hard and commercial versions of the processor.

John also wanted an interface which required less "glue" logic and could take advantage of many memory families. A real simple system could be built without additional chips. This is especially important in space, and other applications, which put a premium on power consumption. He designed a programmable bus interface where:

I divided the 4 gigabyte address space into 16 regions. Each region has a programmable number of wait states, with a 6 bit register per region. There is a very shallow, write buffer 32 bits wide and 1 word deep.

The new processor has the ability to boot from an external byte wide interface (the SC32 needs a 32 bit wide EPROM). There is a programmable on-chip interrupt controller, responding to edge, or level inputs, with at least 6 external and 2 internal interrupts. It includes an on chip UART with a programmable baud rate, 1 start, 8 data and 1 stop bit. This additional functionality requires extra pins. Whereas the SC32 is available in an 84 pin package, the FRISC4 will require at least 144 pins in a square package

The processor core was finished in September 1992 and was executing programs on a simulator. The interrupt controller, byte wide boot, bus interface, 4 kbyte direct map cache and a UART have been designed and implemented. Error correcting code (ECC) hardware has been added on chip which can detect double bit and detect and correct single bit errors. This feature is enabled or disabled in software. When used, 7 additional memory bits are required. Marty Fraeman points out:

There will be minimal impact on performance since we're usually running out of cache.

The ECC was inspired by radiation problems encountered with the Freja satellite. (see *Forth in Space, Volume 1, No. 1*). John notes that there is a potential conflict in:

simultaneously designing a fast commercial processor while building a low power space processor. However, both like rich features, except ECC, which is less important on the ground.

Also under consideration is a dual counter timer and a parallel port.

It is expected that FRISC4 will see silicon by the middle of 1994. The processor will be implemented in .7 micron silicon. Simulation shows that a 33 MHz clock rate is possible. As John notes:

somewhere on the curve we ought to get 40 MHz parts.

For more information contact:

Dr. George Nicol
Silicon Composers
655 W. Evelyn Ave#7
Mtn View, CA 94041
(415)-961-8778

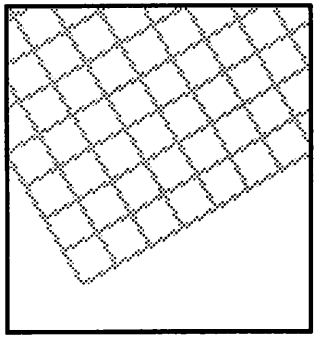
[1] Koopman, P., *Stack Computers: the new wave*, John Wiley and Sons: 1989. pp 139-144.

[2] Hayes, J. and Lee, S., "The Architecture of the SC32 Forth Engine", *Journal of Forth Application and Research*, Vol 5 No 4: 1989. p 504.

PostScript vs. Forth

Imagine you are working late one night and go to your drawer for some quadrille paper and there is none. There's none in the supply cabinet, either. You are faced with the dilemma, give up on the deadline, or, get out the straightedge and eversharp and carefully hand make your own ruled paper. Now the former is not career enhancing, and the later requires several hours of tedium. But wait! There's that PostScript printer on the network. Why not program it to make what you need. Will that be faster? Well, it's worth a shot.

Last time we compared PostScript to Forth and found there were many similarities, both superficial and fundamental in the two stack-oriented, extensible languages. Both PostScript and Forth defining words were shown in action. In this issue a real code example will be given. In a column this short, it is not practical to teach the whole language, so words will be used and explained as necessary, but not in the sequential and thoroughly complete manner that might appear in a text book.



Referring to the PostScript Reference manual, you discover the default unit of measure in PostScript is the "point". A "point"? What's that? Well, in typography it's the unit of choice for describing text sized things. Oh, like point sizes in fonts? Yes, that's it exactly. How many points to the inch? A point is (close enough for almost all purposes) 1/72 of an inch.

If the quadrille is to be 1/4" spacings, there would be 18 points between lines. But wait, there's another way. Why not make 1/4" the new unit-of-measure. The PostScript operator **scale** can set the coordinate system to any desired size. So the command **18 18 scale** would cause a unity graphic step to be represented as a 1/4" step. Hence, an 8.5 x 11" sheet becomes a matrix of integer tracks ranging from 0,0 in the lower left to 36,44 in the upper right corner.

Before using the scale operator, the graphics condition of the machine should be saved in its pristine condition. This is to ensure the following pages come out in regular size, without concern for the context of the previous use of the printer. The command to save the graphic state is **gsave.**

The line to be used to draw the rules should not be at 100% intensity. The **setgray** command can select the appropriate intensity so the lines will not overpower the writing yet to come (the lines you want to draw on the paper). Intensity of the drawing can range from 0 to 1. The lower the value the blacker the drawing. The default setting is 0, for black. In the example it is set to a light level of .9 by the command, **.9 setgray.**

The scaling is accomplished next by the previously described **18 18 scale.**

Left on its own, the width of the line drawn is determined by the default line width setting of 1. Before the scale, this was 1 point. After the scale, it is 1/4". This setting of the line width is not acceptable. The entire paper would be blackened. There would be no white space between the thick, 1/4" lines placed every 1/4". The **setlinewidth** command is used to narrow down the line