SIMPLE SCREEN DIRECTORY

STANDALONE APPLICATIONS IN F83

MENU-DRIVE THE 8250

MORE ABOUT STACKS...
SILICON COMPOSERS

Introduces the

SC/FOX™ Single Board Computer

SC/FOX (Forth Optimized eXpress™) 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™ 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™ 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™ Language in EPROM
- SC/Float™ Floating Point Library
- SC/SBC/PROTO™ Prototype Board
- SC/FOX/SCS™ 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
A SIMPLE SCREEN DIRECTORY - DAVID CORNELL

This utility allows you to assign a symbolic name to any screen, then it builds a directory of all the named screens in your file. LOAD, LIST, and EDIT recognize the screens’ names—so physical locations no longer matter—but use of conventional screen numbers is not affected. Low overhead, great convenience, and no code conversion required.

STANDALONE APPLICATIONS IN F83 - JAMES F. BALL

This article describes the steps required to generate a self-executing application in the F83 dialect of Forth. A modified system, containing a stripped-down F83 kernel and your application, is created via metacompilation. Not widely documented, the author found this approach used in Inside F83.

USING REGISTERS IN DATA STACKS - DON KENNEY

Usually, Forth systems implement a data stack in memory. But many CPUs handle register operations much more rapidly than the analogous memory operations. There are problems with keeping the whole data stack in registers, but this paper shows that mixed register-memory stacks can be much faster than pure memory stacks.

MENU-DRIVE THE 8250 ASYNC CHIP - PAUL COOPER

Talk about communications for long and you’re bound to run into the ubiquitous, asynchronous 8250 chip. This initialization routine lets you speak ASCII or Baudot, and allows the operator to rely on default values or to explicitly set word length, stop bits, and parity. Originally, it was part of an RTTY program for an amateur radio station.

DESIGNING DATA STRUCTURES - MIKE ELOLA

The chief concern of the third installment in this series is abstraction of the host computer, in the interests of program portability, with attention paid to a declaration syntax for portable arrays. For data structures, we often have had to write code that relies on host peculiarities, such as bit-processing widths. But no more!

USING A STRING STACK - RON BRAITHWAITE

In the last issue, the author presented his string package, based on the comprehensive string operations of the MUMPS computer language. It features a dedicated stack and a complete vocabulary, including pattern matching. Here, the remainder of his code is printed.

Editorial

Letters

Advertisers Index
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 microcomputer 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 further information. (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
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 nth stack. The contents of the cell to which the stack pointer points will change with manipulation of the stack (see Figure One).

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 nth 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
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

---

He Wants Proof
Dear Marlin,

I was glad to see your article about Shadow Stacks Get Smart (FD X/3). I have taken those ideas a little further since then.

By making !SHADOW state smart, you can eliminate the semi-kludgey '$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:

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

(Tuck on shadow stack.)

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

(State-smart !SHADOW.)

---

I should mention that I have written a little file that converts two shadow numbers to two double numbers:

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

(Continued on page 18.)
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. Real systems designed to fool benchmarks are slightly faster on nearly empty do loops, but bog down when the color 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 reuse segments anytime, without a system regen.

With the GigaForth option, your programs transparently enter native mode and expand into 16 Meg extended memory anytime, without a system regen.

Benefits beyond speed and program size include word hashed, or from 79 Standard to Forth 83. You can be absolute or scaled to current window.

Colon words become incredibly powerful, with primitives, great for interrupt 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 mnh@ read or write from 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 (FM Metagraphics). We use our Rosetta Stone Dynamic Linker to interface to Meta-windows. 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 required. Complex (mostly fast machine code) and parse format words cover numbers through 16 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 cis’ to improve both performance and compactness. Z = (X-Y) / (X+Y) would be coded: X @ Y @ Y @ + / (X @ @ @ + / @ @) (16 bytes) instead of: X @ @ Y @ Y @ + / 1 2 (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 multi-taskers require a word that exits cleanly at the end of subtask execution. The cooperative round robin multi-tasker provides individual user stack segments as well as user tables. Control passes to the next task/user whenever desired.

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 INTERRUPT 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 mnh@ read or write from 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 (FM Metagraphics). We use our Rosetta Stone Dynamic Linker to interface to Meta-windows. 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 required. Complex (mostly fast machine code) and parse format words cover numbers through 16 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 cis’ to improve both performance and compactness. Z = (X-Y) / (X+Y) would be coded: X @ Y @ Y @ + / (X @ @ @ + / @ @) (16 bytes) instead of: X @ @ Y @ Y @ + / 1 2 (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 multi-taskers require a word that exits cleanly at the end of subtask execution. The cooperative round robin multi-tasker provides individual user stack segments as well as user tables. Control passes to the next task/user whenever desired.

APPLICATION CREATION TECHNIQUES

HS/FORTH assemblies 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 Metacomplier 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 turnedkey 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 or to restore from a file. It is both an indispensable development aid and a great user interface. The macro editor provides reusable 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, compile, 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, 8087, dynamic strings, multitasker $245.

Professional 8087, assembler, turnkey, RSDL link, physical screens $495.

Production ROM, Metacomplier, MetaWindows $495.

Level upgrade, price difference plus $ 25.

OLB modules $495.

Rosetta Stone Dynamic Linker $ 95.

Metawindows by Metagraphics (includes RSDL) $1145.

Hardware Floating Point & Complex $ 95.

Quad integer, software floating point $ 45.

Time slice and round robin multitaskers $ 75.

GigaForth (90286-386 Native mode extension) $295.

HARVARD

SOFTWORKS

PO BOX 69

SPRINGBORO, OH 45066

(513) 746-0390
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 directory organization are shown in Figures One and Two. 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 catalog 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.
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 ?OBRANCH 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.

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.

6I
For my 32-bit 808x Forth, 6I 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
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 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.
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

\{ define \ i:: \}

\{ End of Line, all input from \i\ to end of line is treated as a comment \}

This word must be in the system. Screens are documented with \i\ and it use to define \i\ below

\i: C/L IN\ a Over MOD \i:IN\ +\ IMEDIATE
\i: [COMPILE]\ \i: IMEDIATE

\i: 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 5, 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, (bCCONFLICT) 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 "R", 2R\>\>\>\}
Same as \R \R \R \R

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

-- Doc USE

| TELLDIR | Display Directory |
| TELLSCE (name) | Display Screen with name |
| ni ni CAT-BLOCKS | Catalog blocks ni to ni, add to directory if identified as a logical screen by \i: (name:) on top line |
| n CAT-BLOCK | Catalog a single block |

CatFile - initialize directory, catalog a file or range as defined by \i\ to LastBk

LOADSCR (name) - loads the screen identified by \i\ in 1 he directory

-- Commenting Conventions

\$ - the address of a counted string, 1st byte is length byte
- either, eg. \a: \ = an address or 0
0 | 60 or 6d - "BLK Directory" related word
Embedded \$ - argument is the address of a string
\{ eg. b: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"
A as prefix, "Address, Address of"
p - as prefix, "Pointer to"

-- System Dependent

\{ LastBk \ \ - nbk, returns last b file or range \}
*************
Words \ b: \ as \ WorD \ the string address \\# simplified for this listing \*
\{ if only used at compile time, \ WorD \ may be sufficient \*
\{ \ for \ Forth \T:\B, \H ERE \ not required below \*
\{ EXEC WORD HERE PAD \# \$ PAD ;
\{ WordColsMax \ - \ n, returns Max 0 in current display, \\v00\ 
\{ general purpose default is width of CRT screen \\v0 
\{ SCRN \ nScr \ - \ favorite screen display routine. 
\{ LIST \ - \ general purpose default --
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 metacompilation 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.

**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 à 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 Laxon and Perry to load the utility and CPU system extensions (see standard system load screen 1, lines 7–8 in Figure One).

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.

Standard System Load Screen # 2

<table>
<thead>
<tr>
<th>Scr</th>
<th>2</th>
<th>A&gt;EXTEND86.BLK</th>
<th>08MAY84HHL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>\ Load up the system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Start only forth also definitions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>HELLO \ (S -- )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>START \ only forth also definitions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>HELLO IS BOOT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>HELLO IS BOOT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>HELLO IS BOOT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>MARK \ (S -- )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>CREATE \ DOES \ (FORGET) \ forth definitions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>MARK EMPTY \ HERE FENCE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>CREATE \ DOES \ (FORGET) \ forth definitions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>SAVE–SYSTEM \ APPL.COM \ Where APPL is a unique name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>SAVE–SYSTEM \ APPL.COM \ Where APPL is a unique name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>SAVE–SYSTEM \ APPL.COM \ Where APPL is a unique name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>SAVE–SYSTEM \ APPL.COM \ Where APPL is a unique name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>SAVE–SYSTEM \ APPL.COM \ Where APPL is a unique name</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Modified Load Screen # 2

<table>
<thead>
<tr>
<th>Scr</th>
<th>2</th>
<th>EXTEND86.BLK</th>
<th>JFB880601</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Load up your application</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HELLO \ (S -- )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>START \ only forth also definitions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>RUN–APPL \ Where RUN–APPL executes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>\ a program in APPL.BLK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>HELLO IS BOOT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>HELLO IS BOOT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>MARK \ (S -- )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>CREATE \ DOES \ (FORGET) \ forth definitions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>MARK EMPTY \ HERE FENCE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>CREATE \ DOES \ (FORGET) \ forth definitions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>SAVE–SYSTEM \ APPL.COM \ Where APPL is a unique name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>SAVE–SYSTEM \ APPL.COM \ Where APPL is a unique name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>SAVE–SYSTEM \ APPL.COM \ Where APPL is a unique name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>SAVE–SYSTEM \ APPL.COM \ Where APPL is a unique name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>SAVE–SYSTEM \ APPL.COM \ Where APPL is a unique name</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Application Blocks (APPL.BLK)

Scr # 1
0 \ TAX TIPS: INTRODUCTION
1 \ LOCKUP 0 -2 AT ;
2 \ VARIABLE ROW 1 CONSTANT .LEFT 79 CONSTANT .RIGHT
3 \ RESTORE 1 ROW ! ;
4 \ SIDES ROW @ 2+ DUP ROW ! AT ;
5 \ SPOT 219 EMIT SPACE ;
6 \ LSIDE RESTORE 8 0 DO .LEFT SIDES SPOT LOOP ;
7 \ RSIDE RESTORE 8 0 DO .RIGHT SIDES SPOT LOOP ;
8 \ BAR 40 0 DO SPOT LOOP ;
9 \ TOP 1 1 AT BAR ;
10 \ BOTTOM 1 19 AT BAR ;
11 \ INTRO 24 7 AT ." INCREDIBLE SOFTWARE PRESENTS "
12 \ 24 9 AT ." TAX TIPS "
13 \ 24 15 AT ." Programmed in F83 "
14 \ 24 17 AT ." By James F. Ball" LOCKUP ;
15 \ BOX DARK TOP LSIDE RSIDE BOTTOM ; -->

Scr # 2
0 \ TIP ONE: IRA DEDUCTION
1 \ TITLE
2 \ 11 3 AT ." TAX TIP 1: IS YOUR IRA CONTRIBUTION DEDUCTIBLE?"
3 \ 11 5 AT ." Answer the questions below by entering either: "
4 \ 11 7 AT ." Y = Yes " 11 8 AT ." N = No " ;
5 \ REDO DARK TITLE 11 11 AT ;
6 \ UNSTACK DEPTH 0 ?DO DROP LOOP ;
7 \ Y/N BEGIN KEY 95 AND DUP ASCII Y = SWAP ASCII N =
8 \ 2DUP OR UNTIL DROP ;
9 \ PLAN? REDO ." Are you or your spouse covered" OR 11 SPACES
10 \ ." by a retirement plan at work? " Y/N ;
11 \ AMOUNT? REDO ." Is your adjusted gross income " ;
12 \ REMUN? 11 19 AT ." Press Esc to quit or" 11 20 AT
13 \ ." any key to repeat." ;
14 \ NO-PLAN REDO
15 \ ." Your IRA contribution is 100% tax deductible" ; -->

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

(Screens continued on page 37.)
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, 80800, 8051,
8096, 68000, 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.

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, Trilsee-Neustadt 7651-1660
UK: System Science Ltd., London, 01-246 5862
Japan: Southern Pacific Ltd., Yokohama, 045-316-9514
Australia: Wave-ionic Associates, Wilson, W.A., (08) 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 ac-
cessed, 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 fig-
ures things out from existing information
(whenever the tradeoffs aren’t too oner-
ous).

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

<table>
<thead>
<tr>
<th>Word</th>
<th>freq.</th>
<th>pure-mem</th>
<th>1 register</th>
<th>2 registers</th>
<th>3 registers</th>
<th>4 registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>mem</td>
<td>reg</td>
<td>mem</td>
<td>reg</td>
<td>mem</td>
<td>reg</td>
<td>mem</td>
</tr>
<tr>
<td>@ C8 HERE</td>
<td>0.163</td>
<td>1 0</td>
<td>1 1</td>
<td>1 2</td>
<td>1 3</td>
<td>1 4</td>
</tr>
<tr>
<td>DROP</td>
<td>0.049</td>
<td>0 1</td>
<td>1 1</td>
<td>1 1</td>
<td>1 2</td>
<td>1 3</td>
</tr>
<tr>
<td>ROT</td>
<td>0.018</td>
<td>6 0</td>
<td>4 1</td>
<td>2 2</td>
<td>0 4</td>
<td>0 4</td>
</tr>
<tr>
<td>DUP</td>
<td>0.110</td>
<td>2 0</td>
<td>1 1</td>
<td>1 2</td>
<td>1 3</td>
<td>1 4</td>
</tr>
<tr>
<td>+ - * AND</td>
<td>0.163</td>
<td>3 0</td>
<td>1 1</td>
<td>1 1</td>
<td>1 2</td>
<td>1 3</td>
</tr>
<tr>
<td>SWAP</td>
<td>0.069</td>
<td>4 0</td>
<td>2 2</td>
<td>1 1</td>
<td>0 3</td>
<td>0 3</td>
</tr>
<tr>
<td>OVER</td>
<td>0.045</td>
<td>3 1</td>
<td>2 1</td>
<td>1 2</td>
<td>1 3</td>
<td>1 4</td>
</tr>
<tr>
<td>!</td>
<td>0.065</td>
<td>3 0</td>
<td>2 0</td>
<td>2 0</td>
<td>2 1</td>
<td>2 2</td>
</tr>
<tr>
<td>R&gt; &gt;R</td>
<td>0.049</td>
<td>1 0</td>
<td>1 0</td>
<td>1 1</td>
<td>1 2</td>
<td>1 3</td>
</tr>
<tr>
<td>0 1 2 3</td>
<td>0.200</td>
<td>1 0</td>
<td>1 1</td>
<td>1 2</td>
<td>1 3</td>
<td>1 4</td>
</tr>
<tr>
<td>1+ 2+ 2*</td>
<td>0.064</td>
<td>2 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
</tbody>
</table>

Weighted sums:

<table>
<thead>
<tr>
<th></th>
<th>pure-mem</th>
<th>1 register</th>
<th>2 registers</th>
<th>3 registers</th>
<th>4 registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:1 6809</td>
<td>2.071</td>
<td>2.063</td>
<td>2.429</td>
<td>3.346</td>
<td>4.194</td>
</tr>
<tr>
<td>3:1 6809</td>
<td>6.023</td>
<td>4.408</td>
<td>4.470</td>
<td>5.173</td>
<td>6.021</td>
</tr>
<tr>
<td>4:1 6809</td>
<td>7.999</td>
<td>5.581</td>
<td>5.490</td>
<td>6.087</td>
<td>6.935</td>
</tr>
<tr>
<td>5:1 80186, 8086</td>
<td>9.975</td>
<td>6.754</td>
<td>6.510</td>
<td>7.000</td>
<td>7.848</td>
</tr>
<tr>
<td>6:1 8088</td>
<td>11.95</td>
<td>7.926</td>
<td>7.531</td>
<td>7.914</td>
<td>8.762</td>
</tr>
</tbody>
</table>

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

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

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

---

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 Mountains 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.
Depend on how many serial ports
your PC will support, you can expand or
shrink this program to fit. The X-16 sup-
ports 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 pro-
gram does not concern itself with any inter-
rupt status; we are running in halfduplex
communication on either
HF or VHF ama-
teur 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 rela-
tionship 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 trans-
mitter 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 auto-
matically sets Baudot at five bits, 1.5 stop
bits, and 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 vari-
bables, 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 auto-
matic 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 applica-
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
to word 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 func-
tion, even though the 8250 has that provi-
sion. You can see how it is done from West-
ern Digital Corporation's excellent data
book on this chip and others that it manu-
factures. 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 INIT-
COM is entered, which prompts the oper-
ator 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 as 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 re-
ceiving 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
ANDed with the data derived from a port
fetch of the line status register; if the result
Screen # 2

( RTTY - Array words  pac 15:47 09/10/86 )

: INCARR \ n cells -- (name)...creates indexed array
  CREATE 2d HERE OVER ERASE ALLT ;CODE
  0, 16 BY 1 ADD 20, 72 BY 2 ADD
  RTTY NEXT,  END-CODE

VARIABLE $DUMMY

: <<ark mark stack top, to fill indexed array) SP $DUMMY !
: >>ark end of fill then fill array
  $DUMMY @ SP @ - 6 OVER + DO ! -2 =LOOP ;

: DOWNPAGE 10 DO CR LOOP ;

-->

Screen # 4

( RTTY - B250 Address selection  pac 15:32 09/11/86 )

HEX

: SELECT_ADDR \ select B250 port addresses
  CLS
  DOWNPAGE OF SPACES
  ." WHICH SERIAL PORT ARE YOU GOING TO USE?" 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 DF 3F LCR \ 3F8 DATAL \ 3F9 DATAB \ 3FD LSR \ ENDF
  42 DF 2F LCR \ 2F8 DATAL \ 2FF DATAB \ 2FD LSR \ ENDF
  43 DF 3E LCR \ 3E8 DATAL \ 3EF DATAB \ 3ED LSR \ ENDF
  44 DF 2E LCR \ 2E8 DATAL \ 2EF DATAB \ 2ED LSR \ ENDF
  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 DF 0 B/A ! ENDF
  ( baudot) 42 DF -1 B/A ! 04 LCR @ PC! ENDF
  ( 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 DF 09 D2 ENDF 42 DF 09 00 ENDF 43 DF 06 B8 ENDF
  44 DF 06 00 ENDF 45 DF 04 7E ENDF 46 DF 04 17 ENDF
  47 DF 03 59 ENDF 48 DF 03 00 ENDF 49 DF 01 80 ENDF
  4A DF 00 00 ENDF 4B DF 00 60 ENDF 4C DF 00 40 ENDF
  4D DF 00 3A ENDF 4E DF 00 3E ENDF 4F DF 00 20 ENDF
  50 DF 00 18 ENDF 51 DF 00 10 ENDF 52 DF 00 0C ENDF
  53 DF 00 06 ENDF 54 DF 00 03 ENDF 55 DF 00 02 ENDF
  ENDCASE ; \ see B250 data book for these values and
  \ note that you can program any intermediate
  \ value between 9 and 56k baud

DECIMAL

-->

Screen # 7

( RTTY - Baud rate selection - BAUDREQUEST pac 11:53 09/12/86 )

VARIABLE KEYPRESS 66 KEYPRESS !
5 INDBRR RATE1 \ establish 1st part of baud schedule
  "5 66 75 100 110 120 RATE1"
12 INDBRR RATE2
  "5 66 600 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 ."
  3 SPACES ." 45.45 6 SPACES ." A CR
  5 0 DO I RATE1 @ S.R ." .00 6 SPACES KEYPRESS @ EMIT
  1 KEYPRESS @ CR LOOP 72 KEYPRESS !
  2 SPACES ." 134.5 6 SPACES ." 5 CR
  12 0 DO I RATE2 @ S.R ." .00 6 SPACES KEYPRESS @ EMIT
  1 KEYPRESS @ CR LOOP -->
**Screen # 8**

( RTTY - BAUDREQUEST, cont'd.  pac 11:54 09/12/86 )

```
15 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 )

```
SCREEN # 9

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 )

```
SCREEN # 10

HEX

: WORDREQUEST DLS CR 6 SPACES " Choose bit length, stop "
" bit, 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 )

```
SCREEN # 11

HEX

: INITCOM \ initialize the serial port selected

SELECT_ADDR

B0 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

-->
```

Table Two. Relationship of serial ports and addresses.

<table>
<thead>
<tr>
<th></th>
<th>DATAL</th>
<th>DATAH</th>
<th>LCR</th>
<th>LSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM1</td>
<td>3F8</td>
<td>3F9</td>
<td>3FB</td>
<td>3FD</td>
</tr>
<tr>
<td>COM2</td>
<td>2F8</td>
<td>2F9</td>
<td>2FB</td>
<td>2FD</td>
</tr>
<tr>
<td>COM3</td>
<td>3E8</td>
<td>3E9</td>
<td>3EB</td>
<td>3ED</td>
</tr>
<tr>
<td>COM4</td>
<td>2E8</td>
<td>2E9</td>
<td>2EB</td>
<td>2ED</td>
</tr>
</tbody>
</table>

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.
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.
Host Abstraction

Two cascaded forms of abstraction have been suggested to make data objects more portable. 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 \texttt{BYTES>ADR} (“bytes-to-addresses”). Once the correct number of bytes is known, \texttt{BYTES>ADR} finds the corresponding number of addresses. To discover how many addresses are spanned by a byte on any host, type:

\begin{verbatim}
1 \textbf{BYTES>ADR}
\end{verbatim}

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 \texttt{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 \texttt{BYTES>ADR}.

Also note that the definition of \texttt{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. Specific mappings can be performed by constants, such as:

\begin{verbatim}
BYTES/CELL (-- #bytes )
\end{verbatim}

To avoid having to write \texttt{BYTES/CELL} and \texttt{BYTES>ADR}, both types of mappings can be consolidated as the host-dependent constant \texttt{ADR/}, (“addresses-per-cell-compile”).

Another convenient constant is \texttt{ADR/C}, (“addresses-per-character-compile”), which replaces \texttt{1 BYTES>ADR}. Similarly, the constant \texttt{ADR/D}, (“addresses-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:

\begin{verbatim}
BYTES>ADR ( #bytes -- #addresses )
BYTES/CELL ( -- #bytes )
ADR/ ( -- #addresses )
ADR/C ( -- #addresses )
ADR/D ( -- #addresses )
\end{verbatim}

An additional constant has been included to help with the suite’s customization for a particular host. This constant is \texttt{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 \texttt{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...
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 must 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

5x2ARRAY

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)

BYTE>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> -- )
0 * 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 ;

Uses, 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 pro-
vide increased program readability.

As a parting observation, consider how we specify memory allocations with AL-
LOT. 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
I am going to take a completely different tack 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]**

Messages posted here are posted 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 [scottl]

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
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 Forts 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
Read #1 # 14: Ridu dum kiam vi povas, simiu-lo-knabo!(bandy)
Mon, Aug 22, '88

The two bugs in the editor (they refuse to believe 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 (à 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, simiu-lo-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
( $VERIFY 1$ 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@ $SP @ 1$ + J + $C@ = ( Equal? )
IF DROP -1$ LEAVE THEN ( Found it )
LOOP 0$- ( Get out? )
IF DROP I LEAVE THEN ( <> at 1$ 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 - $SRIGHT $SSWAP $SLEFT ( Make 3$ & 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 )

: $SOUNDEX 0$ -- 1$
 64 - $DUP 0$< OVER 27 < OR ( In range? )
IF " 0123012022455012623010202" + $C@ ( Get code )
( ABCDEFGHJKLMNOPQRSTUVWXYZ) ( Corresponding )
ELSE DROP ASC$ + ( 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 converging be-
tween MacForth and MACH2. The Forth-
language-based differences aren't too se-
vere, MACH2 conforms almost com-
pletely 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 in-
terface 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 bot-
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

---

: $SOUNDEX ( 0S -- 1$ )
  $UPPER 1 HERE C! $SP@ COUNT 0> ( Not null? )
  IF $C@ ( Get char )
  ELSE DROP ASCII 0
  THEN HERE 1+ C! 0 $CNT 1 > ( Store 1st chr )
  IF $SP@ 1+ C@ C>SNDX ( Last char )
  $SP@ 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" $S HERE $S$APPEND 4 $LEFT ; ( 4char code )

( $SMATCH 1$ 0S -- 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. )

: $SMATCH ( 1$ 0S -- flag )
  -1 $SP@ COUNT OVER + SWAP ( Flag, do 0$ )
  ?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 ( )

Forth Dimensions 32 Volume X, Number 4
both products. Both products do multi-tasking, 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

```forth
?DO I C8 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 C8 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 ( )
IF DROP 0 LEAVE THEN ( )
LOOP AND DUP 0= ( )
IF LEAVE THEN ( )
ENDOF ASCII A ( Alphabetic? )
OF -1 1$ COUNT OVER + SWAP ( )
?DO I C8 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 C8 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 C8 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 C8 127 > ( )
IF DROP 0 LEAVE THEN ( )
LOOP AND DUP 0= ( )
IF LEAVE THEN ( )
ENDOF ASCII ' ( Literal? )
OF 0 1$ COUNT OVER + SWAP ( )
?DO I C8 J 1+ C8 = ( )
IF DROP -1 LEAVE THEN ( )
LOOP AND DUP 0= ( )
IF LEAVE THEN ( )
ENDOF ASCII ~ ( Literal NOT? )
OF -1 1$ COUNT OVER + SWAP ( )
?DO I C8 J 1+ C8 = ( )
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 yyyy month day)
```
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 ARCe'd 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

Forth Dimensions 34 Volume X, Number 4
( >$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 ENDIF  ( January 31 )
  2 OF 31 ENDIF  ( February 28 )
  3 OF 59 ENDIF  ( March 31 )
  4 OF 90 ENDIF  ( April 30 )
  5 OF 120 ENDIF  ( May 31 )
  6 OF 151 ENDIF  ( June 30 )
  7 OF 181 ENDIF  ( July 31 )
  8 OF 212 ENDIF  ( August 31 )
  9 OF 243 ENDIF  ( September 30 )
  10 OF 273 ENDIF  ( October 31 )
  11 OF 304 ENDIF  ( November 30 )
  12 OF 334 ENDIF 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 S>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 automatically 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 hh ds -- $ )
( Converts the time integers hh:mm:ss:dd to a time string in the )
( 24 hour format hh:mm:ss:dd. )

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

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

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

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

: >$H:M:12 ( hh ds -- $ )
DROP DUP 255 AND
0 << ASCIIS HOLD #> $CNT@ ( Afternoon? )
256 / DUP 11 > ( 1pm or later? )
IF DUP 12 > ( Am )
THEN " pm" ( )
ELSE DUP 0=
  ( Midnight? )
IF 12 + ( )
THEN " am" ( )
THEN $@ $SSWAP $SAPPEND 0 << # $S #> $CNT@ $SAPPEND ;

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

: $H:M:12> ( $ -- hh ds )
$DUP 2 $RIGHT " pm" $@ $= ( Afternoon? )
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 ( Minutes )
THEN 256 * $D 2DROP + 0 ; ( Minutes )
```
(Screens continued from page 17.)

```
Screen # 4
\TIP ONE: IRA DEDUCTION
1 \ INCOME-PHRASE
   11 12 AT " if your adjusted gross income is " ;
2 \ ADJUSTED-GROSS? AMOUNT? RETURN1 Y/N
3 \ IF REDO NO-DEDUCTION ELSE AMOUNT? RETURN2 Y/N
4 \ IF REDO PARTIAL-DEDUCTION ELSE NO-PLAN INCOME-PHRASE RETURN3
5 \ THEN THEN ;
6 \ PLAN RETURN? ADJUSTED-GROSS? ;
7 \ IRA-TEST PLAN IF PLAN ELSE NO-PLAN " ." THEN LOCKUP ;
8 \ WAIT 400 MS ;
9 \ END-APPL UNSTACK DARK 0 0 BDOS ;
10 \ RUN-APPL BOX MESSAGE WAIT
11 \ BEGIN IRA-TEST RERUN?
12 \ KEY ASCII =
13 \ 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 Wassen
  (213) 649-1428
  North Bay Chapter
  2nd Sat., 10 a.m. Forth, Ar
  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
  Nosher Jesung (714) 842-3032
  San Diego Chapter
  Thursdays, 12 Noon
  Guy Kelly (619) 454-1307
  Sacramento Chapter
  4th Wed., 7 p.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) 386-5147
  Central Illinois Chapter
  Champaign
  Robert Ilyes (217) 359-6039
- INDIANA
  Fort Wayne Chapter
  2nd Tues., 7 p.m.
  IP 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 Chropkiewicz
  (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
  Santa Fe Chapter
  3rd Sat., 7 p.m.
  El Farol
  Richard Kwan
  (505) 957-9893
- NEW YORK
  Metropolitan Chapter
  2nd Mon., 7 p.m.
  Robert Kocen
  (212) 329-8989
  NYFIG Chapter
  3rd Thurs., 7 p.m.
  John Wol myston
  (516) 837-2153
- OHIO
  Cleveland Chapter
  2nd Tues., 7 p.m.
  Western Reserve University
  Nature Center
  Robert Kocen
  (212) 329-8989
- PHILADELPHIA
  Philadelphia FIG
  4th Tues., 7 p.m.
  De Sales University
  Marc Hadas
  (215) 855-3500
- TENNESSEE
  Central Tennessee FIG
  1st Fri., 7 p.m.
  Middle Tennessee State University
  Murphy Library
  John L. Hackett
  (615) 898-5950
- TEXAS
  Southwestern FIG
  2nd Tues., 7 p.m.
  Southwestern College
  James J. Verrette
  (806) 792-7731
- VIRGINIA
  Northern Virginia FIG
  1st Wed., 7 p.m.
  Northern Virginia Community College
  John Scott
  (703) 822-5100
- WASHINGTON
  SeattleFIG
  1st Mon., 7 p.m.
  Seattle University
  Royce H. Barlow
  (206) 221-7200
- WEST VIRGINIA
  Charleston FIG
  1st Mon., 7 p.m.
  Huntington College
  Beth S. Heading
  (304) 526-7902

Forth Dimensions 38 Volume X, Number 4
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

ROCester Chapter
Odd month, 4th Sat., 1 p.m.
Monroe Comm. College
Bldg. 7, Rm.102
Frank LanzaFane (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.
Lincoln-Benton Comm. College
Pann McCuaig (503) 752-5113

TENNESSEE
East Tennessee Chapter
Oak Ridge
2nd Tues., 7:30 p.m.
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
7722 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
Lariksreid 20
2120 Schoten
03/658-6343

Southern Belgium Chapter
Jean-Marc Bertinchamps
Rue N. Momont, 2
B-6290 Naliningen
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 Tues., 7:30 p.m.
McMaster University
Dr. N. Solntseff
(416) 960-1256 (eves.)

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
(021) 810 9289

HOLLAND
Holland Chapter
Victor 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,
47-516-7784

• 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 Amin
46/8-929631

• SWITZERLAND
Swiss Chapter
Max Hugelshofer
Industrieberatung
Zibertstrasse 6
8152 Opikon
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™

- 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