VOLUME X, NUMBER 4 NOVEMBER/DECEMBER 1988

FORTH DIMENSIONS

SIMPLE SCREEN DIRECTORY

STANDALONE APPLICATIONS IN F83

MENU-DRIVE THE 8250

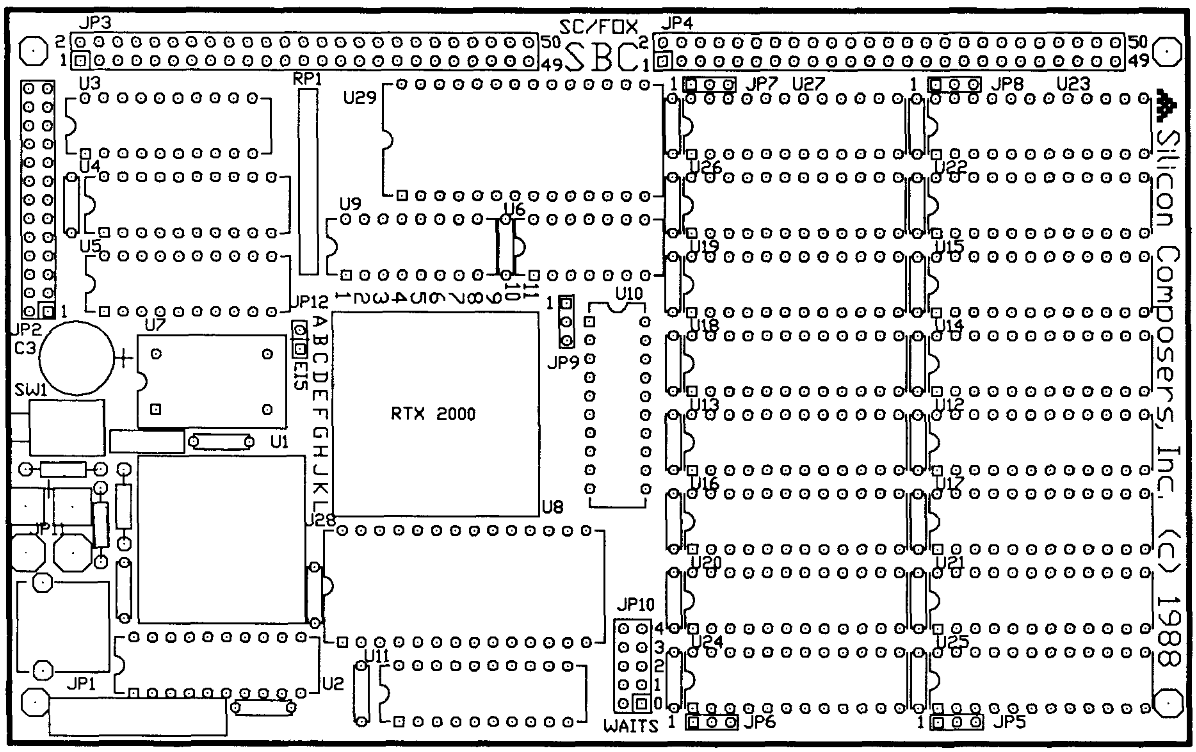
MORE ABOUT STACKS...

Introductory Offer: $100 SC/FOX SBC Discount with Prepaid Orders through 1/1/89

SILICON COMPOSERS

Introduces the

SC/FOX™ Single Board Computer



(actual size)

SC/FOX (Forth Optimized eXpress^tm) SBC:
- 8 and 10 MHz RTX 2000 options
- 32K to 512K bytes 0-wait state SRAM
- 64K bytes of shadow-EPROM space
- Application boot loader in EPROM
- FCompiler^tm Forth software included
- Code converter for EPROM programs
- RS232 serial port with handshaking
- Centronic parallel-printer port
- Single +5 volt board operation
- Two 50-pin application headers
- Eurocard size: 100mm by 160mm
- SC/FOX Coprocessor compatibility
- Retail from $1,195 with software

Harris RTX 2000^tm Forth CPU Features:
- 1-cycle 16 x 16 = 32-bit multiply
- 1-cycle 14-prioritized interrupts
- one non-maskable interrupt
- two 256-word stack memories
- three 16-bit timer/counters
- 8-channel 16-bit I/O bus
- CMOS in 85-pin pin-grid array

Optional SC/FOX SBC Products:
- SC/Forth^tm Language in EPROM
- SC/Float^tm Floating Point Library
- SC/SBC/PROTO^tm Prototype Board
- SC/FOX/SCSI^tm I/O Daughter Board

Ideal for embedded real-time control, high-speed data acquisition and reduction, image or signal processing, or computation intense applications. For additional information, please contact us at:

Silicon Composers, Inc., 210 California Ave., Suite K, Palo Alto, CA 94306 (415) 322-8763

FORTH
DIMENSIONS

EDITORIAL

I was able to attend part of this year's Hackers Conference, possibly the last place where certain core issues about microcomputing still receive general discussion. Do you remember earlier times, when what we now call our business was known as the microcomputing revolution? Do you remember why we called it that? Do you remember why we worked so diligently to promote computer literacy, public access with personal privacy, and interactive mass media? Believe me, it was for better purposes than touting a new class of business machines. If you take the time to explore the influence of man's philosophy on technology, and vice versa, you'll see that our technology bears the fingerprints of its creators, if not all their names.

Hackers 4.0 proved that some of the old sparks are still burning. Arriving with invitation firmly in hand, I encountered people I hadn't seen since the old days at People's Computer Company. I had published or corresponded with some of them, years ago, as the editor of what was then *Dr. Dobb's Journal*. Some of the attendees had been party to key developments in the evolution of microcomputers; others were hackers by temperament and social vision, but not of machines. The multi-faceted personality of the group is partly explained by Bob Bickford's post-Hackers 2.0 definition of a hacker: "Any person who derives joy from discovering ways to circumvent limitations." With this in mind, it is perhaps unsurprising that a half dozen or so of the two-hundred-plus attendees are well known as Forth language pundits. Their participation reinforced my suspicion that Forth is about as close as you can get to a computer hacker's natural medium.

It was refreshing to spend time with this group of individuals, whose interests in hardware specifications and data representation were balanced — and in some cases fueled — by human concerns like ethics, the environment, and personal integrity. Some of these people, but especially the values they represent, helped to shape the machines we use today. Quietly hacking away in their garages or offices, some of them are still helping to shape the machines of the future.

* * *

As for the present, it seems our last issue got many readers' attention with its focus on stacks. We had a hunch it was time to air some fresh ideas about this fundamental feature of Forth. In fact, we got such an interesting response that we are following up on it without delay. You will find in this issue, along with the remainder of Ron Braithwaite's string-stack implementation, an analysis of the speed savings created by implementing *just the top* of a data stack in registers, and two relevant letters to the editor: the first expands on Yngve's idea for an extra stack, and the second shows how to use Johansen's shadow stack while compiling.

* * *

The most recent addition to the schedule of Forth-specific events is SIGForth '89, to be held in February at the Four Seasons Hotel in Austin, Texas. This is a function of a fledgling ACM SIG, and the call for papers stresses real-time software engineering. New Year's Day is the deadline for abstracts, so write to them soon for information if you want to attend or to speak (see advertisement).

This issue is scheduled to hit the streets during the Forth Interest Group's Real-Time Programming convention in Los Angeles. Our next issue will bring full coverage, including the winner of the "world's fastest programmer" contest. The next week finds the yearly FORML meeting on the Monterey peninsula (topic of emphasis: artificial intelligence). You'll be hearing more about that meeting of the minds, too.

—*Marlin Ouverson*
Editor

Forth Dimensions
Published by the
Forth Interest Group
Volume X, Number 4
November/December 1988
Editor
Marlin Ouverson
Advertising Manager
Kent Safford
Design and Production
Berglund Graphics

Forth Dimensions welcomes editorial material, letters to the editor, and comments from its readers. No responsibility is assumed for accuracy of submissions.

Subscription to *Forth Dimensions* is included with membership in the Forth Interest Group at $30 per year ($42 overseas air). For membership, change of address, and to submit items for publication, the address is: Forth Interest Group, P.O. Box 8231, San Jose, California 95155. Administrative offices and advertising sales: 408-277-0668.

Copyright © 1988 by Forth Interest Group, Inc. The material contained in this periodical (but not the code) is copyrighted by the individual authors of the articles and by Forth Interest Group, Inc., respectively. Any reproduction or use of this periodical as it is compiled or the articles, except reproductions for non-commercial purposes, without the written permission of Forth Interest Group, Inc. is a violation of the Copyright Laws. Any code bearing a copyright notice, however, can be used only with permission of the copyright holder.

About the Forth Interest Group

The Forth Interest Group is the association of programmers, managers, and engineers who create practical, Forth-based solutions to real-world needs. Many research hardware and software designs that will advance the general state of the art. FIG provides a climate of intellectual exchange and benefits intended to assist each of its members. Publications, conferences, seminars, telecommunications, and area chapter meetings are among its activities.

"*Forth Dimensions* (ISSN 0884-0822) is published bimonthly for $24/36 per year by the Forth Interest Group, 1330 S. Bascom Ave., Suite D, San Jose, CA 95128. Second-class postage paid at San Jose, CA. POSTMASTER: Send address changes to *Forth Dimensions*, P.O. Box 8231, San Jose, CA 95155."

LETTERS

Superstacks
Dear Mr. Ouverson:

There seems to be some interest in stack extensions, judging by the last issue of *Forth Dimensions*. Victor Yngve, who has given us synonyms and macros, has now given us a simple way to create and manipulate stacks (*FD* X/3). Yngve calls it a confection, so the superstacks described here are just a light dusting of powdered sugar on top of a confection. The aim is to extend the simple extra stack idea to a set of stacks. The method we shall use is to generalize the idea behind XSTACK, which is the fixed address of the stack pointer for a simple stack. We will make the address of the cell containing the stack pointer a variable, and will use it to switch among the stacks. We will redefine XSTACK so that it will contain the address of the stack pointer of the *n*th stack. The contents of the cell to which the stack pointer points will change with manipulation of the stack (see Figure One).

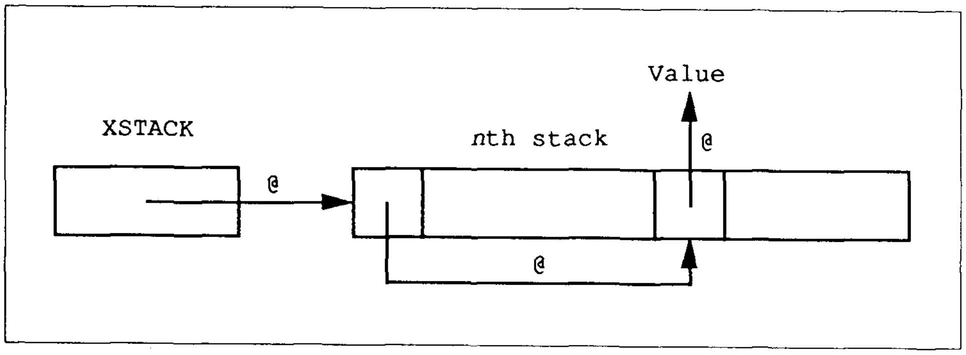


Figure One. Garian's superstack pointers.

The following word creates a data structure consisting of a set of identical stacks. It replaces the definition of an extra stack, which had the effect of making XSTACK a constant.

```
: SUPER
  CREATE #STACKS 0
    DO HERE ,
    XSIZE 2* ALLOT
    LOOP ;
```

where we have first defined:

```
6 CONSTANT SIZE
6 CONSTANT #STACKS
0 VARIABLE XSTACK
0 VARIABLE STK#
```

SIZE is the maximum depth of a stack, #STACKS is the number of stacks in the superstack, XSTACK is a variable for switching stacks, and STK# contains the current stack number.

Next, we actually lay down the structure with:

```
SUPER STACK
```

We now have a 6 x 6 superstack named STACK. Switching stacks is accomplished by computing the address of the stack pointer of the *n*th item in STACK:

```
: SWITCH  ( n -- )
  DUP STK# !
  XSIZE 1+ * 2*
  STACK + XSTACK ! ;
```

And, finally, we have to be sure that XSTACK leaves the address of the current stack pointer:

```
: XSTACK
  XSTACK @ ;
```

That's it, except for some useful words like XSWAP, XDROP, and .STACKS and CLRALL, which operate on the entire superstack:

```
: XSWAP
  X> X> SWAP
  >X >X ;
: XDROP
  X> DROP ;

: .STACKS
  #STACKS 0
    DO I SWITCH
    .X LOOP ;

: CLRALL
  #STACKS 0
    DO I SWITCH
    XCLEAR LOOP ;
```

One of the advantages of this switching technique is that the original stack manipulation words work exactly the same way on all stacks, and you can use each stack independently, without having to provide an index for every stack operation. As for applications, superstacks can be used to hold temporary anonymous values, to sort out information for various purposes (e.g., windows, graphics), and to rearrange the parameter stack more easily. There are probably many more applications out there that will become evident as the limitations of having only one or two stacks is removed.

Robert Garian
2522-E S. Arlington Mill Dr.
Arlington, VA 22206

He Wants Proof

Dear Marlin,

I have been using Forth for five years, making my living with it for two. Recently, I have been in a situation that sharply pointed up some issues about my use of Forth. Both of us have heard these issues again and again, and now I will bring them up one more time: file I/O and extended-precision, or floating-point, arithmetic.

I am working in an environment where there are two different microprocessors, running different Forths and communicating. I am writing graphics software which needs to be executed, alternatively, in either or both environments on one or more of each processor type. Graphics (3-D flavored) eats up lots of resources and needs high-precision calculations to make pretty pictures, so optimizing and balancing the loads are critical.

I find myself constantly hacking away at my Forth source code, mainly trying to integrate data structures and algorithms for file I/O and arithmetic precision. Now that I am dealing with two Forths and two processor types, this has become a circus. Forth (both fig-FORTH and F83) is optimized for screen I/O and single- or double-precision integer math. I would like to challenge anyone to provide any rigorous proof that this is a necessary or sufficient limitation to Forth. If this limit cannot be rigorously defended, I think it incumbent, in this day of multi-megabyte hard drives and 32-bit processors, that Forth move to deal with these issues. Telling someone to go buy a commercial Forth package with these options does not advance by one byte the Forth community's ability to deal with these issues rigorously. I bought the packages and along came the other baggage, the worst being that the best commercial solutions are optimized for a specific processor and the source code sure-as-heck is not portable. I can't believe the answer is to write Forth in C to achieve a transportable solution and have access to I/O and math wordsets that are not someone's copyrighted property.

I would like to see FIG rejuvenate the kind of mental energies that went into the great `CASE` issues of years past. File I/O of the VDI type, and floating-point/extended-precision (64 – 80 bit) arithmetic wordsets are the types of things that, although controversial, can promote the kind of interest and insight that move a little closer to scientific puzzle-solving, and a little further away from processor-dependent code examples that drive me crazy. I would like to see prizes, like a copy of each of the books in the *Forth Dimensions* order form, a free trip to the next overseas Forth symposium, etc.

Sincerely,

Mike McCann
P.O. Box 34160
Omaha, NE 68134

Student's Forth

There are few good textbooks about Forth, *Starting Forth* being an outstanding example. For those looking for an alternative, I would like to recommend another favorite of mine, *The Student's Forth* by Glyn Emery (Blackwell, 1985). This little book seems to have gone unnoticed by the (American) Forth community. It isn't even mentioned in the latest edition of *A Bibliography of Forth References*. In only 100 pages, it covers Forth programming and implementation in a well-structured and clearly written way that makes it a good basis for teaching Forth. This book is exactly what its title suggests.

Yours,

Henning Hansen
#116, Technical Univ. of Denmark
2800 Denmark

Shadow Stacks Get Smart

Dear Marlin,

Thank you for publishing my article, "Shadow Stacks" (*FD* X/3). I have taken those ideas a little further since then.

By making `!SHADOW` state smart, you can eliminate the semi-kludgey `]S` word. Now when `INTERPRET` converts a number, it will store the high 16 bits to the shadow stack if the system is interpreting, or it will compile the high 16 bits as a literal and put that onto the parameter stack when the word is executed:

```
: TUCK_SHADOW
  SHADOW_PTR 2- ! ;
```

(Tuck on shadow stack.)

```
: <!SHADOW> ( n -- )
  STATE @
    IF COMPILE LIT ,
    COMPILE TUCK_SHADOW
    ELSE !SHADOW THEN ;
```

(State-smart `!SHADOW`.)

Compile `<!SHADOW>` into the definition of `INTERPRET` (instead of `!SHADOW` as was described in the article).

As hinted at in the article, the Forth primitive operators can be extended to handle 32-bit numbers, then both 16- and 32-bit numbers will have the same stack effects and can be mixed and handled by "size-smart" words (which will use `@SHADOW`). Some definitions to convert double numbers to "shadow numbers" are needed to set this up:

```
: 2SH->D ( s1 s2 -- d1 d2 )
  >R >R @SHADOW
  R> R> ;
```

(Convert two shadow numbers to two double numbers.)

```
: 2D->SH  ( d1 d2 -- s1 s2 )
  >R >R !SHADOW
  R> R> !SHADOW ;
```

(Convert two double numbers to two shadow numbers.)

Here are some of the redefined primitives:

```
: DUP   ( s1 -- s1 s1 )
  @SHADOW DDUP 2D->SH ;

: SWAP ( s1 s2 -- s2 s1 )
  2sh->d dswap 2d->sh ;

: +  ( s1 s2 -- s1+s2 )
  2SH->D D+ !SHADOW ;

: -  ( s1 s2 -- s1-s2 )
  2SH->D D- !SHADOW ;

: AND  ( s1 s2 -- and )
  2SH->D ROT
  AND >R AND >R !SHADOW ;
```

(Continued on page 18.)

YES, THERE IS A BETTER WAY
A FORTH THAT ACTUALLY
DELIVERS ON THE PROMISE

HS/FORTH

POWER

HS/FORTH's compilation and execution speeds are unsurpassed. Compiling at 20,000 lines per minute, it compiles faster than many systems link. For real jobs execution speed is unsurpassed as well. Even non-optimized programs run as fast as ones produced by most C compilers. Forth systems designed to fool benchmarks are slightly faster on nearly empty do loops, but bog down when the colon nesting level approaches anything useful, and have much greater memory overhead for each definition. Our optimizer gives assembler language performance even for deeply nested definitions containing complex data and control structures.

HS/FORTH provides the best architecture, so good that another major vendor "cloned" (rather poorly) many of its features. Our Forth uses **all** available memory for both programs and data with almost no execution time penalty, and very little memory overhead. None at all for programs smaller than 200kB. And you can resize segments anytime, without a system regen. With the GigaForth option, your programs transparently enter native mode and expand into 16 Meg extended memory or a gigabyte of virtual, and run almost as fast as in real mode.

Benefits beyond speed and program size include word redefinition at any time and vocabulary structures that can be changed at will, for instance from simple to hashed, or from 79 Standard to Forth 83. You can behead word names and reclaim space at any time. This includes automatic removal of a colon definition's local variables.

Colon definitions can execute inside machine code primitives, great for interrupt & exception handlers. Multi-cfa words are easily implemented. And code words become incredibly powerful, with multiple entry points not requiring jumps over word fragments. One of many reasons our system is much more compact than its immense dictionary (1600 words) would imply.

INCREDIBLE FLEXIBILITY

The Rosetta Stone Dynamic Linker opens the world of utility libraries. Link to resident routines or link & remove routines interactively. HS/FORTH preserves relocatability of loaded libraries. Link to BTRIEVE METAWINDOWS HALO HOOPS ad infinitum. Our call and data structure words provide easy linkage.

HS/FORTH runs both 79 Standard and Forth 83 programs, and has extensions covering vocabulary search order and the complete Forth 83 test suite. It loads and runs all FIG Libraries, the main difference being they load and run faster, and you can develop larger applications than with any other system. We like source code in text files, but support both file and sector mapped Forth block interfaces. Both line and block file loading can be nested to any depth and includes automatic path search.

FUNCTIONALITY

More important than how fast a system executes, is whether it can do the job at all. Can it work with your computer. Can it work with your other tools. Can it transform your data into answers. A language should be complete on the first two, and minimize the unavoidable effort required for the last.

HS/FORTH opens your computer like no other language. You can execute function calls, DOS commands, other programs interactively, from definitions, or even from files being loaded. DOS and BIOS function calls are well documented HS/FORTH words, we don't settle for giving you an INTCALL and saying "have at it". We also include both fatal and informative DOS error handlers, installed by executing FATAL or INFORM.

HS/FORTH supports character or blocked, sequential or random I/O. The character stream can be received from/sent to console, file, memory, printer or com port. We include a communications plus upload and download utility, and foreground/background music. Display output through BIOS for compatibility or memory mapped for speed.

Our formatting and parsing words are without equal. Integer, double, quad, financial, scaled, time, date, floating or exponential, all our output words have string formatting counterparts for building records. We also provide words to parse all data types with your choice of field definition. HS/FORTH parses files from any language. Other words treat files like memory, nn@H and nn!H read or write from/to a handle (file or device) as fast as possible. For advanced file support, HS/FORTH easily links to BTRIEVE, etc.

HS/FORTH supports text/graphic windows for MONO thru VGA. Graphic drawings (line rectangle ellipse) can be absolute or scaled to current window size and clipped, and work with our penplot routines. While great for plotting and line drawing, it doesn't approach the capabilities of Metawindows (tm Metagraphics). We use our Rosetta Stone Dynamic Linker to interface to Metawindows. HS/FORTH with MetaWindows makes an unbeatable graphics system. Or Rosetta to your own preferred graphics driver.

HS/FORTH provides hardware/software floating point, including trig and transcendentals. Hardware fp covers full range trig, log, exponential functions plus complex and hyperbolic counterparts, and all stack and comparison ops. HS/FORTH supports all 8087 data types and works in RADIANS or DEGREES mode. No coprocessor? No problem. Operators (mostly fast machine code) and parse/format words cover numbers through 18 digits. Software fp eliminates conversion round off error and minimizes conversion time.

Single element through 4D arrays for all data types including complex use multiple cfa's to improve both performance and compactness. Z = (X-Y) / (X + Y) would be coded: X Y - X Y + / IS Z (16 bytes) instead of: X @ Y @ - X @ Y @ + / Z ! (26 bytes) Arrays can ignore 64k boundaries. Words use SYNONYMs for data type independence. HS/FORTH can even prompt the user for retry on erroneous numeric input.

The HS/FORTH machine coded string library with up to 3D arrays is without equal. Segment spanning dynamic string support includes insert, delete, add, find, replace, exchange, save and restore string storage.

Our minimal overhead round robin and time slice multitaskers require a word that exits cleanly at the end of subtask execution. The cooperative round robin multitasker provides individual user stack segments as well as user tables. Control passes to the next task/user whenever desired.

APPLICATION CREATION TECHNIQUES

HS/FORTH assembles to any segment to create stand alone programs of any size. The optimizer can use HS/FORTH as a macro library, or complex macros can be built as colon words. Full forward and reverse labeled branches and calls complement structured flow control. Complete syntax checking protects you. Assembler programming has never been so easy.

The Metacompiler produces threaded systems from a few hundred bytes, or Forth kernels from 2k bytes. With it, you can create any threading scheme or segmentation architecture to run on disk or ROM.

You can turnkey or seal HS/FORTH for distribution, with no royalties for turnkeyed systems. Or convert for ROM in saved, sealed or turnkeyed form.

HS/FORTH includes three editors, or you can quickly shell to your favorite program editor. The resident full window editor lets you reuse former command lines and save to or restore from a file. It is both an indispensable development aid and a great user interface. The macro editor provides reuseable functions, cut, paste, file merge and extract, session log, and RECOMPILE. Our full screen Forth editor edits file or sector mapped blocks.

Debug tools include memory/stack dump, memory map, decompile, single step trace, and prompt options. Trace scope can be limited by depth or address.

HS/FORTH lacks a "modular" compilation environment. One motivation toward modular compilation is that, with conventional compilers, recompiling an entire application to change one subroutine is unbearably slow. HS/FORTH compiles at 20,000 lines per minute, faster than many languages link — let alone compile! The second motivation is linking to other languages. HS/FORTH links to foreign subroutines dynamically. HS/FORTH doesn't need the extra layer of files, or the programs needed to manage them. With HS/FORTH you have source code and the executable file. Period. "Development environments" are cute, and necessary for unnecessarily complicated languages. Simplicity is so much better.

HS/FORTH Programming Systems
Lower levels include all functions not named at a higher level. Some functions available separately.

Documentation & Working Demo	(3 books, 1000 + pages, 6 lbs)	$ 95.
Student		$145.
Personal	optimizer, scaled & quad integer	$245.
Professional	80x87, assembler, turnkey, dynamic strings, multitasker RSDL linker, physical screens	$395.
Production	ROM, Metacompiler, Metawindows	$495.
Level upgrade, price difference plus		$ 25.
OBJ modules		$495.
Rosetta Stone Dynamic Linker		$ 95.
Metawindows by Metagraphics (includes RSDL)		$145.
Hardware Floating Point & Complex		$ 95.
Quad integer, software floating point		$ 45.
Time slice and round robin multitaskers		$ 75.
GigaForth (80286/386 Native mode extension)		$295.

HARVARD SOFTWORKS

PO BOX 69
SPRINGBORO, OH 45066
(513) 748-0390

SIMPLE SCREEN DIRECTORY

DAVID CORNELL - HARRINGTON PARK, NEW JERSEY

Forth, traditionally, uses numbered screens for source code. Screens correspond, in one way or another, to the 1024-byte physical blocks around which external storage is organized. All this probably had its origin as a way to easily port systems to new hardware — once 1K units of storage were available, the system was off and running with no hassles about file formats, operating systems (if any), and all the rest. Proponents opine that this encourages small, modular organization, facilitates incremental program development, and allows easy manipulation of source code. Others point out that it is unnecessarily simplistic, isolates the Forth programmer from available editors and editing tools, and takes too much programming time to fit code into an inflexible format. Everyone is right.

Screens are a fact, an artifact of the Forth world. For the Forth programmer, even when other formats are supported, screens are old friends of questionable merit that come with the territory and remain much in evidence.

Dealing with screens can be improved considerably by using meaningful names and separating the logical screen from the physical block. `LOADSCR SQUARE-ROOT` says more than `23 LOAD`, and it doesn't matter what the block number is, even if it changes. The latter attribute is particularly useful in loader screens.

It is common to have one screen serve as a loader, with entries like:

```
3 LOAD 4 LOAD
10 LOAD 16 LOAD
```

If block four is deleted, or if a new block is inserted because it logically belongs with block three, subsequent references to 4, 10, and 16 will be wrong.

If screens are given the *logical* names 3 and 4, their physical locations don't matter; a new block that logically belongs with 3 can be named `3A`, `3.1`, or a more meaningful name. Then it can be physically moved, or not, and `LOADSCR 4` or `LOADSCR SQUARE-ROOT` will still load the same code.

> ***"Screens are old friends of questionable merit."***

Having just made a case against numbered screens, I now must say that sometimes it's just easier and more convenient to type `10 EDIT` or `10 LOAD` instead of a longer name. And referencing a screen by number makes it easier to find in a listing. Also, the requirement that all existing code be converted to another format is an unacceptable price, at least for a first pass.

So, the main specifications are:

1. Refer to screens by name for `LOAD`, `LIST`, `EDIT`, etc.
2. Support a return to the old ways, when desired.
3. Simple enough to allow additions and modifications.
4. Minimal bulk added to the main dictionary.

Data Organization

The obvious starting point is to dedicate one or more blocks to use as a directory. My Forths use DOS files, and a quick check showed there is no file for which one block would not accommodate a directory. This may not be the case if you are accessing a 40-megabyte hard disk in 1K physical Forth blocks.

The next decision is how to organize the block directory. A list of records, each record consisting of a variable-width name field and a fixed-width parameter field, has proved to be a particularly versatile data type. It is close enough, conceptually, to the standard Forth dictionary to seem familiar and easy to manipulate. The list may grow upward in memory, or downward (stack-like), with the most recent entry at the beginning. In the case of the screen directory, it really doesn't matter which way it grows. I already had routines to support lists that grow upward, and I wanted to develop words for downward-growing lists (for external vocabularies); this was an opportunity to do so. New entries are put at the beginning of the existing entries, and searches are on a last-in, first-accessed basis. The record and directory organization are shown in Figures One and Two.

Next, a symbol or convention is needed to identify and define the name of a screen to be cataloged in the directory. The only real requirement is that the symbol be ignored at compile time, but it would be nice if it could be reasonably consistent with existing practice. One Forth convention is to describe the contents of a screen in a comment on the first line. The backslash (\) is commonly used to mean "comment to end-of-line," and a colon (:) is associated with "define." Putting these together, we end up with \: on the first line of a screen, to define a name for that screen. When the blocks are cataloged (with `CAT-BLOCK` or `CAT-BLOCKS`), any that begins with `\: <name>` will be identified as a logical screen with the name `<name>`, and an entry is made in the directory.

The symbol \: is defined in the dictionary as "comment to end-of-line."

Integration

Integration with an existing Forth system simply involves:

1. Add the word \: to the Forth dictionary (and \ if it isn't already present).
2. Identify the screens to be cataloged, by placing `\: <name>` on the first line of the screen.
3. Execute `CatFile` to initialize the directory and to catalog the screens.
4. From now on, `CAT-BLOCK` or `CAT-BLOCKS` will maintain the directory.

Words that manipulate blocks and screens may be redefined to exit to `CAT-BLOCK` or `CAT-BLOCKS`. Assuming a screen editor named `EDIT`, the redefinition would be:

```
: EDIT   ( blk# -- )
  DUP EDIT CAT-BLOCK ;
```

or, if you have a word `INSERT-BLOCK` that inserts a new block, then all blocks from the point of insertion to the end of the file (or the block range) would be re-cataloged by:

```
: INSERT-BLOCK   ( blk# -- )
  DUP INSERT-BLOCK
  LastBlk CAT-BLOCKS   ;
```

Note that \: is the only word that needs to be added permanently to the resident dictionary.

Use

The most commonly used words display the directory or a screen, load a screen, and make additions to the directory.

`TELLDIR displays` the entries in the directory.

`TELLSCR <ScrName(s)>` takes the next word in the input stream as the name of the screen, searches the directory for the screen, and displays the screen. For example, `TELLSCR DOC1` looks in the directory for the entry `DOC1` and displays the screen.

`LOADSCR` takes the next word in the input stream as the name of the screen, searches the directory, and loads the block identified as the logical screen. For example, `LoadScr LOADER` looks in the directory for the entry `LOADER` and loads the appropriate block.

Additions are made by cataloging a block (or range of blocks). The cataloging routines check for conflicts caused by two blocks with the same name or by the same block with two names. Conflicts are resolved by removing the earlier of the conflicting screens from the directory and adding the more recent one.

`CAT-BLOCK` catalogs a single block.
`CAT-BLOCKS` catalogs a range of blocks.
`CatFile` reinitializes the directory and catalogs the file.

Enhancements and Extensions

These routines use the Forth screen as the logical unit. They could just as easily catalog words, instead, by searching the entire block for `:` or `CODE`, by establishing another convention to define a logical module. It would probably be necessary to allow for more than one directory block, adding another two bytes to the parameter field for the offset into the block.

If your Forth runs under a file system, then with a screen directory and a few additional words to open and close files, you have a library facility. This can be used explicitly, as in

```
LOADSCR   SQUARE-ROOT   from
MATHLIB.FTH
```

(in which case it becomes an 'include screen' facility), or with a list of unresolved references. The words that maintain the block directory can be easily adapted to maintaining other lists.

Implementation

The new words are straightforward and simple. Compatibility between Forth versions is another matter altogether. To implement these screens, please read the sections below, then check for possible problems, duplication, and equivalence in the utility and support screens in the listing. The compatibility screen should be modified for your system. Note that some words have been simplified for this listing.

NOW FOR IBM PC, XT, AT, PS2 AND TRS-80 MODELS 1, 3, 4, 4P

The Gifted Computer

1. Buy **MMSFORTH** before year's end, to let your computer work harder and faster.
2. Then MMS will reward it (and you) with the **MMSFORTH GAMES DISK**, a $39.95 value which we'll add on at **no additional charge**!

MMSFORTH is the unusually smooth and complete Forth system with the great support. Many programmers report **four to ten times greater productivity** with this outstanding system, and MMS provides **advanced applications programs** in Forth for use by beginners and for custom modifications. Unlike many Forths on the market, MMSFORTH gives you a rich set of the instructions, editing and debugging tools that professional programmers want. The licensed user gets **continuing, free phone tips** and a **MMSFORTH Newsletter** is available.

The **MMSFORTH GAMES DISK** includes arcade games (BREAKFORTH, CRASH-FORTH and, for TRS-80, FREEWAY), board games (OTHELLO and TIC-TAC-FORTH), and a top-notch CRYPTO-QUOTE HELPER with a data file of coded messages and the ability to encode your own. All of these come with **Forth source code**, for a valuable and enjoyable demonstration of Forth programming techniques.

Hurry, and the GAMES DISK will be our free gift to you. Our **brochure** is free, too, and our knowledgeable staff is ready to answer your questions. **Write. Better yet, call 617/653-6136.**

MMSFORTH

and a free gift!

GREAT FORTH:
MMSFORTH V2.4 .... $179.95*
The one you've read about in FORTH: A TEXT & REFERENCE. Available for IBM PC/XT/AT/PS2 etc., and TRS-80 M.1, 3 and 4

GREAT MMSFORTH OPTIONS:
FORTHWRITE $99.95*
FORTHCOM 49.95
DATAHANDLER 59.95
DATAHANDLER-PLUS* 99.95
EXPERT-2 .... 69.95
UTILITIES 49.95

*Single-computer, single-user prices; corporate site licenses from $1,000 additional. 3½" format, add $5/disk; Tandy 1000, add $20. Add S/H, plus 5% tax on Mass. orders. DH+ not avail. for TRS-80s.

GREAT FORTH SUPPORT:
Free user tips, MMSFORTH Newsletter, consulting on hardware selection, staff training, and programming assignments large or small.

GREAT FORTH BOOKS:
FORTH: A TEXT & REF. $21.95*
THINKING FORTH 16.95
Many others in stock.

MILLER MICROCOMPUTER SERVICES
61 Lake Shore Road, Natick, MA 01760
(617/653-6136, 9 am - 9 pm)

NGS FORTH

A FAST FORTH, OPTIMIZED FOR THE IBM PERSONAL COMPUTER AND MS-DOS COMPATIBLES.

STANDARD FEATURES INCLUDE:

- •79 STANDARD
- •DIRECT I/O ACCESS
- •FULL ACCESS TO MS-DOS FILES AND FUNCTIONS
- •ENVIRONMENT SAVE & LOAD
- •MULTI-SEGMENTED FOR LARGE APPLICATIONS
- •EXTENDED ADDRESSING
- •MEMORY ALLOCATION CONFIGURABLE ON-LINE
- •AUTO LOAD SCREEN BOOT
- •LINE & SCREEN EDITORS
- •DECOMPILER AND DEBUGGING AIDS
- •8088 ASSEMBLER
- •GRAPHICS & SOUND
- •NGS ENHANCEMENTS
- •DETAILED MANUAL
- •INEXPENSIVE UPGRADES
- •NGS USER NEWSLETTER

A COMPLETE FORTH DEVELOPMENT SYSTEM.

PRICES START AT $70

NEW◆HP-150 & HP-110 VERSIONS AVAILABLE

NEXT GENERATION SYSTEMS
P.O.BOX 2987
SANTA CLARA, CA. 95055
(408) 241-5909

Compatibilities

WORD

In fig-FORTH and my current versions, WORD does *not* return an address. WORD is, therefore, followed by HERE. Starting with Forth-79, WORD always returns an address. Check your implementation; if WORD does return an address, delete HERE from this listing when it follows WORD.

?IF

At one point, I seemed to be entering a lot of ?DUP IF and ?DUP WHILE statements in my programs, so I added a machine language primitive ?0BRANCH and the control words ?IF and ?WHILE. ?IF can be replaced by ?DUP IF, or a word ?IF can be defined to compile them as described in the listing. Note that ?DUP and IF must be individually compiled into the word with the IF ... THEN structure.

-CMOVE

If -CMOVE isn't in your system, look for <CMOVE. These words move bytes, starting from the end of the bytes to be moved instead of the beginning. To work properly with this listing, -CMOVE should be able to handle a move of zero bytes.

SCREENS, BLOCKS, and BUFFERS

A screen is not necessarily the same thing as a block, nor is a buffer. It simplifies things when they are, and many Forth systems — including mine — choose the simple route. If this is not the case with your system, see the discussion below of core words for help.

PICK and ROLL

These are zero-indexed in Forth-79 and Forth-83 systems. fig-FORTH and some others are one-indexed.

Upper- and lower-case

Forths differ in how upper- and lower-case letters are treated. I believe use of cases makes listings easier to read, so I have kept this listing as it is in my system.

Block zero

Block zero is used for the directory block in this listing. This block will not be available on all systems, and may return the address of the text input buffer (TIB). The only requirement is that the word &DirBlk return the address of the area being used for the directory. Any block or memory area can be used. See the discussion of core words, and the related screens.

&I

For my 32-bit 808x Forth, &I indicates that a 32-bit address (in the form segment-offset) is to be returned. In practice, I have also found this is a convenient mnemonic to differentiate addresses from data, so I have left it in the listing.

Core Words

The words &DirBlk, sBDE, oBdDAT, and bdPARAMS are at the root of all other words. By changing them, different-sized parameter fields, multi-block directories, memory-resident directories, and directories of different sizes can be accommodated, and the routines can be adapted to other applications.

&DirBlk

Returns the address of the directory. Block zero is used as the directory. A logical block zero may not be available; any convenient block or allocation scheme can be used. It is only required that &DirBlk return the address in memory.

sDirBlk

Returns the size of the directory. Note that a buffer and a block cannot be the same size.

oBdDat

The offset to block-directory data. This simply reserves space for a block header, and is arbitrarily set to ten.

sBDE

The size of block-directory entry. An entry consists of a string and a two-byte parameter field. The size of the entry is simply the size of the string, plus two bytes for the parameter field, plus one byte for the string's length.

bdPARAMS

The block-directory parameters. It returns the address of the start of the entries, the address of the limit of the

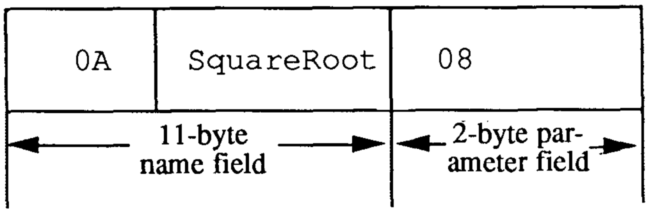


Figure One. A block-directory record has a name field (string) and a parameter field (hard-coded as two bytes, in this application). In this example, the screen named SquareRoot starts on block eight.

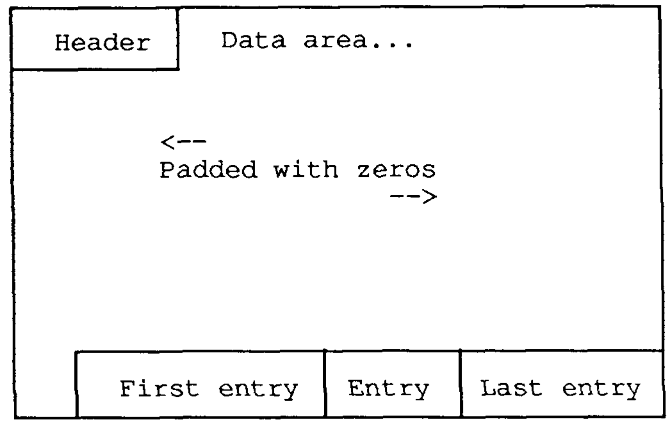


Figure Two. Block-directory format.

(Text continued on page 18.)

F83 USERS

PVM83 is a complete Prolog extension to Laxen and Perry F83. It handles the primary data structures of strings,numbers,logical constants, logical variables, compound predicates, and lists. **PVM83** is designed to add **productivity** and **flexibility**. It is fully interactive between Prolog procedures, and Forth code. PVM83 is a compiled Prolog featuring **fast** execution times.

PVM83 is fully **extensible**. "Standard" definitions gives the programmer flexibility to design just those procedures needed for his application. **PVM83** code can execute Forth words. F83 can call the **PVM83** backtracking and problem solving capabilities.

PVM83 code is **incrementally compiled** in higher memory segments than the F83 core, leaving room in the F83 kernal for the "standard" extensions or other F83 code that the programmer needs.

PVM83 is designed to keep the Forth philosophy of being both compiled, and interactive. You can type in procedures from the keyboard and test them, or supply source code from Forth block files, or text files

Intersegment memory management source code included.

PVM 83

only $69.95

includes manual

requires

DOS 2.0 or higher 256K RAM

Concept 4

PO Box 20136

VOC Az 86341

1
```
3 LOAD  6 LOAD   9 LOAD  ;S

       Screen Directory /  Directory Block
            David Cornell 08-13-1987
To implement:
     1. Check compatibility, utility and Core Word screens,
          modify as required for your system
     2. LOAD screens 3, 6 and 9
     3. Identify screens to be cataloged with
               \: <name>   on the top line
     4. execute CatFile to initialize the directory
     5. after that, use Cat-Block and Cat-Blocks to
          maintain the screen directory, and/or
     6. ReDefine existing words that manipulate screens
          to exit to Cat-Block or Cat-Blocks
```

2
```
-->     Doc USE
   TELLDIR          -  Display Directory
   TELLSCR <name>  -  display screen with <name>

   n1 n2 CAT-BLOCKS   - Catalog blocks n1 to n2, add to
          directory if identified as a logical screen by
          "\: <name>" on top line

   n  CAT-BLOCK       -  Catalog a single block

   CatFile  - initialize directory, catalog a file or range
              as defined by 1 to LastBlk

   LOADSCR <name>  - loads the screen identified by <name>
                     in the directory
```

3
```
( define \   \:     )

 (
  Skips to end of Line, all input from '\' to end of line is
  treated as a comment

  This word must be in the system. Screens are documented
  with "\" and it is use to define "\:"  below
 )

: \   C/L >IN @  Over MOD - >IN +! ;    IMMEDIATE

: \: [COMPILE] \ ;                      IMMEDIATE
```

4
```
\: Commenting Conventions

$ -  the address of a counted string, 1st byte is length byte
| -  either,  eg.  adr | 0,  = an address or 0
BD or bd - "Blk Directory" related word

Embedded "$"  -  argument is the address of a string
        eg. bd$Find,  expects a string address on the stack
Embedded "-"  -  "expects argument(s)", used occasionally (!)
       to differentiate between related words
        eg. CAT-BLOCK  CAT-BLOCKS CatFile
        CatFile does not require an argument, others do
s - as prefix,  "Size"
c - as prefix,  "Count of"
& - as prefix,  "Address, Address of"
p - ax prefix,  "Pointer to"
```

5
```
\: DOC  Compatibility

Screen 14, -CMove, move from end. Also <CMove or a "smart" CMove
     that can recognize overlap. NOTE: this word must be
     able to handle a 0 length move
Screen 6, WORD HERE, This is FIG Compatible, other
     Forths will PROBABLY return an address after
     WORD.  In that case, delete 'HERE' when is follows 'WORD'

Screen 15  (bdCONFLICT) uses the truth value for arithmetic
     and requires that TRUE = 1, if TRUE is -1 on your
     system, add ABS after each of the two compares
Screen 17     >2R,    2R>
      Same as >R >R,  R> R>

Pick, Roll are 0 relative. OK for 79/83. FIG is 1 relative
```

6
```
\: System Dependent

: LastBlk  \  -- nBlk,  returns last blk in file or range
     ************ ;
: Word$ \  Delim -- $ , as WORD but returns string address
 \  ** simplified for this listing **
  \ if only used at compile time, WORD may be sufficient
   \  ** for Forth79/83, HERE  not required below **
   ?EXEC WORD HERE PAD $! PAD  ;

: WndColsMax  --  n, returns Max Cols in current display window
  \  general purpose default is width of CRT screen
   80 ;
: .SCRN  \  nScr -- ,  favorite screen display routine,
   LIST ; \   general purpose default
                                                      --
```

```
---- 7 ----
\: Utility

: Tuck   \  n1 n2  --  n2 n1 n2,  opposite  of "Over"
     Swap Over ;
: BOUNDS  \  Adr Len  --  Limit Index,  sets up DO Loop
     Over + Swap ;

: &I  ?COMP COMPILE I ;                    IMMEDIATE  \  see text

: ?IF   \  same as ?Dup IF
    ?COMP Compile ?Dup  [COMPILE] IF ; IMMEDIATE  \ see text

: Count+  COUNT + ;

: 0! 0 SWAP ! ;                                  -->
```

```
---- 8 ----
\: Utility2  / String Support

: $LEN   \  $ -- n,   returns length of a counted string
    C@ ;
: $!  \  $ adr, store string at adr, stores len byte
    Over $Len 1+ CMove ;
: $Move  \ $  adr ,  move str to adr, length byte NOT moved
    Swap Count Rot Swap Cmove ;

: $.     \ $    print a Counted String
    Count Type ;
: $=    \  $1 $2 -- t/f,     abs compare    see text
    Dup $Len  1+ Swap  -Text 0= ;
: $==    \ $1 $2 -- t/f,   compare strings, ignore case
    $= ;  \   u/l case not supported this listing
```

```
---- 9 ----
\: CORE WORDS   - some system dependent
10 CONSTANT oBdDAT          \  offset to data in blk directory

: &DirBlk 0 BLOCK ;         \  address of Directory Block
: sDirBlk B/Buf 2 ;         \  size of Directory Block
: sBdE  \  &AdrNameField  --  SizeBlkDirEntry
      $Len 3 + ;

: -ZC   \ adr n  -- adr c ,  skips leading 0's, Count of 0's
   Over >R 0 DO Count IF 1- LEAVE THEN LOOP Dup R> -  ;

: bdParams  \   --  &Entries  &Limit  cAvailBytes,
   &DirBlk oBdDat +             \  Start Address Data
   sDirBlk oBdDat -             \ max size data
   2Dup + >R
   -ZC  R> Swap ;     -->       \ &Names sAvail
```

```
---- 10 ----
\: bd$Find


: bd$Find  \ $ -- &Entry ! 0
  bdParams  Drop Swap
  DO
     Dup &I $==
     IF Drop &I 0 LEAVE THEN
      &I sBDE
  +LOOP  IF FALSE THEN ;






                                                        -->
```

```
---- 11 ----
\: NameToNumber Words ,
: $->BLK  \ $ -- BlkNumber
        bd$Find ?IF COUNT+ @  ELSE  0 THEN ;
: ?$->BLK
      $->BLK DUP FAILS ABORT" Screen not in Directory " ;
: .$SCR   \ $ --  ,    display screen identified by $
      $->BLK ?IF .SCRN THEN ;
: $LOAD   \  $ --  ,  loads screen identified by $
     ?$->BLK LOAD ;
: BLK>   \    -- n  ,  takes next word as blk identifier,
      BL WORD$ ?$->BLK ;  \  eg.  BLK> LOADER
: LOADSCR     \  --  ,  next word is name of screen to load
      BLK> LOAD ;  \    use:     LoadScr <ScreenName>
: TELLSCR  BLK> .SCRN ;  \ use:, TELLSCR <ScreenName>
                                                   -->
```

```
---- 12 ----
\: TellDIR vii -- cursor carried TOS, requires WndColsMax

: TellDir \    --    ,  prints dir entries in 16 col field
  WndColsMax 1- bdParams Drop Swap \ cursor posn carried TOS
  DO
    Dup                         \  copy current csr posn
    -16 AND 16 + &I $Len +      \ cursor posn after print
     WndColsMax  /MOD           \  exceeds Wnd Line Len ?
          IF Swap Drop CR       \  Yes, start new line
          ELSE Dup -16 AND      \  No,
             Rot - Spaces       \   Pad to start of field
          THEN
     &I Count WndColsMax Min    \  don't wrap narrow window
     TYPE
   &I  sBDE                     \ +size this entry -> next
   +LOOP Drop ;                            -->
```

Screen 13

```
\: bdAdd                          * pick 0 rel

: bdAdd  \ n $  --  ,  add $ to BlkDir with Param n
   Dup sBDE  bdParams   \  n $ sReq  pEntries &Limit nAvail
        3 PICK <
        ABORT" No Room In Directory Block"
   Drop    \ limit not used, n $ s pE
   Swap -  \  n $ Dest,   final dest = new start of bdNames
   Tuck $!
       Count+ !  UPDATE ( FLUSH ) ;
                                                       -->

**  Caller must FLUSH buffers. This assumes that an UPDATEd
**  buffer will not be reused until it is flushed. If in doubt,
**  include FLUSH in bdAdd. The same reasoning applies to
**  bdDel in the next screen.
```

Screen 14

```
\: bdDel   bd$Delete
\ src = pFirstEntry,  Dest = pFirstEntry  + sEntry,
\ count = pEntry - pFirstEntry, requires smart Cmove or -Cmove

: bdDel  \ &BdNF --    ,   delete entry at address
  >R bdParams 2Drop  \  adr 1st Entry
  R@ sBDE
  2Dup Over +        \  pEntries sEntry Src Dest
    Over R> Swap -   \  pEntries sEntry Src Dest Count
    -CMove           \   ** -CMove must handle  0 Len move **
  Erase              \  maintain leading 0's
  UPDATE ;           \   ** caller guarentees FLUSH

: bd$Delete \ $ ---    ,   search for entry, delete if found
  bd$Find ?IF bdDel THEN ;                           -->
```

Screen 15

```
\: bdCONFLICT                     * pick 0 rel

: bdCflct?  \ n  $      &Entry  Limit  -- pEntry flag | FALSE
           \ flag: 2 Same, 1 Changed, 0 doesn't exist
  Swap DO
       Over &I Count+ @ =  \  **  requires TRUE = 1 **
       Over &I $= +          \  **  requires TRUE = 1 **
       ?IF  -Rot 2Drop       \  ** add ABS after = and $= **
            &I Swap  0  LEAVE \  preserve flag
       THEN
  &I sBDE
  +LOOP                      \    n $adr | Flag  0
  IF Drop False  THEN ;

: bdConflict?  \ n  $  -- pEntry flag | FALSE
  bdParams  Drop bdCflct?  ;                 -->
```

Screen 16

```
\: AddToBd     bdTell

: bdTell \ bdName  --  ,    tell Name & Blk of entry
  Dup $. ."  -  "  Count+ @ U. ;

: AddToBd  \ nBlk $  -- adds entry, checks for conflict
          \  prints message with old params if replaced
          \  conflict may come from same name or same blk#
  BEGIN 2Dup BdConflict?
     Dup 1 =
   WHILE Drop
     Dup   CR ." Replacing "  bdTell
     bdDel
  REPEAT
  0= IF  bdAdd ELSE Drop 2Drop THEN ;                      -->
```

Screen 17

```
\: CATALOG     -- add to blk dir if flagged as Dir Entry Screen
: (catblk)   \  n --  ,  catalog block n
    Blk @ >IN @  >2R
    Blk ! >IN 0!             \  check top line of screen
    BL WORD$     " \:" $=     \  for identifier ...
     IF Blk @  Bl WORD$ AddToBD \  If found, add entry to BlkDir
     THEN  2R> >IN ! Blk ! ;
: CAT-BLOCK   \ Blk  --
       (catblk) FLUSH ;
: CAT-BLOCKS   \  Blk1 Blkn --
      1+ Swap DO I (CatBlk) LOOP FLUSH  ;
: IniBd  \  --  , initialize Blk Dir, erases blk, writes header
      &DirBlk sDirBlk Erase
      " \: BD" &DirBlk $MOVE UPDATE FLUSH ;
: CatFile  \  --  , catalog entire file
      IniBd   1  LastBlk  Cat-Blocks ;
```

STANDALONE APPLICATIONS

JAMES F. BALL - COLUMBUS, OHIO

This article describes how to generate a self-executing application for any F83 program. In order to accomplish this task, a modified Forth system containing your application is created through the meta-compilation process.

Background

I have no experience with other programming languages, and am a self-taught Forth user (Laxen and Perry's F83 on an IBM PC). My main textbooks for learning Forth have been *Starting Forth* by Leo Brodie, *Mastering Forth* by Anderson and Tracy, *Inside F83* by C.H. Ting, Ph.D., and *Forth Dimensions*. I have also found that my hard copy of the various F83 screens (including shadow screens) proves as valuable as a written manual.

"Creating a standalone application is not so complicated..."

One problem I encountered as a result of choosing Forth as my first programming language, was that once I learned the fundamentals, finding F83-specific or machine-specific guidance was difficult. I wanted to create a self-executing application in F83, but I was uncertain that I could accomplish this task, based on my limited Forth background.

Standard System Load Screen # 1

```
Scr # 1        A:EXTEND86.BLK
( Load Screen to Bring up Standard System          07Apr84map
)   CR  .( Loading system extensions.)  CR
2 VIEW#       ( This will be view file# 2 )
WARNING OFF

   3 LOAD     ( BASICS )
   6 LOAD     ( FILE-INTERFACE )
   FROM CPU8086.BLK    1 LOAD ( Machine Dependent Code )
   FROM UTILITY.BLK    1 LOAD ( Standard System Utilities )

WARNING ON
-->
```

Modified Load Screen # 1

```
Scr # 1        A:EXTEND86.BLK
\ Load Screen to Bring up Application System         JFB880601
  CR .( Loading application )  CR
2 VIEW# !  ( This will be view file# 2 )
WARNING OFF

  3 LOAD    ( BASICS )
  6 LOAD    ( FILE INTERFACE )
  FROM CPU8086.BLK       1 LOAD ( Machine Dependent Code )
  FROM UTILITY.BLK       1 LOAD ( Standard System Utilities )

  FROM APPL.BLK    1 LOAD ( Loads your application )

WARNING ON

-->
```

Metacompilation

As it turned out, creating a self-executing application was not so complicated. The solution came primarily from *Inside F83*. The process of metacompilation and running the metacompiler is described in chapter 25. Metacompilation is the process of creating a new system out of the existing system. This allows one to create a modified Forth kernel (KERNEL.COM), which is necessary for the creation of a standalone application.

In order to create the metacompiled KERNEL.COM (which will later be used to generate your application), the following F83 files (only) should be copied onto a working diskette:
F83.COM
META86.BLK
KERNEL86.BLK

Next, open the META86.BLK in F83, and load the first block as follows:

```
A>F83 META86.BLK
1 LOAD
```

This begins the metacompilation process. After the process is complete, the KERNEL.COM file will be created on the working disk (check your directory). The 12K KERNEL.COM is a stripped-down version of F83 and becomes the core of your application.

Creating Your Application

If you aren't using the high-density, 3.5" diskettes á la PS/2, you'll need to take steps to avoid running out of space on your disk. After you have created the KERNEL.COM file on the working disk, return to DOS and erase the KERNEL86.BLK and the META86.BLK from the working disk, in order to conserve disk space. Then, copy EXTEND86.BLK, CPU8086.BLK, UTILITY.BLK, and your application (APPL.BLK) onto the working disk.

At this point, the following files should be on the working disk:
F83.COM
KERNEL.COM
EXTEND86.BLK
CPU8086.BLK
UTILITY.BLK
APPL.BLK (containing your F83 application)

Next, open EXTEND86.BLK using F83, and make the modifications shown in Figure One to standard system load screens 1 and 2:

```
A>F83 EXTEND86.BLK
1 EDIT
```

Your application should be written so that it will load from the first block. By using the `FROM` command (see modified load screen 1, line 10), your application is loaded. The technique of using `FROM` and `1 LOAD` to load your application is modeled after the method used by Laxen and Perry to load the utility and CPU system extensions (see standard system load screen 1, lines 7 – 8 in Figure One).

Standard System Load Screen # 2

```
Scr # 2        A:EXTEND86.BLK
 0 \ Load up the system                                   08MAY84HHL
 1 : HELLO   ( S -- )
 2   CR ." 8086 Forth 83 Model "
 3   CR ." Version 2.1.0 Modified 01Jun84 "
 4   START    ONLY FORTH ALSO DEFINITIONS   ;
 5 ' HELLO IS BOOT
 6 \ 13 LOAD   ( Configuration: change and load as desired. )
 7
 8 : MARK    ( S -- )
 9   CREATE     DOES> (FORGET)  FORTH DEFINITIONS   ;
10 MARK EMPTY      HERE FENCE !
11
12 CR .( System has been loaded, Size = ) HERE U.
13 SAVE-SYSTEM F83.COM
14 CR .( System saved as F83.COM )
15
```

Modified Load Screen # 2

```
Scr # 2         EXTEND86.BLK
 0 \ Load up your application                             JFB880601
 1 : HELLO    (S -- )
 2    START   ONLY FORTH ALSO DEFINITIONS
 3            RUN-APPL ; \ Where RUN-APPL executes
 4                            \ a program in APPL.BLK
 5 ' HELLO IS BOOT
 6
 7 : MARK    (S -- )
 8    CREATE    DOES> (FORGET)  FORTH DEFINITIONS   ;
 9 MARK EMPTY     HERE FENCE !
10
11     SAVE-SYSTEM  APPL.COM  \ Where APPL is a unique name
12                            \ for the program.
13
14     BYE       \ Exit F83; Type APPL to run your application.
15
```

The next step is most crucial, to make your application self-executing. In the modified load screen 2, the application word which causes your program to execute (using `RUN-APPL` as an example) is added into `HELLO`, and a unique name (for example, APPL.COM) is assigned with `SAVE-SYSTEM` (line 13).

Finally, open the modified EXTEND86.BLK using the newly generated KERNEL.COM, then load the first block via the following command:

```
A>KERNEL EXTEND86.BLK
1 LOAD
```

Once the loading is complete, the application APPL.COM will be created on the working disk, ready to run by typing the application's filename at the DOS prompt:

```
A>APPL
```

Conclusion

Your program will now run as a self-standing, executable application. It can be copied to another disk for distribution without the F83 system file. The F83 kernel itself (KERNEL.COM) is incorporated as part of your APPL.COM. The kernel consumes about 25K of disk space. However, by further editing out non-essential Forth words from EXTEND86.BLK and the related CPU8086.BLK and UTILITY.BLK (prior to creating the KERNEL.COM), the application's size can be reduced. For this reason, a customized KERNEL.COM might be created for each application.

Finally, by creating a simple user interface that prevents access to the F83 system (by limiting the vocabulary and providing appropriate error checking), one can create a professional, standalone application.

Application Blocks (APPL.BLK)

```
Scr # 1
\ TAX TIPS: INTRODUCTION
: LOCKUP   0 -2 AT ;
  VARIABLE ROW    1 CONSTANT .LEFT   79 CONSTANT .RIGHT
: RESTORE    1 ROW ! ;
: SIDES    ROW @ 2+ DUP ROW ! AT ;
: SPOT   219 EMIT SPACE ;
: LSIDE    RESTORE  8 0 DO .LEFT  SIDES SPOT   LOOP ;
: RSIDE    RESTORE  8 0 DO .RIGHT SIDES SPOT   LOOP ;
: BAR   40 0 DO SPOT LOOP ;
: TOP       1  1  AT BAR ;
: BOTTOM    1 19  AT BAR ;
: INTRO    24    7 AT ." INCREDIBLE SOFTWARE PRESENTS "
           24    9 AT ."          TAX TIPS            "
           24   15 AT ."       Programmed in F83      "
           24   17 AT ."         By James F. Ball"  LOCKUP ;
: BOX   DARK TOP LSIDE RSIDE BOTTOM ;    -->
```

```
Scr # 2
\ TIP ONE: IRA DEDUCTION
: TITLE
    11 3 AT  ." TAX TIP 1: IS YOUR IRA CONTRIBUTION DEDUCTIBLE?"
    11 5 AT  ." Answer the questions below by entering either: "
    11 7 AT  ." Y = Yes "  11 8 AT  ." N = No " ;
: REDO   DARK TITLE  11 11 AT ;
: UNSTACK   DEPTH 0 ?DO DROP LOOP ;
: Y/N   BEGIN  KEY 95 AND  DUP ASCII Y = SWAP ASCII N =
              2DUP OR UNTIL DROP ;
: PLAN?   REDO ." Are you or your spouse covered" CR 11 SPACES
                 ." by a retirement plan at work? " Y/N ;
: AMOUNT?   REDO ." Is your adjusted gross income " ;
: RERUN?   11 19 AT ." Press  Esc  to quit or" 11 20 AT
                     ."  any key to repeat." ;
: NO-PLAN   REDO
      ." Your IRA contribution is 100% tax deductible" ;  -->
```

```
Scr # 3
\ TIP ONE: IRA DEDUCTION
: NO-DEDUCTION   ." Your IRA contribution is not deductible." ;
: PARTIAL-DEDUCTION
   ." Your deduction is between $200 and $1,990." ;
  DEFER RETURN1   DEFER RETURN2   DEFER RETURN3
: JOINT1        ." more than $50,000? " ;
: JOINT2        ." $40,050 - $49,999? " ;
: JOINT3        ." less than $40,050? " ;
: SINGLE1       ." more than $35,000? " ;
: SINGLE2       ." $25,050 - $34,999? " ;
: SINGLE3       ." less than $25,050 " ;
: RESET    ['] SINGLE1 IS RETURN1   ['] SINGLE2 IS RETURN2
           ['] SINGLE3 IS RETURN3  ;
: RETURN?   REDO  ." Are you filing a joint return? " RESET
     Y/N IF ['] JOINT1 IS RETURN1   ['] JOINt2 IS RETURN2
            ['] JOINT3 IS RETURN3   THEN ;   -->
```

(Screens continued on page 37.)

BRYTE FORTH

for the

INTEL 8031

MICRO-CONTROLLER

FEATURES

- —FORTH-79 Standard Sub-Set
- —Access to 8031 features
- —Supports FORTH and machine code interrupt handlers
- —System timekeeping maintains time and date with leap year correction
- —Supports ROM-based self-starting applications

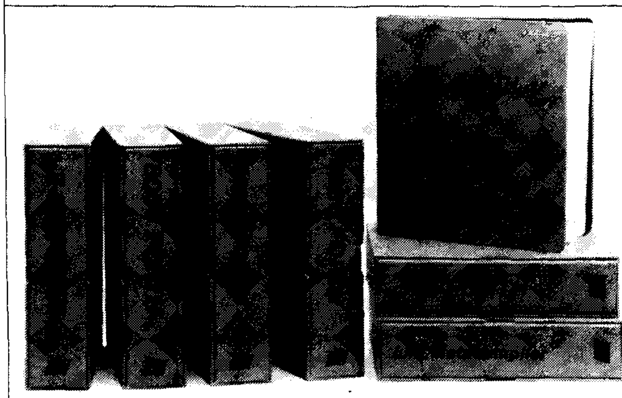
COST

130 page manual —**$ 30.00**
8K EPROM with manual—**$100.00**

Postage paid in North America.
Inquire for license or quantity pricing.

Bryte Computers, Inc.
P.O. Box 46, Augusta, ME 04330
(207) 547-3218

TOTAL CONTROL with *LMI FORTH*™



For Programming Professionals: an expanding family of compatible, high-performance, Forth-83 Standard compilers for microcomputers

For Development: Interactive Forth-83 Interpreter/Compilers
- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 400 page manual written in plain English
- Options include software floating point, arithmetic coprocessor support, symbolic debugger, native code compilers, and graphics support

For Applications: Forth-83 Metacompiler
- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8086, 68000, 6502, 8051, 8096, 1802, and 6303
- No license fee or royalty for compiled applications

For Speed: CForth Application Compiler
- Translates "high-level" Forth into in-line, optimized machine code
- Can generate ROMable code

Support Services for registered users:
- Technical Assistance Hotline
- Periodic newsletters and low-cost updates
- Bulletin Board System

Call or write for detailed product information and prices. Consulting and Educational Services available by special arrangement.

LMI ***Laboratory Microsystems Incorporated***
Post Office Box 10430, Marina del Rey, CA 90295
Phone credit card orders to: ***(213) 306-7412***

Overseas Distributors.
Germany: Forth-Systeme Angelika Flesch, Titisee-Neustadt, 7651-1665
UK: System Science Ltd., London, 01-248 0962
France: Micro-Sigma S.A.R.L., Paris, (1) 42.65.95.16
Japan: Southern Pacific Ltd., Yokohama, 045-314-9514
Australia: Wave-onic Associates, Wilson, W.A., (09) 451-2946

(Continued from page 6.)

```
: OR ( s1 s2 -- or )
  2SH->D ROT
  OR >R OR >R !SHADOW ;

: XOR  ( s1 s2 -- xor )
  2SH->D ROT
  XOR >R XOR >R !SHADOW ;
```

The redefinition of any of the Forth primitives can be extended in this way. In any speed-critical application, these should be recoded in assembly language.

Darrel Johansen
Orion Instruments
702 Marshall Street

(Continued from page 11.)

entries, and the number of available bytes. The start and the limit are used to set up `DO ... LOOP` searches; the count of available bytes is used for memory management.

Optimizing

Each time the block directory is accessed, a search is made for the first entry, skipping over leading zeros. This is done by the high-level word `-ZC`. This can be recoded in assembly language or replaced by a variable maintained in the directory header. I have a definite prejudice against maintaining (i.e., trying to maintain) flags and variables, and I prefer code that figures things out from existing information (whenever the tradeoffs aren't too onerous).

Conclusion

A lot of ideas that seem clever at the time, end up not being used, either because they weren't so clever after all, or because they were too clever and confused the programmer. These Forth words have survived initial enthusiasm and have lasted long enough — without unduly confusing the programmer — to prove their value.

USING REGISTERS IN DATA STACKS

DON KENNEY - SAN DIEGO, CALIFORNIA

Usually, Forth systems implement a data stack in memory. Because many widely used microcomputer CPUs handle register operations much more rapidly than the analogous memory operations, several people have suggested keeping the data stack in registers, instead. There are real problems with that approach. But there is another possibility worth investigating — that of keeping the top few stack elements in registers, and the remainder in memory. As this paper shows, such mixed stacks can be much faster than pure memory stacks.

First, let's look at memory-based data stacks. The problem here is that many CPUs don't handle them efficiently. For example, an Intel 8088 performs an inter-register transfer in two clocks and an inter-register ADD in three clocks. On the other hand, POPs from memory require 12 clocks, and PUSHes require 15. ADDing a register to memory requires 20 clocks. Thus, a simple Forth + implemented as *POP, ADD-to-memory* requires 32 clocks. If the + could somehow be done in registers, it would require many fewer clocks.

Not surprisingly, some people have looked at keeping the data stack in registers. There are two problems with this. First, there usually aren't enough registers. A three- or even six-register stack isn't large enough to support complex applications, much less recursive algorithms. Second, the use of any significant number of registers in a data stack introduces another inefficiency, in that data has to be moved register-to-register through every stack register whenever the stack length changes, unlike a memory stack. Even though inter-register transfers are fast, a register-based stack of reasonable size would require far too many of them.

Let's take a look at a third alternative — keeping the top of the stack in registers, and the remainder in memory. Table One tabulates the number of memory and register operations required to execute the most common Forth run-time words, for stacks with varying numbers of words in memory. As close examination will show, the effect of putting part of the stack in memory is different for different words. For example, DROP is almost certainly most efficient on a pure memory stack, while ROT is almost as certainly going to be better with several registers atop the stack.

"Keeping the top word in a register is preferable..."

Table Two tells us that fast memory-based stack operations are necessary if one expects Forth to run fast. This leads one to expect that a dedicated chip like the NC4016 could substantially outperform a general-purpose CPU with superficially better specs when running Forth, unless the general-purpose machine happens to be optimized for stack handling. The single surprise in the table is that keeping the top stack word in a register is slightly preferable to a pure memory stack even when memory operations take no more time than register operations.

The tabulated data shows with higher memory:register speed ratios, more stack words should be in registers. But it also says that one word in a register is better than (or almost as good as) two, and is never worse than three — even for the 8088, which has about as large a disparity between register and memory operations as one is likely to encounter.

Word frequencies are based on "F83 Word Usage," by C.H. Ting (*Forth Dimensions* VII/4). Counts were run on seven unspecified F83 files with 230 code screens.

Numbers for memory and register operations are based on simple algorithms that superficially look right. They weren't tested, or even examined very deeply. There may be a clever (or obvious) way to cut the number of operations. Some necessary operation may have been forgotten. A lot of analysis might change the numbers slightly, but it's unlikely that it would alter the conclusions.

Timing for an inter-register transfer (e.g., Intel's MOV reg,reg) is used for a register operation. An average of PUSH and POP times was used for a memory operation. Some instruction sets contain operations which allow a memory operation to be combined with a logic/math operation (e.g., ADD reg,mem) so a timing-optimized set of basic Forth words can and should effectively improve the raw access ratios computed above. Perhaps, in practice, an 8086 memory:register speed ratio is only 3:1.

Examination of Table Three shows pretty clearly that, for a machine with no speed penalty for accessing memory, a pure memory stack is fine, and that one word in a register is about as good. For real, general-purpose CPUs which often perform inter-register operations much more quickly than memory accesses, it appears to pay to carry the top stack element in a machine register. For some CPUs, it might

Word	freq.	pure-mem mem	pure-mem reg	1 register mem	1 register reg	2 registers mem	2 registers reg	3 registers mem	3 registers reg	4 registers mem	4 registers reg
@ C@ HERE	0.163	1	0	1	1	1	2	1	3	1	4
DROP	0.049	0	1	1	1	1	1	1	2	1	3
ROT	0.018	6	0	4	1	2	2	0	4	0	4
DUP	0.110	2	0	1	1	1	2	1	3	1	4
+ - * AND	0.163	3	0	1	1	1	1	1	2	1	3
SWAP	0.069	4	0	2	2	1	1	0	3	0	3
OVER	0.045	3	1	2	1	1	2	1	3	1	4
!	0.065	3	0	2	0	2	0	2	1	2	2
R> >R	0.049	1	0	1	0	1	1	1	2	1	3
0 1 2 3	0.200	1	0	1	1	1	2	1	3	1	4
1+ 2+ 2*	0.064	2	0	0	0	0	0	0	0	0	0
Weighted sums:		1.98	0.10	1.17	0.89	1.02	1.41	0.91	2.43	0.91	3.28

Table One. Register and memory operations required for common Forth words.

		pure-mem	1 register	2 registers	3 registers	4 registers
1:1	6809	2.071	2.063	2.429	3.346	4.194
2:1		4.047	3.235	3.449	4.260	5.107
3:1		6.023	4.408	4.470	5.173	6.021
4:1		7.999	5.581	5.490	6.087	6.935
5:1	80186, 8086	9.975	6.754	6.510	7.000	7.848
6:1	8088	11.95	7.926	7.531	7.914	8.762

Table Two. Operation time with N registers in stack, for various memory:register operation speed ratios.

		pure-mem	1 register	2 registers	3 registers	4 registers
1:1	6809	1.00	1.00	1.17	1.62	2.03
2:1		1.00	0.80	0.85	1.05	1.26
3:1		1.00	0.73	0.74	0.86	1.00
4:1		1.00	0.70	0.69	0.76	0.87
5:1	80186, 8086	1.00	0.68	0.65	0.70	0.79
6:1	8088	1.00	0.66	0.63	0.66	0.73

Table Three. Operation time with N registers in stack, relative to pure memory stack, for various memory:register speed ratios.

even pay to put the top two stack elements in registers.

Let's look at two real-world examples.

For the Intel 808x CPUs, the stack-manipulation time saved by using a data stack with two words in registers, instead of in pure memory, can be expected to be between 25 – 35%. Since Forth spends a good deal of its time doing non-stack-manipulative things, like jumping around memory and actually performing operations on data, the expected time saved by using a combined register-and-memory stack will be less. Depending on how inefficient the other operations are, a 5 – 15% overall improvement seems a reasonable expectation.

On the other hand, the Motorola 6809 inter-register operations (e.g., LEA_ 0,reg) are only slightly faster than memory accesses. The 6809 also has efficient auto-increment/decrement memory address modes for handling stacks during operations on data. Moreover, the 6809 does not allow inter-register ADDs or MULs. We probably would spend some time analyzing before implementing a 6809 data stack with a word in a register. We wouldn't use more than one 6809 register, and we wouldn't expect more than 1 – 3% overall performance improvement thereby.

If there are any surprises in the above analysis, they are that keeping the top stack word in a register will probably yield results which are either optimal or near optimal, no matter how efficient or inefficient register operations are, compared to memory operations. To put it a little differently, if you're writing a Forth interpreter from scratch and don't want to do a detailed analysis of optimal stack structures, put the top data stack element in a register; your stack handling will then probably be about as efficient as it can be.

Donald Kenney says that, like other FIG members, he started off to write his own Forth kernel. He got sidetracked by the material presented here, and his kernel still isn't running.

SDS FORTH for the INTEL 8051

Cut your development time with your PC using *SDS* Forth based environment.

Programming Environment

- ☐ Use your IBM PC compatible as terminal and disk server
- ☐ Trace debugger
- ☐ Full screen editor

Software Features

- ☐ Supports Intel 805x, 80C51FA, N80C451, Siemens 80535, Dallas 5000
- ☐ Forth-83 standard compatibility
- ☐ Built-in assembler
- ☐ Generates headerless, self starting ROM-based applications
- ☐ RAM-less target or separate data and program memory space

SDS Technical Support

- ☐ 100+ pages reference manual, hot line, 8051 board available now

Limited development system, including PC software and 8051 compiled software with manual, for $100.00. (generates ROMable applications on top of the development system)

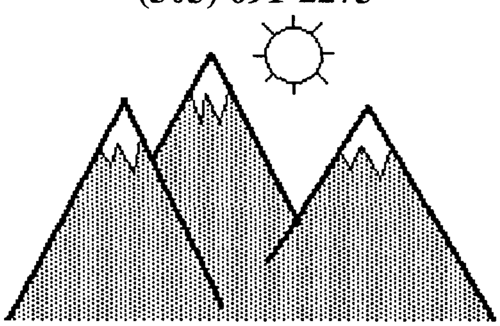
SDS Inc., 2865 Kent Avenue #401, Montreal, QC, Canada H3S 1M8 (514) 461-2332

CONSULTANTS

CIBER, a national consulting firm, has Forth assignments in the Denver area. If you are looking for a change, and the Rocky Mountians appeal to you, please give us a call or send your resume to:

CIBER

Beth Kern, Recruiter
4100 E. Mississippi Ave., Suite 1810
Denver, CO 80222
(303) 691-2273



SOFTWARE ENGINEER

Join us at Advanced Energy Industries, Inc., the technical leader in the design and manufacture of high reliability plasma, ion and magnetron power processors for the semi-conductor and thin film industry.

Our phenomenal growth rate requires an experienced software designer who has specific experience in microprocessor application software.

The successful candidate will have 3 years control and instrumentation programming experience, one year of which must include FORTH. BSEE preferred, with minimum of AA degree + equivalent experience.

Excellent salary and comprehensive benefits. Send resume and salary history to Human Resources Department, 1600 Prospect Parkway, Fort Collins, CO 80525.

ADVANCED ENERGY INDUSTRIES, INC.
EOE/M/F/H/V

AE ADVANCED ENERGY®

Forth-83

MENU-DRIVING THE 8250 ASYNC CHIP

PAUL COOPER - CHATSWORTH, CALIFORNIA

Depending on how many serial ports your PC will support, you can expand or shrink this program to fit. The X-16 supports COM1, COM2, COM3, and COM4, the starting PC addresses of which are (hex) 3F8, 2F8, 3E8, and 2E8, respectively. Most PC clones support only COM1 and COM2. You will probably, in your entire life, never need more than two serial ports. This program does not concern itself with any interrupt status; we are running in half-duplex mode, which is standard for on-the-air communication on either HF or VHF amateur radio bands. This initialization routine is part of an RTTY program I wrote for my amateur radio station (K6PY).

The registers of the 8250 with which we concern ourselves are LSR (line status register), LCR (line control register), DATAL (low data byte), and DATAH (high data byte). Table One shows the relationship of addresses and serial ports.

Two bits are necessary to monitor the line status register, to determine whether there is incoming data or whether the transmitter holding register is empty. They are, respectively, the data-available bit (hex 01) and the transmitter-holding-empty bit (hex 20). These two bits are monitored in the words which query the port for data coming in and which send a character out to the port to be transmitted. We create two constants for these bits, `DAV` and `TBE`. Variables are used for the registers, and a variable `B/A` is named to designate whether you want to run Baudot or ASCII. The program automatically sets Baudot at five bits, 1.5 stop bits, and no parity. You may, however, select any baud rate from 45.45 up to 56K baud (but who would want to run Baudot at that rate?). You can see these constants and variables listed in screen three. One can even run slower than 45.45, but who cares?

In screen four is the word `SELECT_ADDR`. This word places the necessary register addresses in our variables, upon the selection at the keyboard. `ASCII/BAUDOT?` in screen five places a low flag in `B/A` for ASCII selection, or a high (i.e., true) flag for Baudot, with automatic bitwise selection of word length and parity.

Screen six is a case word, `BAUDCASE`, which leaves the hex representation of the numerical divisors necessary to generate a 16X clock. It is assumed that your applicable clock is using a 1.8432 MHz. crystal. In the word `INITCOM`, which is the main word in screen 11, `BAUDCASE` leaves two values on the stack which are port stored in `DATAL` and `DATAH`. These two values are relative to baud rates and can be seen in screen seven, which holds the word `BAUDREQUEST`. This word interactively accepts the rate selected from the keyboard.

`WORDREQUEST` in screen ten allows the operator to select any combination of word length, stop bits, and parity (only for ASCII). As stated earlier, this function is automatically set when Baudot is used. `WORDCASE` leaves the value on the stack relative to word length, stop bits, and parity. Then it is port stored in `LCR`, as shown in screen 11, when `INITCOM` is running. I did not make available a six-bit-word function, even though the 8250 has that provision. You can see how it is done from Western Digital Corporation's excellent data book on this chip and others that it manufactures. The only difference is that bits zero and one of the hex values, as shown in `WORDCASE` (screen nine), would be changed to a value of one for bit zero, and zero for bit one.

To use the routine, the word `INITCOM` is entered, which prompts the operator for all data. You will see the phrase `80 LCR @ PC!` on line four of screen 11. To begin initialization of the 8250 chip, an 80 (hex) must be written to the line control register. This is akin to a reset — it toggles the divisor latch access, so that the chip knows it is going to get new data. Unless you do this first, as in the program, all will be for naught.

I have included in screen 12 the basic receive and transmit words associated with my RTTY program. `?SIO` queries the COM port selected and, if there is a character there, it leaves a true flag; if not, a false flag is placed on the stack. `SKEY` brings the character to the stack is the flag is true. The sequence would be:

```
?SIO IF SKEY THEN
```

If a character was brought in, an additional `EMIT` or other action word would perform a task. I merely use this to emit the character to the screen when receiving an out-station's data; but one could `DUP` it and send it to a printer or, possibly, to long storage. Long storage could be polled until the count reached 1024 characters, and then the group could be written to disk. (My hard copy comes from the ASR-28 teletype machine.)

The word to send a character out the port is `SEMIT` and its basic structure is that of a `BEGIN ... UNTIL` loop. The character to be transmitted is placed on the stack, the transmit-buffer-empty bit is `AND`ed with the data derived from a port fetch of the line status register; if the result

Screen # 2

```
( RTTY - Array words                          pac 16:47 09/10/86 )

: INDARR \ n cells --- <name>...creates indexed array
   CREATE 2* HERE OVER ERASE ALLOT ;CODE
   AX, 2 [BX] LEA  BX POP  AX, BX ADD  AX, BX ADD
   AX PUSH NEXT,  END-CODE

VARIABLE $DUMMY

: << ( mark stack top, to fill indexed array) SP@ $DUMMY ! ;

: >>  \ mark end of fill then fill array
   $DUMMY @ SP@ - 6 - OVER + DO I ! -2 +LOOP ;

: DOWNPAGE   10 0 DO CR LOOP ;
-->
```

Screen # 3

```
( RTTY - Constants and Variables              pac 16:03 09/12/86 )

HEX
01 CONSTANT DAV     \ data available bit
20 CONSTANT TBE     \ transmit holding register empty bit
DECIMAL

VARIABLE B/A     \ Baudot [-1] or Ascii [0] tx/rx mode
VARIABLE LSR     \ line status register of 8250 chip
VARIABLE LCR     \ line control register of 8250 chip
VARIABLE DATAL   \ low data byte
VARIABLE DATAH   \ hi data byte
-->
```

Screen # 4

```
( RTTY - 8250 Address selection               pac 15:32 09/11/86 )
HEX
: SELECT_ADDR  \ select 8250 port addresses
  CLS
  DOWNPAGE OF SPACES
  ." WHICH SERIAL PORT ARE YOU GOING TO USE?" CR CR OF SPACES
  ." <Press A for COM1>" CR OF SPACES
  ." <Press B for COM2>" CR OF SPACES
  ." <Press C for COM3>" CR OF SPACES
  ." <Press D for COM4>" KEY CASE
  41 OF 3FB LCR ! 3F8 DATAL ! 3F9 DATAH ! 3FD LSR ! ENDOF
  42 OF 2FB LCR ! 2F8 DATAL ! 2F9 DATAH ! 2FD LSR ! ENDOF
  43 OF 3EB LCR ! 3E8 DATAL ! 3E9 DATAH ! 3ED LSR ! ENDOF
  44 OF 2EB LCR ! 2E8 DATAL ! 2E9 DATAH ! 2ED LSR ! ENDOF
   ENDCASE ;
DECIMAL -->
```

Screen # 5

```
( RTTY - Ascii or Baudot selection            pac 15:32 09/11/86 )
HEX
: ASCII/BAUDOT?  \ select one or the other
   CLS DOWNPAGE OF SPACES
   ." DO YOU WANT ASCII OR BAUDOT TRANSMISSION?" CR CR
   11 SPACES
   ." <Press A for ASCII or B for BAUDOT>"
    KEY
      CASE
 ( ascii)    41 OF  0 B/A ! ENDOF
 ( baudot)   42 OF -1 B/A ! 04 LCR @ PC! ENDOF
 ( automatic word selection of 5 bits, 1.5 stop bits, 0 parity)
      ENDCASE ;
DECIMAL
-->
```

Screen # 6

```
( RTTY - Baud rate Case - BAUDCASE            pac 14:23 09/11/86 )
HEX
: BAUDCASE \ case to store the baudrate selected
   CASE
  41 OF 09 D2 ENDOF  42 OF 09 00 ENDOF  43 OF 06 B8 ENDOF
  44 OF 06 00 ENDOF  45 OF 04 76 ENDOF  46 OF 04 17 ENDOF
  47 OF 03 59 ENDOF  48 OF 03 00 ENDOF  49 OF 01 80 ENDOF
  4A OF 00 C0 ENDOF  4B OF 00 60 ENDOF  4C OF 00 40 ENDOF
  4D OF 00 3A ENDOF  4E OF 00 30 ENDOF  4F OF 00 20 ENDOF
  50 OF 00 18 ENDOF  51 OF 00 10 ENDOF  52 OF 00 0C ENDOF
  53 OF 00 06 ENDOF  54 OF 00 03 ENDOF  55 OF 00 02 ENDOF
   ENDCASE ; \ see 8250 data book for these values and
             \ note that you can program any intermediate
             \ value between 0 and 56K baud
DECIMAL
-->
```

Screen # 7

```
( RTTY - Baud rate selection - BAUDREQUEST pac 11:53 09/12/86 )
VARIABLE KEYPRESS 66 KEYPRESS !
5 INDARR RATE1 \ establish 1st part of baud schedule
<< 50 66 75 100 110  0 RATE1 >>
12 INDARR RATE2
<< 150 300 600 1200 1800 2000 2400 3600 4800 7200 9600
   19200  0 RATE2 >> \ this is second part of baud schedule
: BAUDREQUEST \ select the baud rate desired
    CLS ." Select the Baud Rate you wish:" CR CR
 ." Available baud rates are:" CR ."     Rate  Press Letter" CR
 3 SPACES ." 45.45" 6 SPACES ." A" CR
 5 0 DO I RATE1 @ 5 .R ." .00" 6 SPACES KEYPRESS @ EMIT
       1 KEYPRESS +! CR LOOP 72 KEYPRESS !
 2 SPACES ." 134.50" 6 SPACES ." G" CR
 12 0 DO I RATE2 @ 5 .R ." .00" 6 SPACES KEYPRESS @ EMIT
       1 KEYPRESS +! CR LOOP -->
```

Screen # 8

```
( RTTY - BAUDREQUEST, cont'd.            pac 11:54 09/12/86 )
  ." 38400.00"  6 SPACES ." T" CR ." 56000.00" 6 SPACES
  ." U" CR
 ." <Press key shown to right of baud rate>"  66 KEYPRESS !
 KEY BAUDCASE ;
-->
```

Screen # 9

```
( RTTY - Bit size, stop bits, and parity   pac 16:06 09/11/86 )
HEX
: WORDCASE \ do action of WORDREQUEST
   CASE    \ choose bit size, stop bits, and parity
  41 OF 02 ENDOF
  42 OF 0A ENDOF
  43 OF 1A ENDOF
  44 OF 06 ENDOF
  45 OF 0E ENDOF
  46 OF 1E ENDOF
  47 OF 03 ENDOF
  48 OF 0B ENDOF
  49 OF 1B ENDOF
  4A OF 07 ENDOF
  4B OF 0F ENDOF
  4C OF 1F ENDOF  ENDCASE ;  DECIMAL -->
```

Screen # 10

```
( RTTY - WORDREQUEST  bits, stops, parity  pac 16:29 09/11/86 )
: WORDREQUEST CLS CR 6 SPACES ." Choose bit length, stop "
  ." bits, and parity:" CR 39 SPACES ." Press Letter" CR
 2 SPACES ." 7 bits, 1 stop bit,  no parity           A" CR
 2 SPACES ." 7 bits, 1 stop bit,  odd parity          B" CR
 2 SPACES ." 7 bits, 1 stop bit,  even parity         C" CR
 2 SPACES ." 7 bits, 2 stop bits, no parity           D" CR
 2 SPACES ." 7 bits, 2 stop bits, odd parity          E" CR
 2 SPACES ." 7 bits, 2 stop bits, even parity         F" CR
 2 SPACES ." 8 bits, 1 stop bit,  no parity           G" CR
 2 SPACES ." 8 bits, 1 stop bit,  odd parity          H" CR
 2 SPACES ." 8 bits, 1 stop bit,  even parity         I" CR
 2 SPACES ." 8 bits, 2 stop bits, no parity           J" CR
 2 SPACES ." 8 bits, 2 stop bits, odd parity          K" CR
 2 SPACES ." 8 bits, 2 stop bits, even parity         L" CR
 ." <Press key to the right of description>" KEY WORDCASE ; -->
```

Screen # 11

```
( RTTY - INITCOM, initialize serial port   pac 12:00 09/12/86 )
HEX
: INITCOM \ initialize the serial port selected
   SELECT_ADDR
   80 LCR @ PC! BAUDREQUEST DATAL @ PC! DATAH @ PC!
   ASCII/BAUDOT?
   B/A @ 0= IF WORDREQUEST LCR @ PC! THEN CR CR CR
   15 SPACES
    ." YOUR PORT IS INITIALIZED..." CR CR ;
DECIMAL
-->
```

(Screens continued on page 37.)

is any number other than zero, the character is sent out.

Screen two contains some machine language words which create an indexed array used for setting up the baud rate display tables. INDARR is also useful for making look-up tables, and is used for the Baudot conversion tables in the RTTY program. Screen seven shows how tables are set up. This program is written in Laboratory Microsystems' PC/Forth 3.1, a version of Forth-83. Those of you still using an older version from LMI (or a Forth-79 implementation) must use a 1 rather than a -1 for a truth flag. Please note that your COM3 and COM4 port addresses may differ from mine; if so, just insert the correct addresses where needed in the code.

	DATAL	DATAH	LCR	LSR
COM1	3F8	3F9	3FB	3FD
COM2	2F8	2F9	2FB	2FD
COM3	3E8	3E9	3EB	3ED
COM4	2E8	2E9	2EB	2ED

Table Two. Relationship of serial ports and addresses.

SIG FORTH '89 CALL FOR PAPERS

acm

for the first annual

FORTH APPLICATIONS SYMPOSIUM on REAL TIME SOFTWARE ENGINEERING

FOUR SEASONS HOTEL • Austin, TX • Feb. 17-19, 1989

The objective of this symposium is to share, discuss and disseminate recent research on and production of real time (software and hardware) computer applications. Attendees will hear presentations from industry experts on many topics, including:

Interrupt Driven Systems
Programming Environments
Multitasking / Multiuser Systems
Parallel Processing
Fault Tolerant Systems
Forth Engines and Software
Specialized Architectures
Microcontroller Applications
Industrial Systems
Computer Networks
Biomedical Engineering
Robotics & Machine Intelligence

Papers for oral and poster presentations are requested from computer professionals and other interested parties. Facilities will be available for scientific and technical demonstrations. Pre-publication proceedings will be made available to the participants at the symposium. Vendors of software and/or hardware may request exhibit space. Authors should submit an abstract of 250 words or less, typed, double spaced by the deadline below. Contributed papers should be previously unpublished work. You are not required to present a paper to attend the symposium.

Please send abstracts, and requests for symposium information to:

Dr. Paul Frenger or Mr. Rick Hoselton
Technology Information Center, Inc.
2900 Wilcrest Drive #400
Houston, TX 77042
Telephone: (713) 952-1060

TIMETABLE:

Receipt of Abstract	Jan. 1, 1989
Notification of Acceptance	Jan. 15, 1989
Receipt of Final Manuscript	Feb. 10, 1989

Sponsored by the ACM Special Interest Group on Forth

For ACM SIGForth membership information, refer to the above address.

Part Three

DESIGNING DATA STRUCTURES

MIKE ELOLA - SAN JOSE, CALIFORNIA

Host Abstraction

Two cascaded forms of abstraction have been suggested to make data objects more portable[1]. One of these is data abstraction. The other is abstraction of the host computer. Our chief concern will be with abstraction of the host computer in the interest of program portability. Additionally, much attention will be given to the declaration syntax for portable arrays.

Forth already hides peculiarities of the host computer behind its own data stack, return stack, etc. For data structures, however, we often have no other choice but to write code that depends upon host peculiarities, such as bit-processing widths. But not any more.

By avoiding direct references to host-specific quantities, we can write code that can be transported to other hosts without change. To hide more details about the host computer from Forth data objects, the kernel of every host should include certain words. As illustrated in Figure 3-1, only about five new words are needed. They help translate our intended actions into appropriate actions for particular hosts.

How to Hide the Host

One of the host peculiarities we need to hide is its bit-processing width. Another is the number of addresses spanned by a cell and a double.

While we may know the size of a datum we wish to skip over during an address operation, we don't know how many addresses a unit of data will span on an arbitrary host. We can't even say how many bytes may be allocated to a given unit, such as a cell, although we do know that the minimum number must be two bytes.

Two separate mappings are needed to hide these host characteristics. To hide the number of bytes per cell, a constant can be used. To hide the number of addresses spanned by any number of bytes, a mapping function is needed.

The routine that provides a general mapping function is `BYTES>ADR` ("bytes-to-addresses"). Once the correct number of bytes is known, `BYTES>ADR` finds the corresponding number of addresses. To discover how many addresses are spanned by a byte on any host, type:

```
1  BYTES>ADR
```

> ***"Forth already hides peculiarities of the host computer."***

The resulting value is two for a nibble-addressing processor; a more common result would be one.

Note that there is often a non-linear relationship between the output and the input of `BYTES>ADR`. A series of inputs such as 1, 2, 3, 4, 5 may produce 1, 1, 1, 1, 2 as output. Only when the host is a byte-addressing processor is there a linear relationship between the output and input of `BYTES>ADR`.

Also note that the definition of `BYTES>ADR` given in Figure 3-1 does not take into account alignment requirements of a host system. That task is left to individual readers to perform as necessary[2]. Specific mappings can be performed by constants, such as:

```
BYTES/CELL     ( -- #bytes )
```

To avoid having to write `BYTES/CELL BYTES>ADR`, both types of mappings can be consolidated as the host-dependent constant `ADR/,` ("addresses-per-cell-compile").

Another convenient constant is `ADR/C,` ("addresses-per-character-compile"), which replaces `1 BYTES>ADR`. Similarly, the constant `ADR/D,` ("address-per-double-compile") can help hide details about the implementation of doubles on a particular host. Together, these names give rise to the following simple glossary:

```
BYTES>ADR    ( #bytes -- #addresses )
BYTES/CELL   ( -- #bytes )
ADR/,        ( -- #addresses )
ADR/C,       ( -- #addresses )
ADR/D,       ( -- #addresses )
```

An additional constant has been included to help with the suite's customization for a particular host. This constant is `BITS/ADR` ("bits-per-address"). All the other constants and definitions in the suite rely, directly or indirectly, upon this value and the value of `BYTES/CELL`. These two constants alone should adapt the suite to a new host and, by extension, any applications that engage these routines faithfully. (Don't forget that host alignment requirements may need to be taken into account as well.)

Usage

Besides the examples shown here, Figure 3-2 offers a portable implementation for arrays, along with many extra features.

To skip over the count byte in a counted string, either use

```
[ 1  BYTES>ADR ]
LITERAL +
```

or, you can use

```
ADR/C, +
```

To create a compound data object, consisting of a double followed by a normal variable value, the following code can be used:

```
VARIABLE   *SINGLE
VARIABLE   *DOUBLE
: DOUBLE&SINGLE
  CREATE 0 , 0 , 0 ,
  DOES>
   DUP   *DOUBLE !
    [  ADR/,   2   *   ]
    LITERAL   +
    *SINGLE   !   ;
```

This example reveals the author's desire to abstract the data object. Through the use of `*SINGLE` and `*DOUBLE`, other operations should not have to "know" about the physical layout of the compound object.

To create a declarator for a Forth jump table, try:

```
: CASES-OF  (  #cases -- )
  CREATE   0
    DO   FIND   , LOOP
  DOES>   (  idx <pfa> -- )
    SWAP   ADR/,   *   +
    @   EXECUTE   ;

3 CASES-OF TH-QUITTER
ABORT   QUIT   BYE
```

Notice that `CASES-OF` uses `FIND` to leave a cell on the stack which is later compiled by `,` (cell-compile). At run-time, these code fields are restored to the stack by `@` (cell-fetch). Although they are being manipulated as if they were cells, values fetched and stored this way must still be executable addresses.

Array Design Considerations

To correctly index an array, the addresses spanned by each of the elements in the array must be known. To produce arrays of cells, doubles, or bytes using a single array declarator, the width of the elements should be recorded in the instance object itself. By recording this value in bytes and using `BYTES>ADR` when indexing the array, an array declarator is produced that can be transported to many different hosts.

Consider the following definition of `TABLE`. Notice that `BYTES>ADR` is not part of the indexing algorithm. Instead, this mapping takes place only once, when the table is declared.

```
: TABLE
( #elements #bytes/element -- )
  CREATE
    BYTES>ADR   DUP   ,
    ALLOT
  DOES>
    DUP @
     ( idx <pfa> #adr/element -- )
    ROT    *   (  <pfa> offset -- )
    +   ;

50 1 TABLE TH-CHAR
```

`TABLE` is not limited to producing arrays of cells, doubles, and bytes. To declare an array of ten-character strings,

```
10 10 TABLE TH-10CSTRING
```

can be used. But the prefix parameters do not tell much of the story. Likewise,

```
2   2   TABLE   2ITEMS
```

conveys that the table `2ITEMS` has two elements of two bytes each. But we don't have enough clues to be certain what these elements are. The array `2ITEMS` could store addresses, two-byte strings, or variable values.

A different syntax can communicate more about the array, leading to self-documenting code:

- Bytes, or byte-based units, should be explicitly stated as the unit of measurement for each element.
- When a number has no corresponding units, such as a dimension in an array, a place-holder such as `BY` should help clarify its meaning.
- An identifier such as `ELEMENT` also helps make the declaration clearer.

With these enhancements, the preceding array can be declared as:

```
2 BY 2 BYTE ELEMENT
ARRAY   XY
```

Because the new syntax provides more clues about the contents of the array, arrays so defined are less subject to misuse.

`BY` can also help declare n-dimensional matrices by counting the number of dimensions as they are specified. (Although I have not shown the definition of an n-dimensional matrix declarator in Figure 3-2, it is not difficult to conceive one.)

By defining the unit identifiers `CELL` and `BYTE` in a special fashion, their use can be required when declaring arrays. One of these unit identifiers must precede `ELEMENT`.

`ELEMENT` must appear after a unit identifier and before `ARRAY`. This syntax is enforced through `UNITS-CK`. The presence of the dimension identifier `BY` is enforced through `#BYS-CK`.

Array Declaration Style

Although an enforced array syntax is a major step towards clearer array declarations, there are other ways to promote clarity. A stylistically correct declaration also clearly describes the array. For example, there are five valid index values for the following array of doubles (six, actually, because zero can also be used as an index value in this implementation):

```
5 BY 1 DOUBLE ELEMENT
ARRAY   5DOUBLES
```

The preferred style is to declare the element as a single unit long (with a couple of exceptions). When it is not a single unit, the intent is obscured. What kind of elements comprise the following array?

```
5 BY 2 CELL ELEMENT
ARRAY   5x2ARRAY
```

Because the length of each element is two cells, the type of the data elements is unclear. Furthermore, the second cell of each pair of cells cannot be addressed using any of the possible index values. Extra address arithmetic is required to address the second cell in each pair. Nevertheless, this is an appropriate declaration for an array of two-cell elements to be manipulated with `2@` and `2!` (eliminating the need for extra address arithmetic). This is another exception to the guideline that an element should always be a single unit long.

Although `BYTE` can be used to declare an array of doubles, it is a poor practice:

```
5 BY 4 BYTE ELEMENT
ARRAY   5ITEMS
```

`BYTE` is the preferred units identifier only when declaring an array of byte val-

Figure 3-1. Basic suite.

```
8  CONSTANT  BITS/ADR
2  CONSTANT  BYTES/CELL
( Correct values shown for my system;
your system may require other values)

: BYTES>ADR   ( #bytes -- #addresses-spanned )
  DUP  8  *  BITS/ADR  MOD   >R
  8  BITS/ADR  */          (  #addresses -- )
  R>  IF  1+  THEN  ;

BYTES/CELL   BYTES>ADR  CONSTANT  ADR/,
  1  BYTES>ADR  CONSTANT  ADR/C,
```

Figure 3-2. Extensions for array support.

```
VARIABLE  BASIC-LEN  ADR/,  ALLOT
0 0  BASIC-LEN  2!
: BYTE-SCALER                ( <name>  -- )
  CREATE              (  bytes/unit  -- )
          ,
  DOES>          ( #elements  <pfa> -- )
    @  *  DUP      ( #bytes  #bytes  -- )
      BASIC-LEN  !  ;

BYTES/CELL   BYTE-SCALER  CELL
       1  BYTE-SCALER  BYTE

: ELEMENT   ( #bytes  -- #addresses-spanned )
  DUP  BASIC-LEN  ADR/,  +  !
  BYTES>ADR  ;

: UNIT-CK  ( -- )
  BASIC-LEN  2@  -
    IF  CR
      ." Missing ELEMENT or unit identifier"
      ABORT
    THEN ;

: UNIT-CLEAR ( -- )
  0  BASIC-LEN  !
  -255  BASIC-LEN
  ADR/,  +  !   ;
```

(This unit-checking scheme employs the raw number of bytes as calculated by the byte-scalers, such as CELL and BYTE. This value is left on the stack for processing by ELEMENT. Since both the byte-scaler and ELEMENT write the same value into one of the cells of BASIC-LEN, UNIT-CK only has to confirm that each of those cells is equal.)

```
VARIABLE  #BYS   0  #BYS   !
: #BYS-CK   ( -- )
  #BYS  @    0  #BYS  !    ( #bys -- )
    1-  IF  UNIT-CLEAR  CR
      ."  Missing or extra 'BY'"
      ABORT
    THEN ;

: BY
  1  #BYS  +!  ;

: BOUNDS-CK   ( th-element  *max  -- )
  @  >  IF
    CR  ."  Maximum index exceeded"
    SWAP  .S  ABORT
  THEN ;

: ARRAY  ( #elements   scaled-element-length  -- )
  #BYS-CK  UNITS-CK  UNITS-CLEAR
  CREATE
    OVER  ( max -- )  ,
   DUP  ,  SWAP  1+  *  ALLOT   ( 0/1-based indexing)
  DOES>  ( idx  <pfa>  -- idxth-element )
    2DUP  BOUNDS-CK
    ADR/,  +  ( idx  *size -- )
    DUP  ADR/,  +    ( idx *size  *1st-element  -- )
    SWAP  @    ( idx 1st-element element-length -- )
    ROT  *  +  ;
```

ues, or an array of strings, as in the following declarations:

```
75 BY 1 BYTE ELEMENT
ARRAY  TH-CVALUE

5 BY 15 BYTE ELEMENT
ARRAY  TH-15CSTRING
```

Even though their memory requirements are the same, one has 75 valid index values and the other has only 5 valid index values (not counting zero as a possible index value). The value preceding BYTE helps clarify which is an array of byte values, and which is an array of strings.

Conclusion

Use of host abstraction eases the porting problems of Forth with respect to data structures. To best realize this abstraction, we need to habitually engage the new routines, and we need to follow the syntactical rules and style guidelines presented for the declaration of arrays. Besides increased program portability, these practices provide increased program readability.

As a parting observation, consider how we specify memory allocations with ALLOT. The units identifiers that have been suggested can bring similar benefits in this context. By not specifying an exact number of addresses, but instead specifying a number of "abstract" units, host-tailored memory allocations can be made appropriately. In this way, 4 CELL ALLOT (or ALLOTMENT) would automatically allocate 128 bytes on a host computer with 32-bit memory words.

References

1. Elola, Mike. "Designing Data Structures," *Forth Dimensions*, Vol. X, Issue 2.
2. Tracy, Martin. "A Forth Standard Prelude," *Software Tools*, October 1987.

THE BEST OF GENIE

GARY SMITH - LITTLE ROCK, ARKANSAS

I am going to take a completely different tact in this issue, and discuss what can only be described as an experiment that may yet end in failure. Our hope is that it will succeed beyond our dreams and result in eventual connection with any Forth resource in the world. The dream is a virtual Forth network called ForthNet.

I had the idea before we got set up on GEnie, and with the help of Jack Woehr (JAX), sysop on the WELL and regular on GEnie, we began our noble experiment to establish a virtual ForthNet by connecting GEnie and the WELL. We have since added the *x* Coast Forth Boards, and hope to integrate much more of the Forth community in the near future.

The biggest problem now is connectivity. ForthNet exists only because I am willing to serve as a mule, porting messages from each point in the loop. This, obviously, is not the long-term way to succeed. We must begin letting the computers do the porting for us. GEnie does not currently provide any gateways, so we are immediately confronted with a stumbling block. The solution may rest in the establishment of another link in Denver of the *x* Coast Forth Boards, which will bring them and the WELL into a tighter loop to which I can link via PCPursuit, limiting the problem of porting to one junction.

The following will give you a look at the problems and promises. I invite your participation and suggestions.

Topic 1
Fri Aug 19, 1988 GARY-S [Gary]
Sub: From the WELL:

Messages posted here are ported from a public-domain area of the comp.lang.forth on the WELL. Replies in this topic will be ported back to the WELL and the xCFBs. 7 message(s) total

Category 14, Topic 1, Message 1
Topic 39: ForthNet Gateway : If you enter a message here it is public domain.
Read #1 # 7: Ridu dum kiam vi povas, simiulo-knabo! (bandy)
Mon, Aug 15, '88

Okay, I have a question for folks at large:

I have two applications written in MacForth+, which tends to fall apart and make giant-sized applications. Let's not even mention that it blows up regularly on the Mac II and that the Sibley Editor has two serious bugs in it.

So, am I stuck with this or is there a (semi-?) compatible language system that I can run on the Mac+/SE/II? I have about 160K of source code to work with, and I wouldn't mind putting in a man-week making conversions to how it handled windows and events.

Read #1 # 8: Jack J. Woehr (jax)
Thu, Aug 18, '88

The mailman cometh; gars (GARY-S) is on his way to waft your question to the all-knowing MacForth gurus of GEnie! :-) In the meantime, if the darn editor doesn't work, why don't you fix it? Why don't we fix it here, all of us?

Read #1 # 9: Ridu dum kiam vi povas, simiulo-knabo! (bandy)
Fri, Aug 19, '88

If (a) I could figure out from the silly docs how to rebuild the system, and (b) I could find the part of the code for the editor that is passing NIL instead of "" for the names of the scroll-bars....

Sorry for the delay in picking up the mail. I have been very busy. I did post notice on the x Coast Forth Boards, and a Category will be set aside on GEnie. Bandy, you have the honor of breaking the ice on what could develop into a virtual Network — and that's exactly what Jax and I are hoping for. Look for a response (hopefully quicker than the pick-up) in this same area, with messages for WELL digestion from GEnie and the xCFBs. 7: Gary

Category 14, Topic 1, Message 2
Sat Aug 20, 1988 S.W.SQUIRES [scott]
Response to MacForth+ questions:

What version is he running? Can he be more specific about the problems? Since I haven't experienced 'two serious bugs' in the editor I'm not sure what he's referring to. Some older versions had minor problems, but to my knowledge those have been solved. The source code is included, so he can review and change it directly, if needed.

MacForth has been used to create applications running on 128K Macs, so the size issue can certainly be dealt with. The vocabularies and extensions can be trimmed down (recommended in the manual). Memory space that is allocated should use heap space instead of object space, when possible.

If he really has a problem, I suggest he contact Creative Solutions and get his questions answered directly. The only

Forth-83

USING A STRING STACK

RON BRAITHWAITE - LOS ANGELES, CALIFORNIA

Screens continued from preceding issue.

```
( D>$           d -- $                                          )
( Converts the double precision integer d to the string $ on    )
( the string stack.                                             )

: D>$           ( d -- $ )
      DUP >R  <# #S R> SIGN #>  $CNT@  ;


( $>D           $ -- d n                                        )
( Converts the string $ on the string stack to the double       )
( precision integer d, using the current radix, and the         )
( conversion count n. If all characters in the string $ are     )
( converted, the flag is -1. If the string $ is partially       )
( converted, n is the number of characters that converted.      )
( If n is 0, the value of d is undefined.                       )
(                                                               )
( The position of the decimal point is placed in the variable   )
( DPL. If no decimal point was present, DPL will contain the    )
( value -1. If either hardware or software floating point       )
( extensions have been loaded, the action of $>D and the value )
( in DPL may vary from this description.                        )

: $>D           ( $ -- d n )
        BASE @ >R  -1 DPL !  $P@ COUNT  0 TUCK   ( Set up scan  )
 ?DO    OVER C@ DUP ASCII 0 < SWAP ASCII 9 > OR ( 0 > c > F?   )
  IF    LEAVE                                    ( Get out      )
  THEN  1+ SWAP 1+ SWAP                          ( Inc cnt&addr )
 LOOP   NIP  0 $CNT  OVER =                      ( Pure number? )
 IF     DROP  -1                                 ( Don't adjust )
 ELSE   DUP  $LEFT                               ( Extract num  )
 THEN   $P@  NUMBER?                             ( Convert      )
 IF     ROT                                      ( Offset       )
 ELSE   ROT  DROP  0                             ( Didn't go    )
 THEN   $DROP  R> BASE !  ;                      ( Restore base )


( $INDEX        1$ 0$ -- o                                      )
( Returns the offset into the second string, 1$, on the string )
( stack of the first position matching the pattern of the first)
( string 0$. If 0$ is not a subset of 1$, -1 is returned.       )
(                                                               )
( $INDEX is equivalent to the LMI word STRNDX                   )
```

(Continued from previous page.)

other real Forth for the Mac is MACH2 from Palo Alto Shipping. It has the advantage of being closer to Forth-83 and somewhat faster. The disadvantage is not having much in the way of true extensions (i.e., doing it the same as all the Mac languages — from the ground up). This may have changed, since I haven't seen a recent version for a year or so. Both Forths are good, it just depends on the particular user's needs. He should make sure the problem really is with the system, and not with his program or process of using it.
—Scott

Category 14, Topic 1, Message 3
Tue Aug 23, 1988 GARY-S [Gary]
> PORTED FROM THE WELL ==>
Topic 39: ForthNet Gateway : If you enter a message here it is public domain.
Read #1 # 14: Ridu dum kiam vi povas, simiulo-knabo!(bandy)
Mon, Aug 22, '88

The two bugs in the editor (they refuse to belive it) are that when it is making the controls for the horizontal and vertical scroll bars, it passes NIL as a StringPtr (for the control name) rather than "" (a pointer to an empty string). Macs no longer have a 0 at 0.

My current beef is that both my applications blow up on the Mac II. Setting TMON to Strict discipline reveals that, by the time it calls the SlotManager from the GINIT routine, the heap is quite trashed. This doesn't happen with the smaller TURNKEY applications, such as the Engine Demo, but it's happening in my program before any of my code gets executed!

So does MACH2 basically have no support for windows (á la Lightspeed-everything), menus, etc.? There aren't any weird little differences with anything fundamental like ROLL?

Read #1 # 15: Ridu dum kiam vi povas, simiulo-knabo! (bandy)
Mon, Aug 22, '88

I did the obvious thing and looked up Palo Alto Shipping in the Palo Alto phone book, and no number... Number please?

Category 14, Topic 1, Message 4
Wed Aug 24, 1988 D.RUFFER [Dennis]

Well, the Palo Alto Shipping company

```
: $INDEX        ( 1$ 0$ -- o )
       1$ COUNT  0 TUCK                        ( Set up loop  )
 ?DO   DROP  0 $CNT  1 $CNT  I -  >            ( Run out?     )
  IF   -1  LEAVE                               ( Not subset   )
  THEN  DUP C@  $P@ 1+ C@  =                   ( First char=? )
  IF    DUP  -1  $P@ COUNT OVER + SWAP         ( Flag, indices)
   ?DO  DROP  DUP C@  I C@  <>                 ( Not equal?   )
    IF  0  LEAVE                               ( Get out      )
    THEN 1+  -1                                ( Inc ptr, flag)
   LOOP NIP                                    ( Drop addr    )
   IF   1$  - DUP  1-  LEAVE                   ( Offset       )
   THEN                                        (              )
  THEN  1+  0                                  ( Try next char)
 LOOP   NIP  $2DROP  ;                         ( Leave offset )


( $VERIFY       1$ 0$ -- o                                    )
( Returns the offset into the second string 1$ on the string  )
( stack of the first character in the first string 0$ which is )
( not found in the second string <i.e., the length of the     )
( initial substring of 0$ which consists entirely of characters)
( in 1$>.                                                     )
(                                                             )
( $VERIFY is equivalent to the LMI word STRSPN                )

: $VERIFY       ( 1$ 0$ -- o )
       0 $CNT  DUP 0                           ( 0$ loop      )
 ?DO    0  1 $CNT  0                           ( 1$ loop      )
  ?DO   1$ 1+ I + C@  $P@ 1+ J + C@  =         ( Equal?       )
   IF   DROP -1  LEAVE                THEN     ( Found it     )
  LOOP  0=                                     ( Get out?     )
  IF    DROP  I  LEAVE                THEN     ( <> at I pos  )
 LOOP   $2DROP  ;                              (              )


( $PARSE        1$ 0$ -- 3$ 2$                                )
( Parses the string 1$ for the string 0$, returning the parsed )
( string 2$, without the string 0$, and the remaining string  )
( 3$, without the string 0$. If no instances of the string 0$  )
( are found, string 2$ is the null string and string 3$ is 0$. )

: $PARSE        ( 1$ 0$ -- 3$ 2$ )
       $2DUP  $INDEX  DUP  -1 <>               ( Find pos     )
 IF    1-  0 $CNT  $DROP  $DUP  OVER +         ( Offset to 3$ )
       0 $CNT  SWAP -  $RIGHT  $SWAP  $LEFT    ( Make 3$ & 2$ )
 ELSE  DROP  $DROP  $NULL                      ( Not found    )
 THEN  ;


( $SOUNDEX      0$ -- 1$                                      )
( Computes the soundex code string 1$ of the string 0$ on the  )
( string stack. The soundex code is in the range 0 => s => 9999)

: C>SNDX        ( c1 -- c2 )
       64 -  DUP  0<  OVER  27 <  OR           ( In range?    )
 IF    " 01230120022455012623010202"  + C@     ( Get code     )
       ( ABCDEFGHCJKLMNOPQRSTUVWXYZ)           ( Corresponding)
 ELSE  DROP  ASCII 0                           ( Not char     )
 THEN  ;
```

is accessible right here on GEnie. Type MACH2 to get to their RoundTable. And for those who are not here yet [on GEnie], their address is:

Palo Alto Shipping Company
P.O. Box 7430
Menlo Park, CA 94026
1(800)44FORTH
GEnie address: PASC

Category 14, Topic 1, Message 5
Mon Sep 05, 1988 D.MILEY

Dennis (D.RUFFER), thanks for posting the information about the MACH2 RT. I'd like to add that the 1(800)44FORTH number is for orders only; our product support number is (415)363-1399. Also, our GEnie address is D.MILEY (not PASC).

My first programming language on the Macintosh was MacForth (back in the 128K Mac days). However, about two years ago I began using MACH2 almost exclusively, and in November of 1987 I started working for Palo Alto Shipping (the parent company of MACH2). Given my experience, I don't mind saying that CSI's MacForth is a fine Forth-language-based development system (but, of course, I prefer MACH2).

My opinion is that there would be a significant amount of work converting between MacForth and MACH2. The Forth-language-based differences aren't too severe, MACH2 conforms almost completely to the Forth-83 Standard while MacForth is somewhere between Forth-79 and Forth-83. The big differences come in the interface to the Macintosh toolbox. MacForth supplies its own high-level interface to much of the Mac toolbox, while MACH2 uses a CALL "hook" to reference each toolbox/trap directly (almost — we still use "glue" to size each parameter and move values to/from the system stack or processor registers). As an example, to draw an oval in the current grafPort, you rely on the Mac ROM routine FrameOval. The Pascal definition of FrameOval (as found in *Inside Macintosh*) is:

```
PROCEDURE   FrameOval   (r:
Rect);
```

From MacForth you might draw an oval by saying:

```
50 100 100 200 FRAME OVAL
```

where 50, 100, 100, and 200 specify the the pixel coordinates of the top-left and bottom-right corners of the bounding rectangle.

To draw an oval in MACH2 you might say:

```
MyRect CALL FrameOval
```

where `MyRect` must return a pointer to a rectangle record (eight bytes). Note that the MACH2 example closely parallels the Pascal interface. Of course, in the MACH2 case (as in Pascal), you have to initialize the rectangle record before you use it.

This could be done as follows:

```
VARIABLE MyRect 4 VALLOT
```

(declare an 8-byte, global-variable record)

```
MyRect 50 100 100 200 CALL
SetRect
```

(initialize the rectangle record)

`VALLOT` is a "cousin" to `ALLOT`. `VALLOT` reserves bytes in the Macintosh global-variable space (*not* in the object-code space as `ALLOT` does).

`SetRect` is another Macintosh ROM routine. Its Pascal definition is:

```
PROCEDURE SetRect
(VAR r: Rect;
left,top,right,bottom:
INTEGER);
```

You should note the following from the above example: When interfacing to the Macintosh ROM, MacForth tries to reduce the amount of "work" required of the programmer, while MACH2 tries to conform directly to *Inside Macintosh* (Apple's technical reference to the Macintosh computer family). I personally find MACH2's CALL interface to be much more powerful and flexible, but others tend to appreciate MacForth's simplified interface to the Mac's toolbox. However, in some situations MacForth's toolbox approach can cause significant difficulty (complexity). If MacForth doesn't supply a high-level equivalent to a particular ROM routine, you may face some pretty ugly stack manipulations in order to interface directly to the ROM (or you may have to resort to assembly language). This shouldn't happen in MACH2 because nearly all of the ROM routines are supported by the same CALL interface (nearly 900 toolbox routines are supported by `CALL`).

Both MacForth and MACH2 offer a pre-written event loop. That means events are handled more or less automatically by

```
: $SOUNDEX      ( 0$ -- 1$ )
       $UPPER  1 HERE C!  $P@ COUNT  0>          ( Not null?    )
 IF     C@                                        ( Get char     )
 ELSE   DROP  ASCII 0                             (              )
 THEN   HERE 1+ C!  0 $CNT  1 >                   ( Store 1st chr)
 IF     $P@  1+  C@  C>SNDX                       ( Last char    )
        $P@  COUNT  OVER +  SWAP  1+              ( Rest of $    )
  ?DO   I C@  C>SNDX  TUCK  =                     ( Last =?      )
        OVER  ASCII 0  =  OR  0=                  ( Not =|0      )
   IF   DUP HERE COUNT + C! HERE DUP C@ 1+ SWAP C! THEN
  LOOP  DROP                                      ( Run thru 0$  )
 THEN   $DROP " 000" $@ HERE $@ $APPEND 4 $LEFT ;  ( 4char code)


( $MATCH        1$ 0$ -- flag                                       )
( Returns TRUE if the string 1$ on the string stack matches the)
( pattern of 0$. The pattern of 0$ may consist of the pattern  )
( codes of C, G, N, P, A, L, U, E, `, or ~. If the pattern code)
( is a ` or ~, the following character is taken as a literal   )
( value. The pattern is the union of the pattern codes in 0$.  )
(                                                               )
( The significance of the pattern codes are:                    )
(        C        33 Control characters, including DEL          )
(        G        128 Graphic characters above DEL              )
(        N        10 Numeric characters                         )
(        P        33 Punctuation characters, including SP       )
(        A        52 Alphabetic characters                      )
(        L        26 Lower-case alphabetic characters           )
(        U        26 Upper-case alphabetic characters           )
(        E        Everything non-graphic                        )
(        `        The following character is present            )
(        ~        The following character is not present        )


( Implementation note:  This is a very long which would        )
( normally be divided into much smaller words. In this case,   )
( however, further decomposition would make it more clumsy.    )


: $MATCH        ( 1$ 0$ -- flag )
        -1  $P@ COUNT  OVER +  SWAP               ( Flag, do 0$  )
 ?DO    I C@                                      ( Get pattern  )
  CASE  ASCII C                                   ( Control?     )
  OF    -1  1$ COUNT  OVER +  SWAP                (              )
   ?DO  I C@  DUP  32 <  SWAP  127 =  OR NOT      (              )
    IF  DROP   0  LEAVE                  THEN     (              )
   LOOP AND  DUP 0=                               (              )
   IF   LEAVE                            THEN     (              )
  ENDOF ASCII G                                   ( Graphic?     )
  OF    -1  1$ COUNT  OVER +  SWAP                (              )
   ?DO  I C@  128 <                               (              )
    IF  DROP   0  LEAVE                  THEN     (              )
   LOOP AND  DUP 0=                               (              )
   IF   LEAVE                            THEN     (              )
  ENDOF ASCII N                                   ( Numeric?     )
  OF    -1  1$ COUNT  OVER +  SWAP                (              )
```

```
  ?DO  I C@ DUP  ASCII 0 < SWAP ASCII 9 > OR      (               )
   IF  DROP   0  LEAVE                   THEN     (               )
  LOOP AND  DUP 0=                                (               )
   IF   LEAVE                            THEN     (               )
 ENDOF ASCII P                                    ( Punctuation?  )
 OF     -1  1$ COUNT  OVER +  SWAP                (               )
  ?DO  I C@  DUP     31 > OVER ASCII 0 < AND      (               )
       SWAP DUP ASCII 9 > OVER ASCII A < AND      (               )
       SWAP DUP ASCII Z > OVER ASCII a < AND      (               )
       SWAP DUP ASCII z > SWAP 127 < AND OR OR OR (               )
   IF  DROP   0  LEAVE                   THEN     (               )
  LOOP AND  DUP 0=                                (               )
   IF   LEAVE                            THEN     (               )
 ENDOF ASCII A                                    ( Alphabetic?   )
 OF     -1  1$ COUNT  OVER +  SWAP                (               )
  ?DO  I C@ DUP ASCII @ > OVER ASCII [ < AND      (               )
       SWAP DUP ASCII ` > SWAP ASCII { < AND OR NOT (             )
   IF  DROP   0  LEAVE                   THEN     (               )
  LOOP AND  DUP 0=                                (               )
   IF   LEAVE                            THEN     (               )
 ENDOF ASCII L                                    ( Lower case?   )
 OF     -1  1$ COUNT  OVER +  SWAP                (               )
  ?DO  I C@ DUP ASCII a < SWAP ASCII z > OR       (               )
   IF  DROP   0  LEAVE                   THEN     (               )
  LOOP AND  DUP 0=                                (               )
   IF   LEAVE                            THEN     (               )
 ENDOF ASCII U                                    ( Upper case?   )
 OF     -1  1$ COUNT  OVER +  SWAP                (               )
  ?DO  I C@ DUP ASCII A < SWAP ASCII Z > OR       (               )
   IF  DROP   0  LEAVE                   THEN     (               )
  LOOP AND  DUP 0=                                (               )
   IF   LEAVE                            THEN     (               )
 ENDOF ASCII E                                    ( Not graphic?  )
 OF     -1  1$ COUNT  OVER +  SWAP                (               )
  ?DO  I C@   127 >                               (               )
   IF  DROP   0  LEAVE                   THEN     (               )
  LOOP AND  DUP 0=                                (               )
   IF   LEAVE                            THEN     (               )
 ENDOF ASCII `                                    ( Literal?      )
 OF     0  1$ COUNT  OVER +  SWAP                 (               )
  ?DO  I C@   J 1+ C@   =                         (               )
   IF  DROP  -1  LEAVE                   THEN     (               )
  LOOP AND  DUP 0=                                (               )
   IF   LEAVE                            THEN     (               )
 ENDOF ASCII ~                                    ( Literal NOT?  )
 OF     -1  1$ COUNT  OVER +  SWAP                (               )
  ?DO  I C@   J 1+ C@   =                         (               )
   IF  DROP   0  LEAVE                   THEN     (               )
  LOOP AND  DUP 0=                                (               )
   IF   LEAVE                            THEN     (               )
 ENDOF                                            (               )
 ENDCASE                                          (               )
LOOP   $2DROP  ;                                  ( leave flag    )


( >$YYYYMMDD     y md -- $                                        )
( Converts the standard date format integers y md to a date      )
( string in the format yyyymmdd.                                  )
```

both products. Both products do multitasking, both allow interactive creation of windows, controls, menus, etc. Both provide assemblers, although the MACH2 assembler isn't RPN (MACH2 uses a "standard" Motorola-syntax assembler). MacForth supplies source code to their editor, assembler, and extensions (MACH2 does not, a disadvantage to some). Byte-for-byte, MacForth will usually produce more compact code (smaller size); however, MACH2 will run about two to three times as fast as MacForth (this difference in speed usually isn't meaningful unless you're doing heavy memory access, looping, or number crunching). MACH2 is subroutine threaded, MacForth is token threaded.

Well, that's my not-too-brief summary. MACH2 and MacForth do have significant differences, and I think both are good products. I was a bit surprised to see such a critical attack on MacForth. Frankly (from my experience), I don't think they deserve such treatment.
—Waymen

Category 14, Topic 1, Message 6
Mon Sep 05, 1988 D.RUFFER [Dennis]

Thanks, Waymen, for giving us the correct scoop on contacting Palo Alto, and for the excellent (although slanted) opinion of both Forths for the Mac. Glad to see someone from there is monitoring over here.

Now, maybe Ward will give us the "other" side of the story? <grin> DaR

Category 14, Topic 1, Message 7
Sat Sep 10, 1988 GARY-S [Gary]
> PORTED FROM THE WELL ==>
(Comment on new xCFB in Denver by Jax)
Topic 39: ForthNet Gateway : If you enter a message here it is public domain.
Read #1 # 23: Jack J. Woehr (jax) Thu, Sep 8, '88

Looks like the name of the new board will be the Realtime Control & Forth Board. It will be PCPursuitable, free, dedicated to discussions and files about embedded systems and Forth.
—Your Sysop, G. Who

Topic 2
Fri Aug 19, 1988 GARY-S [Gary]
Sub: From the xCFBs/ForthNet

This topic will be devoted to subjects raised and replies to ForthNet questions generated on the x Coast Forth Boards.

Category 14, Topic 2, Message 1
Fri Aug 19, 1988 GARY-S [Gary]
Date: 08-13-88
To: Gary Smith
From: Sysop
Subj: ForthNet

You must have a lot of free time on your hands! <grin>

All the Forth Conferences on the NCFB, ECFB, and the BCFB contain the same messages. This should save some calling between boards to get everything. As a matter of fact, if your messages were posted in one of the eight networked conferences, you wouldn't have to make another long distance call to post them on the other boards.

I'm also assuming that GEnie isn't going to claim any legal rights to any of the transplanted messages; otherwise, I'll have to insist that permission is received from the message's author.

Date: 08-16-88
To: Gary Smith
From: Sysop
Subj: ForthNet

Gary, have you used the ProDoor ARCM command? In just a couple of minutes, you can capture all the new messages into an ARCed file for your off-line use. An added benefit of this strategy is that the messages won't have any added garbage characters except those inserted by the original author when entering the message.

Don, we have made it clear from the beginning that GEnie Forth RT is an *open* public-domain forum. The messages in the WELL are *not* normally such. So I set up a topic on the WELL, devoted to this virtual ForthNet, that is clearly identified as being public domain. The first such messages were posted on GEnie Category 14 (ForthNet) tonight. Your messages will also be posted on GEnie tonight. Let me know how to upload ASCII text to NCFB and I ... will close the loop as long as I can — or can someone pick up Cat. 14 for posting here?
—Gary

Category 14, Topic 2, Message 2

```
: >$YYYYMMDD      ( y md -- $ )
        SWAP      0 <# # # # # #> HERE 1+  SWAP CMOVE
        256 /MOD 0 <# # # #>      HERE 5 + SWAP CMOVE
                 0 <# # # #>      HERE 7 + SWAP CMOVE
        HERE 8 OVER C!  $@  ;

( $YYYYMMDD>      $ -- y md                                     )
( Converts the date string in the format yyyymmdd to the        )
( standard date format integers y md.                           )

: $YYYYMMDD>      ( $ -- y md )
        $DUP  4 $LEFT  $P@ NUMBER? NIP 0=        ( yyyy?        )
 IF     DROP  0                                  ( 0 year       )
 THEN   $DROP $DUP  4 2 $MID $P@ NUMBER? NIP 0= ( mm?          )
 IF     DROP  0                                  ( 0 month      )
 THEN   256 * $DROP 2 $RIGHT $P@ NUMBER? NIP 0= ( dd?          )
 IF     DROP  0                                  ( 0 day        )
 THEN   $DROP  +  ;


( >$MM/DD/YY     y md -- $                                      )
( Converts the standard date format integers y md to a date     )
( string in the format mm/dd/yy.                                )

: >$MM/DD/YY      ( y md -- $ )
        256 /MOD 0 <# ASCII / HOLD # # #> HERE 1+  SWAP CMOVE
                 0 <# ASCII / HOLD # # #> HERE 4 + SWAP CMOVE
        1900 -   0 <# # # #>              HERE 7 + SWAP CMOVE
        HERE 8 OVER C!  $@  ;


( $MM/DD/YY>     $ -- y md                                      )
( Converts the date string in the format mm/dd/yy to the        )
( standard date integers y md.                                  )

: $MM/DD/YY>      ( $ -- y md )
        $DUP  2 $RIGHT  $P@ NUMBER? NIP 0=       ( yy?          )
 IF     -1900                                    ( 0 year       )
 THEN   1900 +                                   ( This century )
        $DROP $DUP 2 $LEFT  $P@ NUMBER? NIP 0=   ( mm?          )
 IF     0                                        ( 0 month      )
 THEN   256 *                                    ( Shift left 8b)
        $DROP 3 2 $MID  $P@ NUMBER? NIP 0=       ( dd?          )
 IF     0                                        ( 0 day        )
 THEN   $DROP  +  ;                              (              )
```

```
( >$JULIAN      y md -- $                                        )
( Converts the standard date format integers y md to the julian)
( day of the string $. The julian day is the day offset from   )
( the start of the current year. The julian date is the number )
( of days since the last conjunction of the 28 year solar cycle)
( and 19 year lunar cycle, calculated to be January 1, 4713 BC.)
( On December 31, 1986, the julian date was 2,446,796.         )

: >$JULIAN      ( y md -- $ )
       SWAP  4 MOD  0=                            ( Leap year?   )
 IF    1.      ELSE    0.                         ( Compensate?  )
 THEN  ROT   256 /MOD  >R  S>D  D+  R>            ( day month    )
 CASE     1 OF    0  ENDOF                        ( January   31 )
          2 OF   31  ENDOF                        ( February  28 )
          3 OF   59  ENDOF                        ( March     31 )
          4 OF   90  ENDOF                        ( April     30 )
          5 OF  120  ENDOF                        ( May       31 )
          6 OF  151  ENDOF                        ( June      30 )
          7 OF  181  ENDOF                        ( July      31 )
          8 OF  212  ENDOF                        ( August    31 )
          9 OF  243  ENDOF                        ( September 30 )
         10 OF  273  ENDOF                        ( October   31 )
         11 OF  304  ENDOF                        ( November  30 )
         12 OF  334  ENDOF  ABORT" Illegal month" ( December  31 )
 ENDCASE S>D  D+  D>$  ;                          ( y md>julian  )


( $JULIAN>      $ -- y md                                        )
( Converts the julian day of the string $ to the standard date )
( format integers y md. The julian day is the day offset from  )
( the start of the current year. The julian date is the number )
( of days since the last conjunction of the 28 year solar cycle)
( and 19 year lunar cycle, calculated to be January 1, 4713 BC.)
( On December 31, 1986, the julian date was 2,446,796.         )

: $JULIAN>      ( $ -- y md )
       @DATE DROP  DUP  4 MOD  0=                 ( Leap year?   )
 IF    1        ELSE    0                         ( Compensate?  )
 THEN  >R  $>D  2DROP  DUP  32 <                  ( January?     )
 IF    R> DROP  256 +          EXIT               (              )
 THEN  DUP  60 R@ + <                             ( February?    )
 IF    R> DROP  31 -  512 +    EXIT               (              )
 THEN  R> -  DUP  91 <                            ( March?       )
 IF    59 -  768 +             EXIT               (              )
 THEN  DUP  121 <                                 ( April?       )
 IF    90 -  1024 +            EXIT               (              )
 THEN  DUP  152 <                                 ( May?         )
 IF    120 -  1280 +           EXIT               (              )
 THEN  DUP  182 <                                 ( June?        )
 IF    151 -  1536 +           EXIT               (              )
 THEN  DUP  213 <                                 ( July?        )
 IF    181 -  1792 +           EXIT               (              )
 THEN  DUP  244 <                                 ( August?      )
```

Sat Aug 20, 1988 S.W.SQUIRES [scott]

Gary, Some of the previous messages make it a little difficult to figure out who is sending the message, especially the one from sysop. I assume that is Don Madison from the North Coast Forth Board?
—Scott

Category 14, Topic 2, Message 3
Sat Aug 20, 1988 GARY-S [Gary]

Yes, Scott—it was Don. Thank you for bringing the ambiguity to my attention. I see this may take some editing for some of the messages to be more coherent.
—Gary

Category 14, Topic 2, Message 4
Sat Sep 10, 1988 GARY-S [Gary]
> PORTED FROM xCFBs ==>
Date: 09-07-88 (23:50)
To: Gary Smith
From: Lee Brotzman
Subj: ForthNet

Gary, as far as BITNET and FIGI-L goes, if you connect to Usenet's comp.lang.forth, you are automagically connected to BITNET. We have a fully operational, two-way gateway between the groups. All the mail they send, we receive; and vice versa.

This sounds like a wonderful idea to me.

Also on the FIGI-L front: In July, I changed the format from sending compiled periodic digests of the mail traffic, to sending each mail message immediately out to the group. This has been working quite well. However, some of my subscribers actually preferred the digests. So, in the interest of fairness, I have started a "sister" list to FIGI-L that consists of digests of all the FIGI-L mail traffic. I now distribute this periodically to about 14 people that want it.

I hope to get a chance to upload archives of the FIGI-L digests to both the ECFB and GEnie sometime. I just have to get down to Goddard some evening in order to use my AT there to do the archiving/uploading. My little Apple here at home just isn't up to the task.

Date: 09-08-88
To: RJ Brown
From: Mahlon Kelly
Subj: Unix Forth

About a month ago, I tried to help a firm that tried to bring up their own Forth, based on polyForth. I said, use an existing commercial system. Many thousands of dollars later they agreed. The work done by Duncan, Callahan, or others has to be paid for. But it's worth it.

Date: 09-09-88
To: Gary Smith
From: Jerry Shifrin
Subj: ForthNet

Gary, I pretty much agree with Don on this. I think it's best to just transfer everything appearing in this conference, the general Forth discussion area. I don't really see a whole lot of virtue in transferring most questions — there are enough Forth experts on each of the boards to ensure reasonable responses to the bulk of the queries. Obviously, questions about arcane Forth implementations or strange computers will benefit from wider distribution.

I'd personally prefer to see this very wide area network used for such things as news items, novel Forth approaches, product descriptions and reviews, etc.

```
IF      212 -  2048 +          EXIT              (              )
THEN    DUP  274 <                               ( September?   )
IF      243 -  2304 +          EXIT              (              )
THEN    DUP  305 <                               ( October?     )
IF      273 -  2560 +          EXIT              (              )
THEN    DUP  335 <                               ( November?    )
IF      304 -  2816 +          EXIT              (              )
THEN    334 -  3072 +          ;                 ( December     )

( >$HH:MM:SS:DD hm ds -- $                                      )
( Converts the time integers hm ds to a time string in the      )
( 24 hour format hh:mm:ss:dd.                                   )

: >$HH:MM:SS:DD ( hm ds -- $ )
       DUP  255 AND 0 <# # # ASCII : HOLD #> $CNT@    ( Decisec)
            256 /   0 <# # # ASCII : HOLD #> $CNT@ $APPEND
       DUP  255 AND 0 <# # # ASCII : HOLD #> $CNT@ $APPEND
            256 /   0 <# # # #>              $CNT@ $APPEND  ;

( $HH:MM:SS:DD> $ -- hm ds                                      )
( Converts the time string in the 24 hour format hh:mm:ss:dd to)
( the time integers hm ds.                                      )

: $HH:MM:SS:DD> ( $ -- hm ds )
       $DUP $>D 1+ 0 $CNT SWAP - $RIGHT DROP 256 * ( Hours    )
       $DUP $>D 1+ 0 $CNT SWAP - $RIGHT DROP +     ( Minutes  )
       $DUP $>D 1+ 0 $CNT SWAP - $RIGHT DROP 256 * ( Seconds  )
            $>D 2DROP +  ;                         ( Decisecs )

( >$H:MM12      hm ds -- $                                      )
( Converts the time integers hm ds to a time string in the      )
( 12 hour format h:mm am or h:mm pm.                            )

: >$H:MM12      ( hm ds -- $ )
       DROP  DUP  255 AND                        ( Minutes      )
       0 <# # # ASCII : HOLD #> $CNT@            (              )
       256 /  DUP  11 >                          ( After noon?  )
IF     DUP  12 >                                 ( 1pm or later?)
 IF    12 -                                      (              )
 THEN  "  pm"                                    (              )
ELSE   DUP  0=                                   ( Midnight?    )
 IF    12 +                                      (              )
 THEN  "  am"                                    (              )
THEN   $@ $SWAP $APPEND  0 <# #S #> $CNT@  $APPEND  ;


( $H:MM12>      $ -- hm ds                                       )
( Converts the time string in the 12 hour format h:mm am or     )
( h:mm pm to the time integers hm ds. The ds value is 0.        )

: $H:MM12>      ( $ -- hm ds )
       $DUP  2 $RIGHT  " pm" $@  $=              ( After noon?  )
IF     -1      ELSE     0                        (              )
THEN   $DUP $>D 1+ 0 $CNT SWAP - $RIGHT DROP     ( Hours        )
       DUP  12 <  ROT AND                        ( 1-11pm       )
IF     12 +                                      (              )
ELSE   DUP  12 =                                 ( Midnight?    )
 IF    12 -                         THEN         (              )
THEN   256 *  $>D  2DROP +  0  ;                 ( Minutes ds   )
```

Advertisers Index

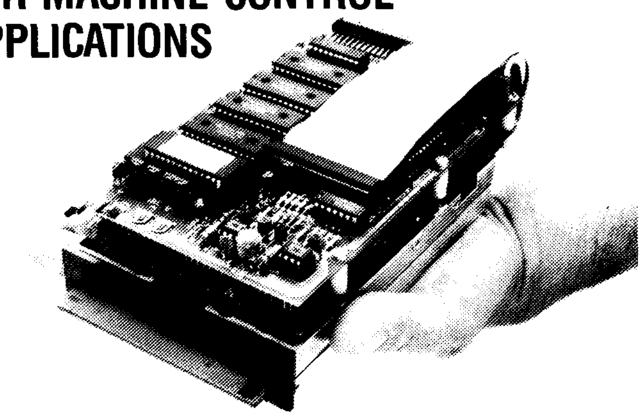
(Screens continued from page 24.)

```
Screen # 12
( RTTY - Receive and Transmit words       pac 11:55 09/12/86 )
: ?SIO  \  --- flag... is there a received char at the port?
   LSR @ PC@ DAV AND ;
: SKEY  \  --- char... if so, bring it to the stack
   DATAL @ PC@ ;
: SEMIT \  char ---... output one character
   BEGIN LSR @ PC@ TBE AND UNTIL DATAL @ PC! ;
```

(Screens continued from page 17.)

```
Scr # 4
 \ TIP ONE: IRA DEDUCTION
 : INCOME-PHRASE
      11 12 AT ." if your adjusted gross income is " ;
 : ADJUSTED-GROSS?   AMOUNT? RETURN1 Y/N
  IF REDO      NO-DEDUCTION ELSE  AMOUNT? RETURN2 Y/N
  IF REDO PARTIAL-DEDUCTION ELSE NO-PLAN INCOME-PHRASE RETURN3
        THEN THEN ;
 : PLAN   RETURN? ADJUSTED-GROSS? ;
 : IRA-TEST   PLAN?  IF PLAN   ELSE NO-PLAN ." ." THEN LOCKUP ;
 : WAIT   400 MS ;
 : END-APPL   UNSTACK DARK 0 0 BDOS ;
 : RUN-APPL   BOX MESSAGE WAIT
       BEGIN  IRA-TEST  RERUN?
              KEY ASCII =
       UNTIL   END-APPL ;
```


COMPLETE DEVELOPMENT SYSTEM FOR MACHINE CONTROL APPLICATIONS



TINY188 is a low cost "PC somewhat compatible" engine for OEM controller applications. A selection of high level languages is available in ROM.

DDS188 An optional development board with EPROM programmer, floppy disk controller and added memory, removes to lower target system cost.

Prices start at $269 each/$99 at 1,000.

Vesta Technology, Inc.

(303) 422-8088

Forth Programmers Needed

FIG CHAPTERS

The FIG Chapters listed below are currently registered as active with regular meetings. If your chapter listing is missing or incorrect, please contact Kent Safford at the FIG office's Chapter Desk. This listing will be updated in each issue of *Forth Dimensions*. If you would like to begin a FIG Chapter in your area, write for a "Chapter Kit and Application." Forth Interest Group, **P.O. Box 8231, San Jose, California 95155**

U.S.A.

- **ALABAMA**
  Huntsville Chapter
  Tom Konantz
  (205) 881-6483

- **ALASKA**
  Kodiak Area Chapter
  Horace Simmons
  (907) 486-5049

- **ARIZONA**
  Phoenix Chapter
  4th Thurs., 7:30 p.m.
  AZ State University
  Memorial Union, 2nd floor
  Dennis L. Wilson
  (602) 956-7578

- **ARKANSAS**
  Central Arkansas Chapter
  Little Rock
  2nd Sat., 2 p.m. &
  4th Wed., 7 p.m.
  Jungkind Photo, 12th & Main
  Gary Smith (501) 227-7817

- **CALIFORNIA**
  Los Angeles Chapter
  4th Sat., 10 a.m.
  Hawthorne Public Library
  12700 S. Grevillea Ave.
  Phillip Wasson
  (213) 649-1428

  North Bay Chapter
  2nd Sat., 10 a.m. Forth, AI
  12 Noon Tutorial, 1 p.m. Forth
  South Berkeley Public Library
  George Shaw (415) 276-5953

  Orange County Chapter
  4th Wed., 7 p.m.
  Fullerton Savings
  Huntington Beach
  Noshir Jesung (714) 842-3032

  San Diego Chapter
  Thursdays, 12 Noon
  Guy Kelly (619) 454-1307

  Sacramento Chapter
  4th Wed., 7 p.m.
  1708-59th St., Room A
  Tom Ghormley
  (916) 444-7775

  Silicon Valley Chapter
  4th Sat., 10 a.m.
  H-P Cupertino
  Bob Barr (408) 435-1616

  Stockton Chapter
  Doug Dillon (209) 931-2448

- **COLORADO**
  Denver Chapter
  1st Mon., 7 p.m.
  Clifford King (303) 693-3413

- **CONNECTICUT**
  Central Connecticut Chapter
  Charles Krajewski
  (203) 344-9996

- **FLORIDA**
  Orlando Chapter
  Every other Wed., 8 p.m.
  Herman B. Gibson
  (305) 855-4790

  Southeast Florida Chapter
  Coconut Grove Area
  John Forsberg (305) 252-0108

  Tampa Bay Chapter
  1st Wed., 7:30 p.m.
  Terry McNay (813) 725-1245

- **GEORGIA**
  Atlanta Chapter
  3rd Tues., 6:30 p.m.
  Western Sizzlen, Doraville
  Nick Hennenfent
  (404) 393-3010

- **ILLINOIS**
  Cache Forth Chapter
  Oak Park
  Clyde W. Phillips, Jr.
  (312) 386-3147

  Central Illinois Chapter
  Champaign
  Robert Illyes (217) 359-6039

- **INDIANA**
  Fort Wayne Chapter
  2nd Tues., 7 p.m.
  I/P Univ. Campus, B71 Neff Hall
  Blair MacDermid
  (219) 749-2042

- **IOWA**
  Central Iowa FIG Chapter
  1st Tues., 7:30 p.m.
  Iowa State Univ., 214 Comp. Sci.
  Rodrick Eldridge
  (515) 294-5659

  Fairfield FIG Chapter
  4th Day, 8:15 p.m.
  Gurdy Leete (515) 472-7077

- **MARYLAND**
  MDFIG
  Michael Nemeth
  (301) 262-8140

- **MASSACHUSETTS**
  Boston Chapter
  3rd Wed., 7 p.m.
  Honeywell
  300 Concord, Billerica
  Gary Chanson (617) 527-7206

- **MICHIGAN**
  Detroit/Ann Arbor Area
  4th Thurs.
  Tom Chrapkiewicz
  (313) 322-7862

- **MINNESOTA**
  MNFIG Chapter
  Minneapolis
  Even Month, 1st Mon., 7:30 p.m.
  Odd Month, 1st Sat., 9:30 a.m.
  Fred Olson (612) 588-9532
  NC Forth BBS (612) 483-6711

- **MISSOURI**
  Kansas City Chapter
  4th Tues., 7 p.m.
  Midwest Research Institute
  MAG Conference Center
  Linus Orth (913) 236-9189

  St. Louis Chapter
  1st Tues., 7 p.m.
  Thornhill Branch Library
  Robert Washam
  91 Weis Drive
  Ellisville, MO 63011

- **NEW JERSEY**
  New Jersey Chapter
  Rutgers Univ., Piscataway
  Nicholas Lordi
  (201) 338-9363

- **NEW MEXICO**
  Albuquerque Chapter
  1st Thurs., 7:30 p.m.
  Physics & Astronomy Bldg.
  Univ. of New Mexico
  Jon Bryan (505) 298-3292

- **NEW YORK**
  FIG, New York
  2nd Wed., 7:45 p.m.
  Manhattan
  Ron Martinez (212) 866-1157

  Rochester Chapter
  Odd month, 4th Sat., 1 p.m.
  Monroe Comm. College
  Bldg. 7, Rm.102
  Frank Lanzafame
  (716) 482-3398

- **OHIO**
  Cleveland Chapter
  4th Tues., 7 p.m.
  Chagrin Falls Library
  Gary Bergstrom
  (216) 247-2492

  Dayton Chapter
  2nd Tues. & 4th Wed., 6:30 p.m.
  CFC. 11 W. Monument Ave. #612
  Gary Ganger (513) 849-1483

- **OREGON**
  Willamette Valley Chapter
  4th Tues., 7 p.m.
  Linn-Benton Comm. College
  Pann McCuaig (503) 752-5113

- **TENNESSEE**
  East Tennessee Chapter
  Oak Ridge
  2nd Tues., 7:30 p.m.
  Sci. Appl. Int'l. Corp., 8th Fl
  800 Oak Ridge Turnpike
  Richard Secrist
  (615) 483-7242

- **TEXAS**
  Austin Chapter
  Matt Lawrence
  PO Box 180409
  Austin, TX 78718

  Dallas Chapter
  4th Thurs., 7:30 p.m.
  Texas Instruments
  13500 N. Central Expwy.
  Semiconductor Cafeteria
  Conference Room A
  Clif Penn (214) 995-2361

  Houston Chapter
  3rd Mon., 7:45 p.m.
  Intro Class 6:30 p.m.
  Univ. at St. Thomas
  Russell Harris (713) 461-1618

- **VERMONT**
  Vermont Chapter
  Vergennes
  3rd Mon., 7:30 p.m.
  Vergennes Union High School
  RM 210, Monkton Rd.
  Hal Clark (802) 453-4442

- **VIRGINIA**
  First Forth of Hampton Roads
  William Edmonds
  (804) 898-4099

  Potomac FIG
  D.C. & Northern Virginia
  1st Tues.
  Lee Recreation Center
  5722 Lee Hwy., Arlington
  Joseph Brown
  (703) 471-4409
  E. Coast Forth Board
  (703) 442-8695

  Richmond Forth Group
  2nd Wed., 7 p.m.
  154 Business School
  Univ. of Richmond
  Donald A. Full
  (804) 739-3623

- **WISCONSIN**
  Lake Superior Chapter
  2nd Fri., 7:30 p.m.
  1219 N. 21st St., Superior
  Allen Anway (715) 394-4061

INTERNATIONAL

- **AUSTRALIA**
  Melbourne Chapter
  1st Fri., 8 p.m.
  Lance Collins
  65 Martin Road
  Glen Iris, Victoria 3146
  03/29-2600
  BBS: 61 3 299 1787

  Sydney Chapter
  2nd Fri., 7 p.m.
  John Goodsell Bldg., RM LG19
  Univ. of New South Wales
  Peter Tregeagle
  10 Binda Rd., Yowie Bay 2228
  02/524-7490

- **BELGIUM**
  Belgium Chapter
  4th Wed., 8 p.m.
  Luk Van Loock
  Lariksdreff 20
  2120 Schoten
  03/658-6343

  Southern Belgium Chapter
  Jean-Marc Bertinchamps
  Rue N. Monnom, 2
  B-6290 Nalinnes
  071/213858

- **CANADA**
  BC FIG
  1st Thurs., 7:30 p.m.
  BCIT, 3700 Willingdon Ave.
  BBY, Rm. 1A-324
  Jack W. Brown (604) 596-9764
  BBS (604) 434-5886

  Northern Alberta Chapter
  4th Sat., 10a.m.-noon
  N. Alta. Inst. of Tech.
  Tony Van Muyden
  (403) 486-6666 (days)
  (403) 962-2203 (eves.)

  Southern Ontario Chapter
  Quarterly, 1st Sat., Mar., Jun., Sep., Dec., 2 p.m.
  Genl. Sci. Bldg., RM 212
  McMaster University
  Dr. N. Solntseff
  (416) 525-9140 x3443

  Toronto Chapter
  John Clark Smith
  PO Box 230, Station H
  Toronto, ON M4C 5J2

- **ENGLAND**
  Forth Interest Group-UK
  London
  1st Thurs., 7 p.m.
  Polytechnic of South Bank
  RM 408
  Borough Rd.
  D.J. Neale
  58 Woodland Way
  Morden, Surry SM4 4DS

- **HOLLAND**
  Holland Chapter
  Vic Van de Zande
  Finmark 7
  3831 JE Leusden

- **ITALY**
  FIG Italia
  Marco Tausel
  Via Gerolamo Forni 48
  20161 Milano
  02/435249

- **JAPAN**
  Japan Chapter
  Toshi Inoue
  Dept. of Mineral Dev. Eng.
  University of Tokyo
  7-3-1 Hongo, Bunkyo 113
  812-2111 x7073

- **NORWAY**
  Bergen Chapter
  Kjell Birger Faeraas,
  47-518-7784

- **REPUBLIC OF CHINA**
  R.O.C. Chapter
  Chin-Fu Liu
  5F, #10, Alley 5, Lane 107
  Fu-Hsin S. Rd. Sec. 1
  TaiPei, Taiwan 10639

- **SWEDEN**
  SweFIG
  Per Alm
  46/8-929631

- **SWITZERLAND**
  Swiss Chapter
  Max Hugelshofer
  Industrieberatung
  Ziberstrasse 6
  8152 Opfikon
  01 810 9289

SPECIAL GROUPS

- **NC4000 Users Group**
  John Carpenter
  (415) 960-1256 (eves.)
  1698 Villa St.
  Mountain View, CA 94041

JOIN THE

FORTH INTEREST GROUP RoundTable™ on GEnie™

INFORMATION SERVICES

GEnie™

General Electric Network for Information Exchange

- Over 600 Downloadable Files of Forth Information and Code
- Technical Information Exchange on our Bulletin Board and by E-Mail
- On-Line Real-Time Conferencing

SPECIAL SIGN-UP FOR FIG MEMBERS ONLY

$29.95

Includes GEnie Manual Plus 5 FREE Hours on GEnie

Using your modem call: 1-800-638-8369, type "HHH" (cr)
Following the U# prompt, type "XJM11849,GENIE" (cr)

Forth Interest Group
P.O.Box 8231
San Jose, CA 95155

Second Class
Postage Paid at
San Jose, CA