EGGS, OVALS EASY

FILLING ALGORITHMS

ACCESS EXTENDED MEMORY

TWO ASSEMBLERS ARE BETTER THAN ONE
SILICON COMPOSERS

Introduces the
World's First 32-bit Forth Microprocessor:

SC32 Stack-Chip

Also, the SC/FOX Parallel Coprocessor System32, a PC based development system.

SC/FOX PCS32 (Parallel Coprocessor System32)

SC32 Stack-Chip Hardware Features

- 32-bit CMOS microprocessor, 34,000 transistors.
- One clock-cycle instruction execution.
- Non-multiplexed 32-bit address bus and data bus.
- 16 gigabyte non-segmented data space.
- 2 gigabyte non-segmented code space.
- 8 or 10 megahertz full-static operation.
- Stack depths limited only by available memory.
- Interrupt and interrupt acknowledge lines.
- Bus request and bus grant lines with on-chip tristate.
- Wait state line for slow memory and I/O devices.
- 84-pin ceramic PGA package.
- Unit Price starts at $195.

SC/FOX PCS32 (Parallel Coprocessor System32)

- Uses the SC32 Stack-Chip microprocessor.
- System speed options: 8 or 10 MHz.
- Full-length 8 or 16-bit PC/XT/AT/386 plug-in board.
- 64K to 1M bytes, 0 wait-state static RAM.
- Hardware expansion, two 50-pin strip headers.
- Operates concurrently with PC/XT/AT/386 host.
- Multiple PCS32 board parallel operation.
- PCS32 memory accessible by PC host.
- PCS32 controlled through PC I/O space.
- Data transfer through 16K shared memory window.
- Prices start at $1,995 with SC/Forth32.

Ideal for embedded systems control, multitasking and multiple-processor operation, data acquisition, image processing, or computation intense applications. For additional information, contact us at:

Silicon Composers, Inc., 210 California Avenue, Suite K, Palo Alto, CA 94306 (415) 322-8763
The method for drawing described here generates shapes like footballs, ellipses, and eggs. An oval is generated by three radii, variations in which produce several classes of oval objects. And it is just as easy to draw filled ovals...

Filling algorithms are specially suited to Forth because of the ease of using the stack for data storage. The author discusses filling areas of known contour and filling polygons whose vertices are known.

Another editor, yes, but with differences... multi-file editing, windows, copying screens between files, single-step execution—with stack control—of on-screen words, nested VIEWS, and more. Even if you don’t use F83 (the flavor of this code), you will certainly want to adopt some of the author’s ideas.

MS-DOS was written for the 80286’s “real” mode, leaving 15 unaddressable Mb of memory that is further limited by the PC architecture. Even with extended memory cards, a program under MS-DOS cannot directly read from or write to such memory. The ROM BIOS does, however, provide a way out. Here are some Forth words that access memory well above the one-meg boundary.

Here the author describes his environment in which to write expert systems. It uses a set of Forth words to define the rules, questions, and answers one finds in an expert system. There are just three basic steps: define the questions, the rules, and at last the order of the rules and the answers they lead to.

The Forth assembler is handy for short pieces of code but cumbersome for large routines. A full-fledged macro assembler is ideal for larger routines but messy to implement. Fortunately, we can use a regular macro assembler and treat the binary output as a Forth word.
Call for Articles:

**FORTH HARDWARE**

Creative Solutions’ Don Colburn was recently attending a developers conference where he met, by chance, one of the Forth Interest Group’s directors. He suggested *Forth Dimensions* pay for articles about a particular theme announced by the editor. I’m usually in favor of paying authors, so I passed the idea to FIG for consideration along with some other items. We didn’t end up with a fixed policy, but we will try it and let the results do the talking.

Therefore, we are happy to offer payments of $450, $300, and $225, respectively, to the three authors whose articles are chosen as best suited for the upcoming theme issue about “Forth Hardware.” The issue is currently scheduled for *FD XI/6*. We need your theme-related submission by November 1, 1989 to be included in this offer.

I will work closely with our technical reviewers to choose articles that merit payment and publication in the theme issue, and those decisions will be final. If an article of high quality and interest is not among the three selected, *Forth Dimensions* may ask to publish it later under our standard terms: payment in the form of peer recognition, technical feedback, and a couple of free copies. Along with, of course, the satisfaction that comes from making a contribution to our growing body of common knowledge.

Now is the time to jot down your article idea, and maybe a brief outline from which to work. Deadlines lurch up unexpectedly fast, especially when you want to track down the last idea you had while polishing that “final” draft and it’s time to prettify the code for publication.

I look forward to hearing from you!

* * * * *

If you haven’t rushed out to write about Forth hardware yet, you probably have a few minutes to peruse the rest of this issue. Dust off your graphics vocabulary and dig into “Eggs, Ovals Easy” (my nomination for best original title) and “Filling Algorithms.” The last is by one of our authors in Poland, where several items have been published about Forth. We have very few details, and can only speculate about how Forth is being used there and how well it is regarded.

In fact, I received a couple of manuscripts from Poland at about the time Jack Woehr’s Chapter Coordinator column pointed out that Forth interest overseas is far from on the wane. Another two articles from Spain showed up in the mail that same week—one appears in this issue titled, “Expert Systems Toolkit.” We appreciate the efforts of these far-flung fellow Forth users, many of whom will be key to the future of Forth development around the world. We welcome each of them to our community—may we learn a great deal from one another.

—Marlin Ouverson
Editor

---

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

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

Copyright © 1989 by Forth Interest Group, Inc. The material contained in this periodical (but not the code) is copyrighted by the individual authors of the articles and by Forth Interest Group, Inc., respectively. Any reproduction or use of this periodical as it is compiled or the articles, except reproductions for non-commercial purposes, without the written permission of Forth Interest Group, Inc. is a violation of the Copyright Laws. Any code bearing a copyright notice, however, can be used only with permission of the copyright holder.

About the Forth Interest Group

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

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

One comment I want to give is that we should adopt the Modula-2 definition module for specifying all Forth abstract data types. This would allow standard libraries to be developed from Forth implementation modules by using the vocabulary structure. Definition modules are readily available for lists, priority queues, binary search trees, AVL trees, records, B trees, sets, polynomials, and graphs.

Also, I want to see more written about using a Forth Prolog interpreter for conversions between regular expressions, deterministic and nondeterministic finite automata, push-down automata, context-free grammars, and Turing machines.

Some discussion about the limits of computability and the Chomsky Hierarchy would help set the diverse Forth community on common ground for an even more productive future.

John Howard
627 N.E. Terrace Drive
Kansas City, Missouri 64116

Curious About Code With Class
Dear Marlin,

A significant reward for being published is the comfort derived from knowing that your interests are shared by others. I now know that Dr. Ayman Abu-Mostafa shares my interest in object-oriented Forth (OOF). For this I am thankful and relieved. Before seeing his letter, I was worried that the audience of Forth Dimensions may not really be interested in this topic. While that possibility remains, at least I have found one voice willing to speak up in support of OOF.

I am overjoyed to hear Forth programmers support object-oriented Forth programming. By taking the role of devil's advocate and criticizing OOF, I hoped to motivate readers to develop better object-oriented Forths than have been offered so far in the Forth journals. At least some of my whines were considered legitimate, such as the mixing of postfix and prefix notation while manipulating data ("It's a bad design"). By the way, Rick Hoselton managed to get the syntax correct on the first try (See "Object-Oriented Forth" in Forth Dimensions X/2).

Clear progress is being made to overcome all the obstacles to OOF. Still, it's hard to express an educated opinion regarding OOF without more published works to draw upon. If I had access to as-yet unpublished works, I might only have positive things left to say about object-oriented Forth (OOF).

As further explanation of my point of view, realize that the guidance offered in my series of articles [FD X/2-5] was the design of data structures, not how to best produce object-oriented Forth. However, to the extent that I observed how well the object-oriented programming model resulted in more portable and reusable designs, I felt compelled to acknowledge OOF and to encourage its study.

Now back to my provocations: I am extremely curious about the implementations Abu-Mostafa has come up with to support postfix messaging, private and public methods, and inheritance.

Of extreme interest is the embodiment of several data and colon-definition declarations within a structure that looks like a super colon definition (delimited by ::CLASS and ;CLASS). What change does this make to the Forth landscape?

I assume that each instance of a class is created by honoring each of the data declarations in the hierarchy of classes. For example, FIXED-STACK creates an instance object by creating the BOS and TOS instance variables inherited from the STACK class, followed by the MAXSIZE instance variable directly declared in the FIXED-STACK class, followed by space for the stack itself. Wow! Presumably the name fields, such as TOS and BOS, aren't recreated with each instance object. Rather, those names should become shared addressing methods for FLOAT and STACK classes (as well as any of their descendent classes).

Less inspirationally, there remains the need for programmer-supplied typing of objects. Abu-Mostafa has provided his object-oriented method of taking the average of two numbers:

```
USE FLOAT
A @
B @
+ 2/ 
C !
```

The introductory phrase USE FLOAT should not be necessary in an object-oriented language. Because objects such as A and B are known to be of a given class, the methods for fetching and adding them should be automatically predetermined. Only those methods owned by an object should be available for use, and the use of any other methods should result in error messages.

I will stand by my description of standard Forth as having a two-stack architec-

(Continued on page 35.)
The method for drawing ovals described here was found in the course of a project I was working on that required drawing arrows on the screen. An oval is, technically, a closed curve bounding a convex domain. It applies to curves shaped like footballs, ellipses, or eggs. Ovals have a certain aesthetic appeal that I find a relief from all the straight lines and rectangles we usually see on computer screens.

Figure One shows the underlying structure of the method. An oval is taken to be generated by three radii R1, R2, and R3. The figure shows the classes of oval objects that can be generated by varying just these radii. The key word in the source code is BARC. BARC stands for biradial arc; i.e., it draws an arc specified by two lengths, R1 and R2. If R1=R2 the arc will be circular. As R2 increases, so does the curvature of the arc.

Closed curves bounding convex domains...

We can define the OVAL function as follows:
OVAL(R1,R2,R3) = (BARC(R1,R2),BARC(R1,R3))

where the separate BARCs sweep out the northern and southern hemiovals.

Figure Two shows the typical egg shape defined by OVAL(40,90,50). (In this demo program, all parameters are positive integers.) Figure Three shows a family of a dozen eggs sorted by their point height.

Figure Four shows a nest of eggs of varying girths and point heights. You might like to experiment with the source code and modify it for applications. It is easy to draw filled ovals just by replacing the PIX! function with the corresponding LINE function. You may also want to experiment with the explicit form of the OVAL function, which can be obtained as follows: solve the following parametric equations of an ellipse for T.

\[ Y = R2 \cos(T) + C \]
\[ X = R1 \sin(T) + D \]

Take the cosine of both sides, and apply the identity:
\[ \cos(SIN^{-1}(T)) = \sqrt{1-T^2} \]

\[ Y = R2 \sqrt{1-(X-D)^2/R1^2} + C \]

Figure One. Relationship of three radii to geometrical shapes. OVAL(R1,R2,R3)

(Text continued on page 35, Figures continued on page 8.)
YES, THERE IS A BETTER WAY
A FORTH THAT ACTUALLY
DELIVERS ON THE PROMISE

HS/FORTH

POWER
HS/FORTH’s compilation and execution speeds are unsurpassed. Compiling at 20,000 lines per minute, it compiles faster than many systems link. For real jobs execution speed is unsurpassed as well. Even non-optimized programs run as fast as ones produced by most C compilers. For systems designed to fool benchmarks are steadily faster on nearly empty do
loops, but bog down when the colon nesting level ap-
proaches 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 seg-
ments anytime, without a system regen. With the
GigaForth option, your programs transparently enter
native mode and expand into 16 Meg extended memory
or a gigabyte of virtual, and run almost as fast as in real
mode.

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

Colon definitions can execute inside machine code
primitives, grow, interrupt & exception handlers.
Multi-cfa words are easily implemented. And code
words become incredibly powerful, with multiple entry
points not requiring jumps over word fragments. One of
many reasons our system is much more compact than
its immense dictionary (16,000 words) would imply

INCREDIBLE FLEXIBILITY

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

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

FUNCTIONALITY

More important than how fast a system executes, is
whether it can do the job at all. Can it work with your
computer. Can it work with your other tools. Can it trans-
form 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 lan-
guage. You can execute function calls, DOS com-
mands, other programs interactively, from definitions,
or even from files being loaded. DOS and BIOS function
calls are well documented HS/FORTH words, we don’t
settle for giving you an INTCALL and saying “have at it”.
We also include both fatal and informative DOS error
handlers, installed by executing FATAL or INFORM.

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

Our formatting and parsing words are without equal. In-
teger, double, quad, financial, scaled, time, date, float-
ing 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 lan-
guage. Other words treat files like memory, nn@H and
nnH read or write from/to a handle (file or device) as
fast as possible. For advanced file support, HS/FORTH
easily links to STRIEVE, 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
capped, and work with our penplot routines. While great
for plotting and line drawing, it doesn’t approach the ca-
pabilities of Metawindows (tm Metagraphics). We use
our Rosetta Stone Dynamic Linker to interface to Meta-
windows. HS/FORTH with MetaWindows makes an un-
beatable graphics system. Or Rosetta to your own preferred
graphics driver.

HS/FORTH provides hardware/software floating point,
including trig and transcendental. Hardware ‘fp covers
full range trig, log, exponential functions plus complex
and hyperbolic counterparts, and all stack and compar-
ison ops. HS/FORTH supports all 8087 data types and
works in RADIANS or DEGREES mode. No coproces-
sor. No problem. Operations, (mostly fast machine code)
and parse format words cover numbers through 18 dig-
its. Software ‘fp eliminates conversion round off error
and minimizes conversion time.

Single element through 4D arrays for all data types in-
cluding complex use multiple cfas to improve both per-
formance and compactness. Z = (X+Y) would be
coded: X Y X Y + /SIZ (16 bytes) instead of: X (cfa)
Y (cfa) E Y E X + /Z (25 bytes) Arrays can ignore 64k
boundaries. Words use SYNONYMds for data type inde-
pendence. HS/FORTH can even prompt the user for
retyr on erroneous numeric input.

The HS/FORTH machine code string library with up to
3D arrays is with out 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 assembles to any segment to create stand
alone programs of any size. The optimizer can use HS/
FORTH as a macro library, or complex macros can be
built as colon words. Full forward and reverse labeled
branches and calls complement structured flow control
Complete syntax checking protects you. Assembler
programming has never been so easy.

The Metacomplier produces thread systems from a
few hundred bytes, or For th kernels from 2k bytes. With
it, you can create any threading scheme or segmenta-
tion architecture to run on disk or ROM.

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

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

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

HS/FORTH lacks a “module” compilation environ-
ment. 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 ink — 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 are available separately.

Documentation & Working Demo
(3 books, 1000 + pages, 6 lbs) $ 95.
Student $145.
Personal optimizer, scaled & quad integer $245.
Professional 8067, assembler, turnkey, dynamic strings, multtasker
RSDL linker, physical screens
Production ROM, Metacomplier, Metawindows $495.
Level upgrade, price difference plus $ 25.
OBJ modules $495.
Rosetta Stone Dynamic Linker $ 95.
Metawindows by Metagraphics (includes RSDL) $395.
Hardware Floating Point & Complex $ 95.
Quad integer, software floating point $ 45.
Time slice and round robin multtaskers $ 75.
GigaForth (60286'886 Native mode extension) $295.

HARVARD
SOFTWARES
PO BOX 69
SOUTHBRIDGE, OH 45666
(513) 748-0390

Volume XI, Number 2
7
Forth Dimensions
Glossary

Special Terms

biradial arc
A circular arc is swept out by a single radius around a point; a biradial arc is swept out by a varying-length line segment arrived at using the values of two fixed radii. A biradial arc becomes a circular arc only when the two radii are equal.

convex
A plane curve is convex if any straight line cuts the curve in just two points.

hemioval
The northern or southern part of a generalized oval, divided by the equator.

oval
A closed curve bounding a convex domain.

pole
The point at which the height of a hemioval is maximum or minimum.

HS/Forth Words

FIND <word> 0= ? ( ... )
Similar to IF THEN and used to load extension modules as needed, i.e. if <word> wasn't found.

VAR
A word created by VAR has three possible actions: return a stored number, store a number, or return the address where a number is stored. VARs are a hybrid between constants and variables. If X is a var, then 10 IS X will store 10 at the address of X, and X alone will simply leave 10 thereafter.

FI
A loop index used by the coprocessor, similar to I.

FSIN, FCOS, F+, F-, F*
Math functions directed toward the coprocessor.

S->F and F->S
Convert single numbers to and from floating-point numbers on the respective stacks.

% Precedes a number to be put on the floating-point stack.

Figure Two. Typical egg shape.

Figure Three. Eggs of increasing point height.

Figure Four. Nested eggs of varying point heights and equatorial radii.

```
FOR X=D-R1 TO D+R1 STEP .03
Y1=R3 * SQR(1-((X-D)/R1)^2) + C
Y2=R2 * -SQR(1-((X-D)/R1)^2) + C
PSET(X, Y1), 1
PSET(X, Y2), 1
NEXT X
```

Figure Five. The explicit oval function written in BASICA.
\ Source code for “Eggs, ovals easy,” by Robert Garian, 24Dec88
\ Written in Harvard Softworks’ HS/Forth 3.4 for IBM PC and compatibles
\ Uses Math Coprocessor.

\ FIND DG->RD 0= ?( FLOAD TRIG ) DECIMAL \ Floating pt. & Trig. ftns.
\ FLOATS
FINIT \ Initialize coprocessor
\ DEGREES
\ EW640 \ Compute in degrees
\ high resolution mode
0 VAR C \ See Fig.1
0 VAR D
0 VAR R1
0 VAR R2
0 VAR R3
0 VAR STP
1 VAR KOLOR
: ORIGIN 99 IS C 319 IS D ; \ centers oval at (99,319)
: BARC ( TO-ANGLE FROM-ANGLE - ) \ Note: .416667=5/12 =aspect
DO F1 FDUP \ next angle
FSIN R2 S->F % .416667 F* F* F->S C + \ compute correspond. row
FCOS R1 S->F F* F->S D + \ compute correspond. col
KOLOR PIX! ) LOOP ; \ set pixel

: OVAL ( - ) \ draw northern arc
361 181 BARC \ draw equator
R3 IS R2
181 0 BARC ; \ draw southern arc
: NEST-OF-EGGS WIPE ORIGIN
CR " Equatorial radius is increasing" \ fixed R3
90 IS R3
30 IS STP
300 50 DO I IS R1
50 150 DO I IS R2
NEW R1 \ new R2
NEW R2
OVAL \ draw the oval
STP -1 * +LOOP
STP +LOOP ;

: ONE-EGG \ draw an egg to specifications
ORIGIN
CR " Use whole numbers between 10 and 200. Typical egg is 50 100 75."
CR " Enter radius at equator: “ #IN IS R1
CR " Enter height of NORTH pole: “ #IN IS R2
CR " Enter height of SOUTH pole: “ #IN IS R3
WIPE OVAL ;

6 VAR #COLS
2 VAR #ROWS
0 VAR HJUMP \ horizontal spacing
0 VAR VJUMP \ vertical spacing
20 VAR F1 \ stretch factor
30 VAR F2 \ stretch factor

(Code continued on page 37.)
Algorithms making use of a stack for storing data are quite often applied in practice. However, if a high-level language is used, it is usually hard to implement a stack efficiently. This does not concern Forth, of course, which is specially suited to such algorithms. Filling algorithms are very applicable here.

The task of filling an area may be defined as follows:

- Filling an area of known contour (not necessarily regular).
- Filling the interior of a polygon (not necessarily convex) whose vertices' coordinates are known.

"Forth is specially suited to such algorithms."

Let us begin with the first algorithm, realized as the word Fill in Listing One. The algorithm assumes that the closed contour of an area is stored in video memory, and that the coordinates of a point lying within the contour are known. This point is called the seed, and its coordinates are pushed onto the stack. In the next step, the seed point is displayed at a given position and intervals connecting it with the contour to the right and left are drawn. At that time, the coordinates of the extreme points Xleft and Xright of the contour are calculated. Next (Scan) is called, which investigates the line above and the line below the plotted interval (limited by Xleft and Xright). The investigation's effect is to find new seeds, whose coordinates are pushed onto the stack. The described actions, as shown in Figure One, are repeated until the stack is empty. The Dot word—used by Fill—allows displaying a point at coordinates X,Y while Point checks to see if the point is displayed.

The second filling algorithm, realized as the word Area in Listing Two, requires that the coordinates of the vertices of a filled polygon be determined. The algorithm does not need a bit map of the image, so this method is used by some plotters. The contour must be closed (i.e., the coordinates of the first vertex must be identical with those of the last). First, the contour is drawn using Line. Next, the minimum and maximum values of coordinate Y are found by searching through all the vertices' coordinates. This is done by MaxY and MinY. Now, for each line Y falling within the determined interval, the X coordinates of the intersections with all the intervals of the contour are found. It can be proven that, for a closed contour, the number of intersections with line Y=const is even. To fill a contour like that, the found points must first be adequately connected, as shown in Figure Two.

The problem in this case is that the X coordinates of the intersections are not de-
rived in order. Thus, the coordinates must be sorted before the points can be connected. However, we usually do not know a priori how many intersections will be obtained. Thus, it is best to store the subsequent X coordinates on the stack, which is sorted by Sort each time a new number is pushed onto it. To make the stack sorting process more effective, the return stack has been applied. The sorting operation is realized by insertion. The idea of the process is that all coordinates smaller than the one to be inserted are popped from the stack and temporarily stored on the return stack. Then the inserted coordinate is pushed onto the stack and all the coordinates on the return stack are pushed back to the stack. After all the intersections are derived and sorted, Draw can connect the points whose coordinates are stored on the stack, thus filling the polygon. Draw (called by Area) draws a line to the point X,Y while Plot just moves to the point.

Listing One.

```
variable xleft      variable xright
variable xr         variable yr

: right>              ( --- )
  1 xr +! begin
    xr @ yr @ point 0= while
    xr @ yr @ dot 1 xr +!
    repeat xr @ 1- xright ! ;

: left>               ( --- )
  -1 xr +! begin
    xr @ yr @ point 0= while
    xr @ yr @ dot -1 xr +!
    repeat xr @ 1+ xleft ! ;

: (scan)              ( --- x y ... )
  begin xr @ xright @ <= while
    0 flag ! begin
      xr @ yr @ point 0= xr @ xright @ <= and while
      1 flag ! 1 xr +! repeat
    flag @ if
      xr @ yr @ point 0= xr @ xright @ = and if
      xr @ yr @ else xr @ 1- yr @ then then
      xr @ begin
      xr @ yr @ point xr @ xright @ < and while
      1 xr +! repeat
      xr @ = if 1 xr +! then
    repeat ;
```

Call for Articles about
Forth Hardware

Forth Dimensions will publish an issue about Forth hardware in the coming months.

To encourage high-quality article submissions, we will be offering payment for the theme-related articles we publish in that issue.

<table>
<thead>
<tr>
<th>Place</th>
<th>Prize</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>$450</td>
</tr>
<tr>
<td>2nd</td>
<td>$300</td>
</tr>
<tr>
<td>3rd</td>
<td>$225</td>
</tr>
</tbody>
</table>

Your article is welcome. For more, see the “Editorial” in this issue.

Send article, self-addressed stamped envelope, and (optional) Macintosh or IBM 5.25" diskette:

Editor
Forth Interest Group
P.O. Box 8231
San Jose, California 95155
(Listing One, continued.)

variable stack

: fill    ( x y -- )
  sp @ 4 + stack !
begin
  yr ! xr !
  x @ yr @ dot
  yr @ right> xr ! left>
  xleft @ xr ! 1 yr +! (scan)
  xleft @ xr ! -2 yr +! (scan)
  sp @ stack @ = until ;

Listing Two.

: points   ( yn xn ..., y1 x1 n -- )
create 0 do , , loop does> ;

: .x   ( adr n -- x [n] ) 2* 2* + @ ;

: .y   ( adr n -- y [n] ) 2* 2* 2* + @ ;

: line   ( adr n -- )
swap dup dup 0 .x swap 0 .y plot
swap 1 do
  dup dup i .x swap i .y draw
loop drop ;

variable pts variable vert

: sort   ( xn ..., x1 x1 n -- )
xmin @ begin sp @ stack @ <> while
  2dup if swap then >r repeat
begin dup xmin @ <> while r> repeat drop ;

: maxy   ( -- maxy )
ymin @ vert @ 0 do pts @ i .y max loop ;

: miny   ( -- miny )
ymin @ vert @ 0 do pts @ i .y min loop ;

: drawscan  ( ys -- )
  >r begin sp @ stack @ <> while r> plot r> draw repeat r> drop ;

: area   ( adr n -- )
2dup line vert ! pts ! .
sp @ stack ! maxy 1+ miny do
vert @ 1- 0 do
  pts @ i .y pts @ i 1+ .y
2dup min j < rot rot 2dup max j = rot rot
<> and and if
  j pts @ i .y -
  pts @ i 1+ .x pts @ i .x -
  pts @ i 1+ .y pts @ i .y -
  #/mod swap drop pts @ i .x + sort then
loop 1? drawscan
loop ;
CALL FOR PAPERS
for the eleventh annual

FORML CONFERENCE

The original technical conference
for professional Forth programmers, managers, vendors, and users.

Following Thanksgiving, November 24–26, 1989

Asilomar Conference Center
Monterey Peninsula overlooking the Pacific Ocean
Pacific Grove, California U.S.A.

Theme: Forth and Object-Oriented Programming

Data structures to support object-oriented program development are readily constructed in Forth. This is possible because Forth is an extensible language that has unique properties for building data structures. These structures may be reused which increases productivity when new applications are developed. Papers are invited that address relevant issues in the development of object-oriented programming and object-oriented applications.

Papers about other Forth topics are also welcome.

Mail abstract(s) of approximately 100 words by September 1, 1989 to:

FORML
P. O. Box 8231
San Jose, CA 95155

Completed papers are due November 1, 1989.

For registration information telephone the Forth Interest Group business office at (408) 277-0668 or write to FORML, P.O. Box 8231, San Jose, CA 95155.

Asilomar is a wonderful place for a conference. It combines comfortable meeting and living accommodations with secluded forests on a Pacific Ocean beach. Registration includes deluxe rooms, all meals, and nightly wine and cheese parties.
This article is about PDE, a program editor for Forth (F83 flavour). Yet another editor? Yes, but with a few differences.

The PDE (Program Development Environment) editor is a derivative of the Henry Laxen WS-like screen editor, and provides many added capabilities, including most of those in VED and FSED, two other editors derived from the original Laxen editor.

Over the years, I had adapted the Laxen editor to a variety of computers (TRS-80, Osborne, IBM PC), and had built in a considerable number of features for testing and debugging. Using that as the backbone, I incorporated the best of FSED and VED and a whole slew of further enhancements. The result is PDE.

"It is integrated and easy to use."

PDE provides:

* editing any number of files
* copying screens or parts thereof between files
* step-wise execution of on-screen word definitions, with inspection and modification of the stack
* inspection of word definitions (built-in SEE and VIEW), with nesting of VIEWS and return to screen being edited
* LOAD part of screen
* upper/lower case and numeric base conversion
* screen insert and delete
* two-window support, one for screen being edited and one scrolling, as for DEBUG output

```
." W2-SCROLL INSTALL
: W2-SCROLL (S -- )
  0 12 AT -LINE 0 18 AT OUT OFF ;

: INSTALL [') W2-SCROLL IS CR ;
```

Figure One. Scrolling window on lower part of screen

```
." INSTRING - Find string S within string B - fast
: INSTRING (S sadr len Pos -- blen 0 )
  2OVER >R C@ >R \ slen & c to >R
  ROT - 0 MAX DUP IF \ Only if blen >= slen
  1+ \ Length to scan (blen-slen+1)
  BEGIN R@ \ sadr badr blen' c
  USCAN DUP IF \ sadr badr blen'
  RD 2OVER R@ COMPARE \ 0=equal, +1 or -1 for not eql
  SWAP >R \ c back to >R
  ELSE 0 THEN \ Fix # of parms on stack
  WHILE 1 -1 D+ REPEAT \ Step past matching char
  THEN R> R> 2DROP \ sadr badr' blen'
  -ROT 2DROP ; \ blen' - length not searched
```

Figure Two. SEARCH for substring, case insensitive.

```
." INS-STR (S addr len Pos -- )
DUP >R 2DUP + DUP CHARS-TO-EOL/B BUF-MV \ Make space (len)
R> BADDR SWAP MOVE E-UPD ;
```

Figure Three. Text insert and replace.
This article describes some of the more interesting techniques used in PDE to allow others to tinker with and enhance PDE. Only a few of the 80-plus screens are covered here.

PDE is in the public domain (non-commercial use) and its source code is available on the East Coast Forth Board [see “Reference Section”], Canada Remote Systems (416-231-0538) and “other fine bulletin boards.” That version of PDE (ver 1.02+) is specific to Laxen and Perry’s F83 and to the IBM PC. PDE is virtually all high-level code and can be readily adapted to other systems.

PDE includes ideas and code from a number of public-domain sources, and I’ll give credit to those sources in the following.

: STEP (S -- ) ( Executes one word & displays stack )
  EXTRACT-WD SET-TIB \ Start addr of word in buff
  EXEC-WD \ copy word & execute
  0 MV-CURS ; \ Restore cursor in CRT screen

: 2STEP (S -> ) ( Executes 2 words & display stack )
  EXTRACT-WD SET-TIB \ Start addr of word in buff
  EXEC-WD ADD-TIB \ Addr of next word in buff
  EXEC-WD \ copy word & execute
  0 MV-CURS ; \ Restore cursor in CRT screen

Figure Four. Execute one or two on-screen words.

: DO-WD (S caddr -- ?? )
  EXTRACT-WD SET-TIB \ Get word pointed to in screen
  @W2AT (EXEC-WD) ; \ Go do words

: SEE-WD (S -- )
  ['] SEE DO-WD ; \ Execute SEE against TIB
  ( SEE-WD issues SEE against word pointed to)

: DEBUG-WD (S -- )
  ['] DEBUG DO-WD ; \ Execute DEBUG against TIB

: (EVIEW) ['] DO-WD ((EVIEW)) NXT-SCR-FILE ;

Figure Five. Operate on word under cursor.

---

**SDS FORTH for the 8051**

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

**Software Features**
- Supports Intel 805x, 87C51FA, 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**
- 150+ pages reference manual, hot line, 8051 development board available now

Limited development system, including PC software and 8051 compiled software with manual, for $150.00.

(generates ROMable applications on top of the development system)

SDS Inc., 5375 Pare Avenue #210, Montreal, QC, Canada  H4P 1P7  (514) 731-5797
Self-Contained Editor

PDE is self-contained and does not need the traditional F83 EDITOR vocabulary. Instead, screens 5B–5E (all numbers in hex) are adapted from F83's screens 0C–1B of the file UTILITY.BLK. From there comes the scrolling window, where the top 12 lines of the CRT display the screen under test, while the lower portion scrolls to allow debug output, etc.

Figure One shows the two words that take care of scrolling, W2-SCROLL and INSTALL, which installs W2-SCROLL as CR. W2-SCROLL (adapted from EDITOR's. ALL on screen 13), uses -LINE to delete the top line of the second window, causing everything below it to scroll up.

Note that this form of windowing works by preempting the PC's ROM routines from scrolling the whole screen. As long as CR is used to force a new line, we're okay. When we output a line of more than 50h characters, or one that includes characters 0D or 0A, the whole screen scrolls and our windowing gets messed up. That is one reason I'm using a version of .S that only displays the top six items on the stack, and thereby avoids line wrap on the display. (This .S also handles stack underflow—it is found in screen 5A.)

The other words from EDITOR are cursor addressing and GET-ID, etc. GET-ID (screen F) uses the system clock; it does not require a hardware clock. Simply set the PC's system date using the usual DOS command. Most of the supporting code on screen five is lifted from FSED.

Search Capability

The original Laxen editor used SCAN+ and SCAN- for forward and backward scanning to find a character; SCAN+<> and SCAN-<> to scan for the first occurrence of anything other than the character. These were high-level definitions and, given their frequent use, kind of slow.

F83, of course, has a machine code SCAN and SKIP, which are the same as SCAN+ and SCAN+<> respectively, though with some stack differences. Screens six through eight implement a machine-language version of SCAN- and SKIP- for both 8086 and 8080 assembler.

Also included in screens nine and OA are USCAN and INSTRING (Figure Two), which together are a faster version of SEARCH. Both INSTRING and SEARCH

---

Forth Dimensions 16 Volume XI, Number 2
can do a case-insensitive search (where upper-case and lower-case characters are acceptable).

**SEARCH** uses **COMPARE** against every possible substring. For instance, if looking for a five-character string within a 400h character block, it would compare the search string against characters one through five of the block, then against two through six, then three through seven, etc. More than a thousand times!

**INSTRING** goes on the basis that we first look for the first character of the search string, then see if the rest matches. Since the machine-language **SCAN** is so much faster than a high-level loop, we gain. Of course, if we’re looking for a string starting with a blank in a screen that’s 75 percent blanks, we don’t gain all that much.

For an even faster **SEARCH**, look at the machine-language version published in *Forth Dimensions*.* That version is case-sensitive, however. It also returns the absolute memory address where the substring was found, rather than the offset in the block as returned by the F83 SEARCH. **INSTRING** returns the length “not searched,” i.e., the offset relative to the end of the block. Hence the different name: it is not identical to **SEARCH**.

**INSTRING** is used by the multi-screen search capability (**QF** in good WordStar fashion), the heart of which are **QF**, **QF-STR** and **REPL** in screen 26; see Figure Three.

**INS-STR** uses **CHAR-TO-EOL/B** to determine the number of characters to the end of the line or block. (When you insert a string, the text to the right of the insertion point has to be moved over, with truncation occurring at the end. Similar to deletion, padding occurs at the end.) PDE will support truncation/padding at either end of the line, or at the end of the block (or any number in between; see **PAGEL**, **LINE-MODE**, and **BLOCK-MODE** in screen 12). The Laxen editor supported only end-of-line truncation. Thus, if text was bumped past the end of the line, it was lost.

By supporting end-of-block truncation, anything bumped off one line gets carried over to the next. It does mean, however, that the entire rest of the screen may change and need to be re-displayed, which makes this mode too slow for normal use.

---

**Execution of Words on the Screen**

One of the powerful features of PDE is that it can take words off the screen, move them to TIB (the keyboard input buffer), and then interpret them. The idea for this originally came from Tom Blakeslee, whose code I had adopted as-is; when I started factoring his code into smaller words, it became really interesting!

**STEP** (screen 45) was the original word (see Figure Four). The basic process is:

**EXTRACT-WD**

Get the memory address and length of the word (character string) where the cursor is at on the screen, and skip past it.

**SET-TIB**

Copy that string to the TIB and set #TIB accordingly.

**EXEC-WD**

Issue **INTERPRET** to handle the text string now in TIB, showing the stack afterwards.

Now, by factoring **EXEC-WD** into an (**EXEC-WD**) that can execute words other than **INTERPRET**, we can easily do some interesting stuff, such as **SEE** a word, **VIEW/FIX** or **DEBUG** a word—see Figure Five. Note that the approach taken is not to redirect **SOURCE**, since any errors would "hang" the system—see **eLOAD** below.

Similarly, **EXTRACT-WD**, which returns the memory address and length (minus trailing blanks) of the word (text string) under the cursor (and moves the cursor to the next word), makes it easy to implement other transformations, such as **LWCASE**, **UPCASE** and **NUM>NUM**, for case or number base conversion; see Figure Six.

**PROC-WD** sets everything up, then executes the word whose code field address (CFA) is passed and re-displays the screen (from the original cursor position). Note the use of **DISPL-TO-EOL/S** in **PROC-WD** to allow for actions that affect more than just the current screen line.

The word **LOWER** in **LWCASE** is similar to the F83 code word **UPPER**, except it changes alphas to lower case.

**Multi-File Editing**

F83 maintains its block buffers by using a list of buffer headers that describe where in memory the block is, etc. The buffer header also contains a pointer to the FCB (File Control Block) for the file that contains this block.

**BLOCK**, the word to read a specified block, passes to **(BLOCK)**—the primitive that actually brings in the wanted block (or locates it, if already in a buffer)—the address of the FCB to use (from the pointer **FILE**).
That made it very easy to implement multi-file capability. Simply set FILE to point to a different FCB, make sure the file is open (OPEN-FILE), and issue BLOCK (GET-FILE) in screen 34 and Figure Seven does that.

Well, almost that easy. If the current block was changed, UPDATE has to be executed before issuing BLOCK, because it always applies to the current block—the one in the first position in the buffer header list.

There appears to be yet another problem. At least some updates to a block would not be written out when switching between files. The SAVE-BUFFERS in (GET-FILE) and the extra code in MARK-UPD (Figure Seven) should not be necessary, but seem to have fixed the problem.

Another place where multi-file capability had an impact was in the TAGS. FSED used five to six fields to note a specific screen and line number, e.g., to mark a spot for fast return, or to mark a block or range of lines for copying. PDE extends this in two ways. Firstly, every TAG includes the

FCB for that block; secondly, every tag includes the exact cursor location at the time the tag was noted.

PDE also maintains a circular TAG list of the last ten screens edited. This allows us to quickly revisit screens or to return from inspecting definitions of words (EVIEW).

A very powerful capability in PDE is on-screen VIEW (EVIEW, screen 47, Figure Eight). While (EVIEW) is simply an adaptation of F83's VIEW, it is important because it is integrated and easy to use.

Say you're writing a new definition that uses SEARCH, but don't remember what SEARCH needs/leaves on the stack. Simply position the cursor at the start of the word SEARCH on the screen (which you had typed as part of the definition you're creating), press ^F5 and, presto, you're looking at the screen on which SEARCH is defined (screen 0A in UTILITY.BLK), including the comment that describes its stack effect. Use ^PgDn to get to its shadow.

If you want more detail, such as about the word \STRING that SEARCH uses, point, press ^F5 again, and you're looking at screen 3F of KERNEL86.BLK. A repeated ^F6 gets you back where you were.

Of course, all of this supposes that the words you're looking up (SEARCH and \STRING, in our example) are "visible" in a context vocabulary, and that you used VIEWS, etc. when originally loading those definitions. And, yes, it supposes you have a hard disk to have all your permanent source files on line. If not, get one; this feature alone makes it worthwhile: on-line documentation for your entire Forth environment!

Load from the Editor

Occasionally you want to LOAD only part of a screen, such as when you're correcting the second or third definition on a screen. PDE has the ability to mark the beginning and end of a section of a block (^F6, ^F8) and then issue the eLOAD command (see Figure Nine).

eLOAD works by setting the SOURCE that F83 uses (for LOAD as well as for INTERPRET). (eSOURCE) is similar to the regular (SOURCE) in that it can accept input from TIB as well. While eLOAD never uses input from that source (it sets BLK), it is very important.

If there were an error in the (sub) block being loaded, F83 would ABORT/QUIT and reset for keyboard input. It would not reset SOURCE, however, but would look to (eSOURCE) for input. If (eSOURCE) did not allow for keyboard input, you'd have F83 waiting for keyboard input without ever seeing any.

(eSOURCE) uses GET-SUB-BLK and GET-CUR-ADDR (Figure Ten) to return the address and length of the marked section, where 2 TAG@ and 3 TAG@ define the screen number, FCB, and cursor for the start and end of the section. Note how GET-SUB-BLK and GET-CUR-ADDR return an address in two components: start of block address and offset of start of sub-block within that block. That drastically simplified the implementation of (eSOURCE), since it allows us to use >IN to tell F83 "we've already processed that portion of the block."

Miscellaneous

A few other techniques are of note:

eKEY in screen four handles the PC's peculiarity of returning certain keystrokes as two characters (the "extended" keys,

(Continued on page 35.)
Although the 80286 processor is capable of addressing up to 16 Mb of memory in its protected mode, MS-DOS—for reasons of downward compatibility—was written for the processor's "real" mode. That mode has an absolute limit of 1 Mb of address space, which is further limited by a PC architecture that reserves the upper 384 Kb for special purposes. As a result, MS-DOS programs have access to a maximum of 640 Kb of address space. Even though one can plug so-called extended memory cards with addresses above 1 Mb into a PC/AT, a program under MS-DOS cannot directly read from or write to such memory. This is, of course, not a good thing.

The ROM BIOS in a PC/AT does, however, provide a way out. A service at interrupt 15 (hex), function 87 moves blocks of data between extended memory and DOS-accessible memory. Prior to calling this function, a Global Descriptor Table (GDT) that lists the source and destination addresses must be set up in memory, and the table's own address must be placed in extended memory. My own need for this technique was put to the test while working on a video digitizer (Data Translation's DT2851 frame grabber) in an application where I wanted to manipulate an image directly. The digitizer has two on-board image buffers, each 256 Kb in size and each capable of holding a 512 x 512 x 8-bit image. Obviously, this total of 512 Kb of image memory cannot readily be placed in MS-DOS' meagre 640 Kb, since that would leave essentially no memory for the operating system and programs. In fact, the digitizer's default address settings put the two image buffers at (hex) A0 0000 and A4 0000—well above 10 0000, the 1 Mb boundary. To be able to inspect and manipulate images, move them to disk, and do other operations that require direct access to the digitized image, it was necessary first to move part of the image down into DOS-accessible memory. (This is the same technique used by standard software for extended memory cards.) The largest block of data that interrupt 15 can move is one segment, or 64 Kb; thus, a 256 Kb image would have to be treated as at least four separate segments. However, I actually found it more efficient to move one video line at a time (512 bytes). The main reason is that in our implementation of Forth-83 (Uniforth from Unified Software Systems), it is much easier to perform subsequent operations on data that is held in the same memory segment as the Forth dictionary, rather than to use the more cumbersome and time-consuming procedures that manipulate data held in other segments. Consequently, the application examples described below all work with single video lines.

The most generic of my Forth words are shown on screens 1 and 2. GDT is a 48-byte array that will hold the Global Descriptor Table. MAKE-GDT sets up the descriptor table, based on 32-bit double addresses for the source and destination, and a 16-bit count of the number of bytes to be moved. (The table actually requires 24-bit addresses but, for convenience, 32-bit operators are used and the highest-order byte is later overwritten.) The addresses are calculated by subsequent words in the listings. The code word INT15, written in assembly language, calls the BIOS interrupt to move the data. INT15 requires 16-bit addresses for the start of the descriptor table and the table's segment, and for the number of words (not bytes) to be moved. (In Uniforth's 8086 assembler, unlike Intel's assembler, the syntax of instructions like AX SI MOV follows the Forth practice of first giving the source, then the destination, then the action.) Both MAKE-GDT and INT15 are general-purpose words that can be used to move any batch of data in any direction.

The use of these words is illustrated in excerpts from an imaging application on screens three through nine. A 512-byte array (VIDLINE) is created in the Forth dictionary to hold one video line. Absolute 32-bit addresses for this array and for the image buffers, required by MAKE-GDT, are calculated by words in screens three, four, and six. (Some of the defining words, like DSEG and SEG>ADR, are specific to Uniforth.) New Forth words that move single video lines are DICT>VIDLINE (screen five), which copies the VIDLINE array from the dictionary to the image buffer; and VIDLINE>DICT (screen six), which copies video line Y from the image buffer to the array. These words are typically used in the opposite order from which they are presented: a video line is moved from the image to the dictionary.
manipulated, and then moved back to the image. Since a video line is usually replaced at the same location it was taken from, \texttt{DICT>VLINE} does not automatically recalculate the Y-value address. (This lack of symmetry was adopted for speed, but it strikes me as a potentially risky programming style.) A few words that act on images are shown as examples in screen seven. \texttt{!PIXEL} and \texttt{@PIXEL} access a single pixel (picture element); when one considers how much hidden work these words invoke, screen seven may also be taken as a commentary on the disastrous architecture of Intel processors. Nevertheless, filling the screen (\texttt{VFILL}) takes only about a second.

Transferring images to and from disk (screens eight and nine) is an example of the more general task of moving data between a disk and extended memory. The “save” routines utilize a loop that copies each of an image’s 480 lines into the \texttt{VIDLINE} array, from which the line is written to disk. (The disk-handling words are from Uniforth.) The real work of copying images is carried out by the intermediate-level words \texttt{IMAGE>DISK} and \texttt{DISK>IMAGE}; the high-level words \texttt{SAVE} and \texttt{RESTORE} handle the additional tasks of specifying filenames, and opening and closing files.

(Screens on next page.)

Richard F. Olivo, a neurobiologist, is a professor at Smith College. He uses Forth for laboratory-oriented imaging and analog data acquisition. He is currently programming a PCIAT data acquisition system with drop-down menus, dialog boxes, and other Macintosh-like tools. He wishes Forth had standard extensions to make such efforts unnecessary.
NGS FORTH
A FAST FORTH,
OPTIMIZED FOR THE IBM
PERSONAL COMPUTER AND
MS-DOS COMPATIBLES.

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

PRICES START AT $70

NEW: HP-150 & HP-110
VERSIONS AVAILABLE

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

---

0 \ ACCESS TO 80286 EXTENDED MEMORY UNDER MS-DOS.

2 Uses interrupt 15, function 87:

3 Swaps block of data between PC/AT's extended memory
4 (addresses above 1 MB), and DOS-accessible memory below 640K.
5 Based on James T. Smith, IBM PC AT Programmer's Guide (1986)
6 section 10.4: "Using extended memory."
7
8 Written in Uniforth implementation of Forth-83
9 (Unified Software Systems, PO Box 21294, Columbus, OH 43221)
10
11 Prof. Richard F. Olivo
12 Department of Biological Sciences
13 Smith College
14 Northampton, Massachusetts 01063
15

SCR # 0
0 \ WRITE GLOBAL DESCRIPTOR TABLE (GDT)
1 decimal
2 create GDT 48 allot \ Global descriptor table for INT15
3 4 : MAKE-GDT ( src-daddr dest-daddr #bytes -- )
5 \ daddr: uses 32 bit address, eg. source 00A0 0000
6 \ #bytes: bytes moved can be up to 64K (1 segment)
7 GDT 48 erase \ Fill entire table with 00
8 dup GDT 16 + ! \ Store source length, #bytes
9 GDT 24 + ! \ & destination length, #bytes
10 GDT 26 + 2! \ Store destination address
11 GDT 18 + 2! \ & source address
12 147 GDT 21 + c! \ Store access-rights byte $93,=147 dec
13 147 GDT 29 + c! ; \ in two places in GDT
14 --
15

SCR # 1
0 \ INTERRUPT 15 CODE: moves data, using GDT
1 hex
2 code INT15 ( gdt.addr segment #words -- status | 0=success)
3 3 CL, \ disable interrupts (time of day clock)
4 BX CX MOV, \ #words from top-of-stack (=BX) to CX
5 AX POE, \ GDT's (=dictionary's) segment
6 AX ES MOV, \ GDT segment into ES
7 SI DX MOV, \ save SI, Uniforth's interpreter pointer
8 AX POE, \ GDT's address (=offset in segment)
9 AX SI MOV, \ GDT offset into SI
10 00 # AL MOV, 87 # AH MOV, \ function # 87 to AH
11 15 INT, \ call interrupt 15
12 DX SI MOV, \ restore SI
13 AH BL MOV, \ put status code on tos
14 00 # BH MOV, \ clear high byte, tos; restore interrupts
15 STI, NEXT, END-CODE decimal --

---

Volume XI, Number 2 21
Forth Dimensions
SCR # 3
0 \ APPLICATION: Copy to/from video frame-grabber (DT2851)
1 \ CONSTANTS, ARRAYS & VARIABLES
2
3 \ Frame-grabber has 2 buffers, each 256KB, starting at address
4 \ 00A0 0000 in extended memory. We will access these one video
5 \ line at a time (512 bytes/line).
6
7 \ decimal
8 \ 512 constant PIXELS/LINE \ In Forth dictionary, make
9 \ create VIDLINE pixels/line allot \ array to hold 1 video line
10
11 \ GDT requires 32-bit addresses, =>double constants & variables
12 \ Absolute address of vidline array in Forth dictionary:
13 \ dseg vidline seg>adr \ Segment & offset of vidline array,
14 \ 2constant VIDLINE(DADR) \ converted to 32-bit(daddr) const.
15 \-

SCR # 4
0 \ CONSTANTS & VARIABLES: ADDRESSES FOR DT2851 IMAGE BUFFERS
1 \ hex
2 \ Absolute 32-bit addresses of DT2851 image buffers 0 & 1:
3 \ 00A0 0000 \ 2constant BUFFER0 \ daddr of buffer0
4 \ buffer0 4 0 d+ \ 2constant BUFFER1 \ daddr of buffer1
5 \ 2variable CURRENT-VLINE \ daddr of active video line
6
7 \ Automatic calculation of active buffer’s address:
8 \ (?out-buf returns 0 or 1 to indicate active output buffer)
9 \ CURRENT-BUFFER \ -- daddr \ \ active image buffer
10 \ ?out-buf 4 * 0 buffer0 d+ ;
11 \ OTHER-BUFFER \ -- daddr \ \ inactive image buffer
12 \ buffer1 ?out-buf 4 * 0 d- ;
13 \-
14 \-

SCR # 5
0 \ COPY ONE VIDEO LINE FROM FORTH DICTIONARY SPACE TO IMAGE
1 \ Requires previous calculation of current-vline (next screen).
2 \ decimal
3 \ DICT>VLINE \ \ Move one line up to image.
4 \ vidline(dadr) \ source: constant daddr of vidline
5 \ current-vline 2@ \ destination: prev calc’d daddr
6 \ pixels/line \ #bytes: pixels on one line
7 \ make-GDT \ write GDT
8 \ GDT dseg pixels/line 2/ \ gdt addr & segment, #words
9 \ int15 \ \ call int15
10 \ dup not if drop \ then test status: 0=ok, drop code
11 \ else ” Error:” . quit \ if error, print code, & quit
12 \ then ;
13 \-
14 \-

(Screens continued on page 28.)
SISIFOrth is an environment in which to write expert systems. It is not a program to make expert systems, but a toolkit with which to make them. Once SISIFOrth has been loaded, we can make use of a set of new words to define the rules, questions, and answers. I will try to explain how.

Before writing an expert system, of course, we have to design it. When that is done, we can build it using SISIFOrth. There are three steps on the way: define the questions, the rules, and at last the order of the rules and the answers they lead to.

“We can think of SISIFOrth rules as Prolog sentences.”

Each question of the expert system is stored in any line of any screen. To define a questions, the word QUESTION has to be used in this way:

sn ln QUESTION q_name

where sn is the screen number of the text of the question, ln is the line number on that screen where the question is found, and q_name is the question's name.

A rule is a number of conditions that, if all of them are true, leads to an answer. Each rule is just a word, defined in this way:

: rule_name
   (( question1 & question2 ));
There can be any number of questions inside a rule.

If the answer to any question must be false, we can add NOT after the name of the question:

```forth
: rule_name
  (( question1 & question2 NOT & question3 ))
```

It is possible to use AND or OR to link two or more questions:

```forth
: rule_name1
  (( question1 & question2 AND ))

: rule_name2
  (( question1 question2 OR ))
```

But there are some problems if we do this. If we use AND both questions will be asked of the user, even if the answer to question1 is false. Instead of AND, we should use the word & (the ampersand):

```forth
: rule_name1
  (( question1 & question2 ))

: rule_name2
  (( question1 question2 ))
```

Now if the answer to question1 is "not," question2 will not be asked. The OR case is not so important, because there is no difference between `rule_name2` above and the next two rules.

```forth
: rule_name2.1
  (( question1 ))

: rule_name2.2
  (( question2 ))
```

We have questions and rules; now we order the rules inside a Forth word (the main word of the expert system) in the following way:

```forth
: expert_system_name
  SISIFORTH_OK
  " Starting message"
  sn1 ln1 rule_name1
  sn2 ln2 rule_name2
  (etc....)
```
For each rule, $sn$ and $ln$ are the screen number and line number where the text of the answer has been typed. So, if the rule is true, line $ln$ of screen $sn$ will be printed and SISIForth will stop.

The order of the rules is very important, because SISIForth will stop whenever it finds a true rule. So, for example, if there are two rules like:

```forth
: rule1
  (( question1 & question2 & question3 ));

: rule2
  (( question1 & question3 ));
```

then rule2 should always be after rule1. If not, and if question1 and question3 are both true, SISIForth would stop in rule2 before examining rule1.

**SISIForth Words**

I think SISIForth will be easy to understand, because only a few words do all the work. The power of SISIForth is the way it does the dirty work; we have nothing to do but tell it the rules and where the text is located.

**QUESTIONS**

A constant that holds the maximum number of questions ever used. It has to be changed as needed.

**ANSWERS**

An array that holds the answers made by the user. Answers are stored as 0 for 'question not answered,' 1 for 'question answered false,' and 2 for 'question answered true.'

**N6**

Counter of how many times & is found within a rule, used to compile the correct number of ENDIFs at the end of the rule.

**NQ**

Counter of how many questions were created by QUESTION.

```forth
| 2DROP | ( ) ENDIF |
| :     | ) | ) |
| :     | COMPILE | ( ) |
| N6 & @ | ( &_no) |
| ?DUP | ( If &_no=0) ( &_no) |
| ( If &_no<>0) ( &_no, &_no) |
| IF | ( &_no, &_no, 0) |
| DO | [COMPILE] ENDIF |
| LOOP |
| ENDIF |
| : IMMEDIATE |
| : IMMEDIATE |
| Screen # 2 |
| 0 ( SISIForth, vehicles expert) |
| 1 ( Questions:) |
| 2 BLK+ 1 5 QUESTION MONEY |
| 3 BLK+ 1 6 QUESTION PASSENGERS |
| 4 BLK+ 1 7 QUESTION BAGGAGE |
| 5 BLK+ 1 8 QUESTION SPEED |
| 6 BLK+ 1 9 QUESTION SPORT |
| 7 ( Rules:) |
| 8 : SUBWAY (( MONEY NOT )) ; |
| 9 : CAR (( PASSENGERS BAGGAGE OR )) ; |
| 10 : MCYCLE (( SPEED )) ; |
| 11 : C5 (( SPORT NOT )) ; |
| 12 : BIKE (( SPORT )) ; |
| 13 --> |
| 14 |
| 15 |
| Screen # 3 |
| 0 ( SISIForth, vehicles expert, continued) |
| 1 : SISIFORTH_VEHICLES SISIFORTH OK |
| 2 ." Vehicles Expert System" CR BLK+ 0 11 SUBWAY BLK+ 0 12 |
| 3 CAR BLK+ 0 13 MCYCLE BLK+ 0 14 C5 BLK+ 0 15 BIKE ; !S |
| 4 ( Texts of questions:) |
| 5 Have you got money enough? |
| 6 Are you carrying passengers? |
| 7 Are you carrying baggage? |
| 8 Do you like speed? |
| 9 Do you like sport? |
| 10 ( Texts of answers:) |
| 11 Go to the subway, silly! |
| 12 Buy a car, sure. |
| 13 Buy a speedy motorcycle, then. |
| 14 Well, buy an electric Sinclair C5 tricycle! |
| 15 Better buy a bike, my friend. |
```
Prints titles.

**SISIFORTH_OK**

Clears ANSWERS and the display, to start execution of rules.

Y/N ( -- flag )

Waits for a "Y" or "N" to be typed, and leaves a flag.

**QUESTION** ( sn ln -- )

A word to create other words. Stores sn and ln at the PFA of the new word, with the content of NQ, and increments the content of NQ. When the new word is executed, it checks its flag in ANSWERS and, if it is 1 or 2 (which would mean the question has already been answered by the user), decrements it and leaves in on top of the stack. If the flag is 0, it calls Y/N to get the answer from the user. In any case, the word created by QUESTION leaves a flag (i.e., the answer) when executed.

(!

Resets the content of N& when a rule is compiled.

&

Increments the content of N& and compiles an IF.

() )

The shadow word compiled at the start of ). It checks to see if the rule is true; if so, it writes the answer and ends execution of SISIForth.

))

Compiles () ) and the needed ENDIIFS to complete the rule structure, using the value stored in N&.

**BLK+**

This makes the expert system source code relocatable among screens. Instead of using absolute screen numbers to indicate where an answer or questions has been typed, we can use BLK+ vs where vs stands for 'relative screen,' the offset from the actual compilation screen. Thus, BLK+ 0 denotes the screen currently being compiled, BLK+ 1 means the next screen, etc.

Conversion to fig-SISIForth

To convert this code to run on a fig-FORTH system, just add 0 before any
occurrence of the word VARIABLE. Then, in the word ), use -DUP in place of ?DUP. The word ABS at the end of the definition of Y/N could be deleted. In the definition of BLK+, use WORD HERE instead of simply WORD. If necessary, use <BUILDS instead of CREATE at the start of QUESTION.

Simple Examples

Screens two and three show a simple example. To get it working, just compile the screen where it starts, then type SISIFORTH_VEHICLES. You will have to answer some questions before SISIForth gives an answer. Try again, changing your answers, to see what happens.

I know this example is very simple, but if the way SISIForth works has been understood, it should be clear that very complex expert systems can be written in this way, not based on probabilities but "hard rules" expert systems.

Screens six through ten show another example. As can be seen, the only problem in building an expert system is designing it!
We can also think of SISIFOrth rules as Prolog sentences, where there are conditions to make the rule true. With that idea in mind, as you can see in the last example, it is as easy to write an intelligent data base using SISIFOrth as it would be using Prolog.

One more thing: if the order of the rules inside the main word is chosen with care, some questions of some rules can be saved. But it is better to write every question needed in every rule, it is surer. And the final message should not be needed if the rules have been thought out properly, but it is better to keep it there while debugging an expert system.

About the Name
In Greek mythology, Sisifo (Sisyphus in most English translations) was sentenced by Zeus to roll a big stone up a mountain; but when he arrived, the stone dropped back down and Sisifo had to go down and start again... and so on, again and again, forever. Do you understand?

Marcos Cruz joined the Forth Interest Group last year, and reports that the few Forth users he knows in Spain mostly use 68000-based computers such as the Sinclair QL, Commodore Amiga, and Atari ST. SISIFOrth was originally written for a Sinclair ZX Spectrum with a fig-FORTH from Melbourne House; that was improved upon and translated for a Sinclair QL running a Forth-83 from Computer One. The author welcomes mailed suggestions or improvements at Acacias 44, 28023 Madrid, Spain.

(Screens continued from page 22.)

```
SCR # 6
0 \ COPY VIDEO LINE FROM IMAGE TO FORTH DICTIONARY SPACE
1 2 : !CURREN-VLINE ( Y-val -- ) \ Calc. address in image.
2 ( For active image, calc 32-bit address of video line Y,)
3 ( and save the address in double variable.)
4 pixels/line uss*d \ unsigned mult w.32-bit prod: Y*px/l
5 current-buffer d+ \ added to start.addr = line address
6 current-vline 2! ; \ stored as daddr in variable.
7 9 : VLINE>DICT ( Y-val -- ) \ Copy one line (Y) down.
8 !current-vline current-vline 2@ \ source daddr to stack
9 vidline(dadr) pixels/line make-GDT + dest & #bytes ->GDT
10 GDT dseg pixels/line 2/ \ gdt addr & segment, #words
11 int15 \ dup not if drop \ call int15, test status
12 else ." Error:" . quit then ; \ if error, print code.
13 ->
```

```
SCR # 7
0 \ STORE & FETCH SINGLE PIXELS AT X,Y; FILL IMAGE WITH 1 VALUE
1 2 : !PIXEL ( n x y -- ) \ Replace pixel X,Y with value n.
2 vline>dict \ Bring video line Y from image,
3 vidline + c! \ address pixel X, modify it,
4 dict>vline ; \ and send line back to image.
5 6 : @PIXEL ( x y -- n ) \ Fetch value n of pixel X,Y.
7 vline>dict \ Y-coord specifies one line;
8 vidline + c@ ; \ address pixel X, find value.
9 10 : VFILL ( n -- ) \ Fill entire image with byte n.
11 freeze \ End live video, hold image
12 vidline pixels/line rot fill \ set up video line
13 480 0 do i !current-vline \ address 480 lines of image
14 480 0 loop ; -> \ & copy same array into each.
```

```
SCR # 8
0 \ SAVE AN IMAGE TO DISK FILE
1 decimal
2 : HANDLE# ( -- n ) fcb @ c@ ; \ Get current file handle.
3 4 : IMAGE>DISK ( -- ) \ Move video image to an open file.
5 480 0 do \ For 480 lines of a video image,
6 i vline>dict \ move each line from image to array,
7 vidline 512 handle# \ set address, count & file-handle
8 write-bytes drop \ write array to disk, drop status,
9 loop ; \ and do next line.
10 11 : SAVE ( filename ) ( -- ) \ Save image in new disk file.
12 close freeze \ Close any open file, freeze image;
13 ." Saving image " \ tell user something is happening;
14 make ( filename ) \ open a file using filename supplied
15 image>disk close ; -> \ copy image, and close file.
```

(Screens continued on page 34.)
SC/FOX™ Forth Optimized eXpress™

SC/FOX PCS Parallel Coprocessor System
Uses Harris RTX 2000™ real-time Forth CPU.
System speeds options: 8 or 10 MHz.
Full-length 8 or 16-bit PC/XT/AT plug-in board.
32K to 1M bytes, 0-wait-state static RAM.
Hardware expansion, two 50-pin strip headers.
Multiple PCS board parallel operation.
Data transfer through movable shared 16K window.
Includes FCompiler, SC/Forth optional.
Prices start at $1,995 with software.

SC/FOX SBC Single Board Computer
Uses RTX 2000 real-time Forth CPU.
System speed options: 8, 10, or 12 MHz.
32K to 512K bytes 0-wait-state static RAM.
64K bytes of shadow-EPROM space.
RS232 serial and Centronic printer ports.
Hardware expansion, two 50-pin strip headers.
Eurocard size: 100mm by 160mm.
Includes FCompiler, optional SC/Forth EPROM.
Prices start at $995 with software.

SC/FOX SCSI I/O Daughter Board -- NEW!
Plug-on daughter board for SC/FOX PCS and SBC.
Source s/w drivers for FCompiler and SC/Forth.
SCSI adaptor with 5 Mbytes/sec synchronous or
3 Mbytes/sec asynchronous transfer rates.
Floppy disk adaptor, up to 4 drives, any type.
Full RS-232C Serial Port, 50 to 56K Baud.
16-bit bidirectional, latching-parallel port.
Price $695 with software.

SC/Forth™ Language
Interactive Forth-83 Standard.
15-priority time-sliced multitasking.
Supports user-defined PAUSE.
Automatic optimization and μcode support.
Turnkey application support.
Extended structures and case statement.
Double number extensions.
Infix equation notation option.
Block or text file interpretation.
Optional source code developers system.
Prices start at $695.

Harris RTX 2000 Real-Time Forth CPU
1-cycle multiplier, 14-prioritized interrupts,
one NMI, two 256-word stacks, 16-bit timers/counters,
and an 8-channel multiplexed 16-bit I/O bus.

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 Avenue, Suite K, Palo Alto, CA 94306 (415) 322-8763
The Forth assembler is handy for writing short pieces of code to access the hardware or to speed up a loop, but it can be cumbersome for large routines. A full-fledged macro assembler with all the bells and whistles is ideal for larger routines, but it would be very messy to implement as a Forth vocabulary. Fortunately, this is not necessary. It is possible to write a routine using a regular macro assembler, then to incorporate the resultant binary file as a Forth word. I will describe a way to do this for a PC-type segmented memory structure. Forth-79 is used, along with the Microsoft Macro Assembler under MS-DOS.

"This makes practical a library of binary routines..."

The assembler routine to be adopted into Forth must be written or modified to conform to certain restrictions. It must be in the COM format, with all code and data in the same segment. The instruction ORG 100h—usually included in a COM program—is not used, because we want the routine to begin at offset zero within its segment. (Alternately, keep the ORG 100h instruction and subtract 10h from the segment value in the Forth defining word.) A stack segment is not defined, and SS is not used in the ASSUME statement. Input and output parameters will be passed via the Forth stack. When the routine executes, the top two elements on the stack will be the segment and offset of the Forth re-entry point.

```
SCR #1
0 \ Two Assemblers Are Better than One D. C. Olivier 3-6-89
1 2 VARIABLE SIZE \ Size of file to be loaded
2 VARIABLE BINSSEG \ Segment
3 VARIABLE NAMES 11 ALLOT \ File name string
4 DVAR VABLE REGS \ Holds register contents
5 6 CODE CSSEG \ Leaves contents of the CS register on the stack.
7 CS PUSH
8 NEXT JMP END-CODE

SCR #2
0 \ Two Assemblers Are Better than One D. C. Olivier 3-6-89
1 CODE CALLBIN
2 \ REGS, BP MOV \ Save registers
3 \ REGS 2+, SI MOV \ used by Forth.
4 203 C, \ Intersegment return instruction for 8088
5 END-CODE
6
7 CODE REENTRY
8 AX, CS MOV
9 DS, AX MOV \ Restore DS register
10 BX, # REGS MOV \ Restore registers
11 BP, [BX] MOV \ used by Forth
12 BX INC BX INC \ used by Forth
13 SI, [BX] MOV \ 
14 NEXT JMP END-CODE

SCR #3
0 \ Two Assemblers Are Better than One D. C. Olivier 3-6-89
1 2 : READBIN ..... ;
3
4 EXIT
5
6 The name of the binary file is at NAMES as a counted string.
7 Use your own DOS file interface to open the file, determine and
8 store its size in SIZE, read the file from disk to BINSSEG:0,
9 close the file, do any appropriate error checking.

SCR #4
0 \ Two Assemblers Are Better than One D. C. Olivier 3-6-89
1 2 : BINFIL E \ Saves input stream pointer
2 \ IN @ \ Reads next word from input stream
3 BL WORD DUP \ CSEG 1+ NAMES SWAP CMOVE \ Moves word to variable NAMES
5 \ IN 1 \ Restores input stream pointer
6 CREATE \ Reads same word, creates header
7 HERE 16 / 1+ DUP \ # of paragraphs to HERE, + 1
8 CSEG + BINSSEG + \ Segment for binary routine
9 READBIN \ Read file from disk to BINSSEG:0
10 16 * SIZE 0 + DP \ \ Allot dictionary space
11 DOES> \ PPA on stack at run time
12 CSEG \ REENTRY \ Segment and offset of reentry point
13 ROT 16 / 1+ CSEG + 0 \ Segment and offset of binary routine
14 CALLBIN ; \ Intersegment RET instruction
```
point. These should be saved in variables and restored before the final RET instruction. Define the routine as a FAR procedure, so that the final RET instruction will be assembled as an intersegment return.

Figure One is a trivial example that takes two numbers from the stack, adds them, and places the result on the stack before returning to Forth. Assemble the source code and LINK as usual; use the EXE2BIN utility to convert to binary format, then rename the file if you like. Its Forth name will be the same as its filename.

BINFILE in screen four is the defining word. The syntax is:

BINFILE <name>

where <name> is the name of the binary file on disk. On line two, we save the value of the input stream pointer on the stack. On line three, we read the next word from the input stream, which is <name>. Line four moves <name> to a variable. Line five restores the original value of the input stream pointer so that CREATE can read <name> also. Line six creates a dictionary header for <name>.

The binary routine must start on the first paragraph boundary within the parameter field. (A paragraph boundary is any address evenly divisible by 16.) On line seven, we calculate the number of paragraphs from the beginning of the Forth segment to the parameter field. Line eight adds the value in the CS register to this number and places the result in the variable BINSEG. On line nine, READBIN reads the file whose name is in NAMES from disk to BINSEG:0. Line ten allot dictionary space for the definition.

Line 12 begins the run-time part of the defining word. It places on the stack the address of the Forth re-entry point to be used by the final RET instruction in the binary routine. Line 13 places the address of the beginning of the binary routine on the stack. At first glance it might seem that this is simply BINSEG:0, but if the Forth program has been compiled as a turnkey system, it may be executed from a different segment that the one in which it was compiled. That is why this address must be calculated at run time. Line 14 executes an intersegment RET instruction, which takes the segment and offset from the stack and jumps to it.

This technique makes practical a commercial or public-domain library of binary routines to be incorporated into Forth programs. These could be sorts, searches, graphic routines, math functions, transforms, etc. Of course, such a library would not be portable across CPUs, but it would be portable across Forth dialects. Any implementation dependencies would be hidden in the defining word BINFILE. Such a library could greatly increase the productivity of Forth programmers. Why spend time coding a quicksort routine in Forth, for example, when you could buy a canned assembly language version that had been optimized, tested, and debugged? It is even conceivable that, for some applications, Forth could be used as "glue" to hold together packaged routines that did most of the work.

This is a significant example of how the defining word BINFILE allows the combination of Forth and assembly language routines. The defining word BINFILE makes it possible to link a Forth program to an assembly language routine, and vice versa. The defining word BINFILE allows the Forth programmer to call an assembly language routine, and the assembly language programmer to call a Forth routine. This is a powerful feature that allows Forth and assembly language to work together in a complementary way.

Figure One. An assembly routine can perform its task and return to Forth.

```
TITLE PLUS
;
ASEG SEGMENT PARA PUBLIC
ASSUME CS:ASEG,DS:ASEG,ES:ASEG
;
START:
JMP ADDUP
RET_ADDR1 DW ?
RET_ADDR2 DW ?
ADDUP PROC FAR
;
; Code and data in same segment.
MOV AX,CS
MOV DS,AX
;
; Save the Forth re-entry point.
POP RET_ADDR1
POP RET_ADDR2
;
; Perform the operation.
POP AX ; Get arguments from stack
POP BX
ADD AX,BX ; Add them.
PUSH AX ; Put result on the stack.
;
; Put the segment and offset of the
; Forth re-entry point on the stack.
PUSH RET_ADDR2
PUSH RET_ADDR1
;
; Jump to the re-entry point.
RET
ADDUP ENDP
ASEG ENDS
END START
```
As promised in the last issue, I will continue with recent on-line conferences that featured George Shaw, Mike Perry, Randy Dumse, and Wil Baden. As before, I will feature the guests' opening remarks from their respective conferences. These remarks set the tone and direction of the conference, and they serve that purpose well. I hope most readers will note they also serve to acquaint the attendees of the conference—and now the readers of this column—with the guests' personal philosophies.

This is no accident. When I have approached a prospective guest, I have always asked what they wish to talk about. What is their personal point of wisdom they wish to share? Without exception, those who have accepted the invitation have also accepted the opportunity to share their personal point of view, as opposed to some general subject. It is clear that we all benefit from this unselfishness. I again wish to thank all these gracious people for sharing their insight as they have.

Now, on to the recaps.

George Shaw
December 1988
Owner of Shaw Laboratories.

<[George]> We (myself and others) started the ACM SIG to bring Forth into the professional computing arena. ACM is very visible in the universities and colleges, and is very well respected around the world. We felt that having a SIG would give Forth a large boost in image as a language for serious use by professionals.

Thanks to Alan Furman for starting the whole thing and analyzing the situation to give us direction and goals. I have a list of our initial projects:

Education: Moving Forth into the universities and colleges to create an awareness of Forth and a better supply of Forth programmers.

Forthics: Research to create a basic set of Forth programming ethics as well as management metrics to increase the success of Forth projects.

Market: A study of the job requirements and the Forth programming skills available to determine trends in the Forth job marketplace and skills required for the future.

"The immediacy is lost, but the words remain."

Successes/Failures: A study of the historical applications of Forth to create an awareness of Forth’s widespread use and to determine what the characteristics and causes are for successes and failures.

ANS Forth: Participate through your SIGForth representative in the ANSI FORTH committee to produce an American National Standard for Forth that everyone can use.

State of the Industry: SIGForth periodically compiles surveys of the Forth industry to evaluate the status of the Forth product market, job market, workplace, education availability, job requirements, etc. Participate in these surveys and be one of the first to reap their benefits through their publication in the SIGForth newsletter.

Mike Perry
January 1989
Owner of Even-Odd Designs.

<[mike]> I have benefitted enormously from the work of many other people. I have come to believe strongly in the value of open systems. I want, and need, to have complete control over my tools. Sharing code and avoiding secrets are essential for productivity.

I am convinced that Forth will continue to be interesting because so many new techniques and ideas are explored here; that is possible, in large part, because we share our code and ideas.

Remember, we will only lose control of our machines if we give it away.

Randy Dumse
February 1989
Owner of New Micros, Inc.

<[DUMSE]> Being somewhat isolated here in Texas has its disadvantages. The availability of other informed people to bounce ideas off is limited, so most of my opportunities for such interaction occur at most twice a year: at FORML or Rochester.

On the other hand, not having anyone to give guided direction to your thinking can allow original thought to take some interesting directions. How useful these thoughts are often cannot be determined by the originator. It’s a little like the male complex where no baby is ever pretty—until it’s his own!

So it is with ideas: they are much like the only child a male can bear, and therefore look pretty darn cute to Dada. It can be a bit hard to be objective when there is that feeling of self-investment in the thoughts.

(Continued on page 34.)
Forth Interest Group

The Forth Interest Group serves both expert and novice members with its network of chapters, *Forth Dimensions*, and conferences that regularly attract participants from around the world. For membership information, or to reserve advertising space, contact the administrative offices:

Forth Interest Group
P.O. Box 8231
San Jose, California 95155
408-277-0668

Board of Directors
Robert Reiling, President *(ret. director)*
Dennis Ruffer, Vice-President
John D. Hall, Treasurer
Terri Sutton, Secretary
Wil Baden
Jack Brown
Mike Elola
Robert L. Smith

Founding Directors
William Ragsdale
Kim Harris
Dave Boulton
Dave Kilbridge
John James

In Recognition

Recognition is offered annually to a person who has made an outstanding contribution in support of Forth and the Forth Interest Group. The individual is nominated and selected by previous recipients of the "FIGGY." Each receives an engraved award, and is named on a plaque in the administrative offices.

1979 William Ragsdale
1980 Kim Harris
1981 Dave Kilbridge
1982 Roy Martens
1983 John D. Hall
1984 Robert Reiling
1985 Thea Martin
1986 C.H. Ting
1987 Marlin Ouverson
1988 Dennis Ruffer

On-Line Resources

To communicate with these systems, set your modem and communication software to 300/1200/2400 baud with eight bits, no parity, and one stop bit, unless noted otherwise. GEnie requires local echo.

**GEnie**
For information, call 800-638-9636
- Forth RoundTable *(ForthNet link)*
  - Call GEnie local node, then type M710 or FORTH
  - SysOps: Dennis Ruffer (D.RUFFER), Scott Squires (S.W.SQUIRES), Leona Morgenstern (NMORGENSTERN), Gary Smith (GARY-S)
- MACH2 RoundTable
  - Type M450 or MACH2
  - Palo Alto Shipping Company
  - SysOp: Waymen Askey (D.MILEY)

**BIX (ByteNet)**
For information, call 800-227-2983
- Forth Conference
  - Access BIX via TymeNet, then type j forth
  - Type FORTH at the : prompt
  - SysOp: Phil Wasson (PWASSON)
- LMI Conference
  - Type LMI at the : prompt
  - Laboratory Microsystems products
  - Host: Ray Duncan (RDUNCAN)

**CompuServe**
For information, call 800-848-8990
- Creative Solutions Conference
  - Type !Go FORTH
- Computer Language Magazine Conference
  - Type !Go CLM
  - SysOps: Jim Kyle, Jeff Brenton, Chip Rabinowitz, Regina Starr Ridley

**Unix BBS's with Forth conferences (ForthNet links)**
- WELL Forth conference
  - Access WELL via CompuserveNet or 415-332-6106
  - Fairwitness: Jack Woehr (jx)
- Wetware Forth conference
  - 415-753-5265
  - Fairwitness: Gary Smith (gars)

**PC Board BBS's devoted to Forth (ForthNet links)**
- East Coast Forth Board
  - 703-442-8695
  - SysOp: Jerry Schifrin
- British Columbia Forth Board
  - 604-434-5866
  - SysOp: Jack Brown
- Real-Time Control Forth Board
  - 303-278-0364
  - SysOp: Jack Woehr
Other Forth-specific BBS's
• Laboratory Microsystems, Inc.
  213-306-3530
  SysOp: Ron Braithwaite
This list was accurate as of March 1989. If you know another on-line Forth resource, please let me know so it can be included in this list. I can be reached in the following ways:
Gary Smith
P. O. Drawer 7680

Useful domain: gars@well.UUCP or gars@wet.UUCP
Internet: well!gars@lll-winken.arpa
WELL: gars

+ForthNet is a virtual Forth network that

---

There isn’t even a mother on which to blame half of the genes.

With those thoughts, I begin.

Forth as a Standalone Operating System
Both the R65F11 and F68HC11 single-chip computers have been designed as standalone computer systems. They use Forth as their operating system. In this regard, they follow in the tradition of micros like the KIM-1, SYM-1, and the AIM-65. Each of these had a built-in monitor to allow user interaction with the system. Similarly, other systems used BASIC as their power-on operating system, such as the (if my memory serves) OSI, TRS-80, and Apple.

Something to keep in mind: “operating system” hasn’t always meant “disk operating system.”

As most of us have heard, Forth is nearly its own operating system. In fact, it is often stated that Forth is difficult to install under an existing operating system because it is not well behaved. These comments really have nothing to do with Forth as a general language, but come out of the difficulty of doing blocks under another OS.

Wil Baden
March 1989
Owner of Paleotaurus, Inc.

<Will> Let’s stop kidding ourselves. Forth deserves its reputation as a write-only language. 99 percent of published Forth programs prove this. Until Forth improves its reputation, it will be scorned. Tonight I want to investigate the evil forces that cause this and discuss six necessary but insufficient rules for more readable programs.

1. The stack state must be given for every line.
2. Formatting must show logical structure.
3. Use short definitions, short lines, short phrases.
4. Don’t mumble—your program should pass your spelling checker.
5. Mix upper and lower case—all lower case is just as bad as all upper.
6. Write comments in the English language.

Are there better rules? What else must be done?

* * * *

If you are grinding your teeth and wishing you had participated in one of these terrific guest conferences, all is not lost. The immediacy is lost forever, but the words remain for you to capture and study at your leisure. They are posted in the GEnie FortTable, Library 1.

---

SCREEN continued from page 28.

SCR # 9
0 \ RESTORE AN IMAGE FROM DISK FILE
1 decimal
2
3 : DISK>IMAGE ( -- ) \ Copy an open disk file to video,
4 480 0 do \ For 480 lines of a video image,
5 vidline 512 handle# \ set up array's address, size, file;
6 read-bytes drop \ load array from file, drop status;
7 i !current-vline \ find address in image of this line,
8 dict>vline loop ; \ and move the line to image buffer.
9
10 : RESTORE ( filename ) ( -- ) \ Open & load file to video.
11 freeze open ( filename ) \ Stop live video, open file;
12 disk>image close ; \ copy file, close it.
such as 00 3B for function key one). Using this trivial definition makes it much easier to avoid such problems as the user pressing an extended key after “Press any key ...”

%QUIT in screen 33, in conjunction with the main editor loop (E) in 57, shows how to force an exit, regardless how many levels of calling words (return addresses on the return stack) there are.

In the mainline word (E), we note the position of the return stack (RP$), which we save in our variable &RP0. This notes the point on the return stack with the address to which we ultimately want to exit.

Then, when a word like ABANDON or EXIT-SAVE wants to exit PDE, it uses %QUIT to reset RP (&RP0 @ RP!) and then returns to that higher-level return address. This approach is more general than a series of R> DROPS, particularly when you consider that ABANDON is nested four levels deep when invoked through ^Z, but only three when Alt-F1 is used.

Footnotes
1. “Screen-Oriented Editor in Forth,” by Henry Laxen. Dr. Dobb’s Journal, vol. 6, no. 9, pg. 27.
2. VED (Craig Lindley) and FSED (Gene Czarcinski), September 1986. Credit also to John A. Peters and R.F. Buchanan.
3. F83 is a public-domain implementation of Forth-83 by Henry Laxen and Michael Perry, with many, many fine added features. F83 is available on disks from SIGM (154) or, better yet, from Laxen and Perry’s No Visible Support Software for $25. Go ahead, make their day.
4. “Fast SEARCH for F83,” by Bill Zimmerly. Forth Dimensions VIII/4, pg. 5. [Also IX/2,4, and XI/1.—Ed.]

Robert Garian is a technical information and language specialist at the Library of Congress, specializing in Soviet computing. He has written an AI program called Block Solver that rearranges one multistack configuration of blocks into a specified goal configuration under constraints; and he has worked on automating both software verification and code generation. His current interests include simulation of complex systems, cellular automata, and genetic algorithms.

(Notes, continued from page 5.)

The equivalent (unoptimized) BASICA program for the explicit oval function might be written as shown in Figure Five.

If you try both approaches, you will find the latter explicit method unsatisfactory for drawing ovals without some modifications for angles close to the horizontal axis. The OVALS.HSF demo can be downloaded from the GENie Forth RoundTable and from the East Coast Forth Board [see Reference Section.—Ed.].

The Forth community is better served by first offering algorithms expressed in Forth so the explanations that accompany the code can be more readily understood. Once care is taken to provide such information, assembly-language implementations might be appreciated.

I hope information about object-oriented languages piles up for Forth Dimensions. Aren’t Forth programmers more likely to appreciate creative approaches to the problem of programming computers? Before C++ takes off, perhaps FORTH++ can step in.

Mike Elola
San Jose, California

Frans van Duinen is regional manager of Micro-expertise Inc., a custom software house that specializes in networked database systems. A slightly more recent version of the code may be downloaded from GENie and from Canada Remote System (416-629-0136) as PDE202.ARC.
Here in the high-altitude desert that is Colorado on the eastern side of the Rockies, it is drizzling a drizzle that would do credit to the Pacific Northwest. The difference is that when the spring soddenness arrives in the Cascades, the air is heavy with a primeval green scent, whereas here in Golden, at the foot of the Foothills, there is the tangy aroma of fields of damp straw.

Soon the bull snakes will hatch and warm their coppery beauty in the sunshine of early June. Already it has become a questionable enterprise to climb North Table Mountain; unseasonable eighty-degree weather in late April has the rattlers already emerging from their hibernation to bedevil suburbanites engaged in lawn care in the upscale development injudiciously located on the side of that prominence.

Colorado is like the bull snake fresh from the shell, a coiled potency awaiting exercise. The end of the petroleum boom left many high and dry economically. Last year, mortgage foreclosures surpassed new mortgages for the first time. There is a sense of lack of permanence among the high-tech employed. Miniscribe, for instance, formerly the Boulder-area wunderkind, has been steadily cutting back.

Yet the improbable obtains: Colorado is a hotbed of Forth. Ball Aerospace is here, those arbiters of the final configuration of the RTX-2000. The red brick walls of IBM's city-sized fortress in Longmont reputedly conceal several ongoing Forth efforts. Applied Energy in Ft. Collins has periodically gone to great lengths to obtain qualified Forth assistance. Charles Johansen is working on finishing and foundering an inexpensive Forth chip while Cliff King, president of Denver FIG, is in the process of releasing the first revision of his 32-bit AT&T DSP-chip Forth development system.

The local chapters of the Forth Interest Group have been only partially successful in tying together the disparate practitioners of Forth in a functioning fraternity. Boulder Forth Interest Group seems to have disappeared, its members swallowed but not digested by Denver FIG. The latter organization meets sporadically, usually when a speaker is in town. Since the meeting place moved to Golden, between Denver and Boulder, we have been more successful in "trapping" members of both communities at meetings.

"This small event defies the mortality statistics."

Our most successful recent meeting was at the National Institute of Standards and Technology at the Commerce Department facility in Boulder. We gathered from all over the state to see Dr. Jeffrey Fox demonstrate software he wrote in Forth on various architectures to demonstrate chaotic systems and the Monte Carlo method.

Subsequent to that meeting, we had another well-attended meeting in which we agreed to really get Denver FIG going again. There hasn't been a formal Denver FIG meeting since. Are we unusual?

Our Forth-83 class still meets once a week. Three to seven people attend to learn, gripe, bring incomplete and ailing Forth projects for the Forth Doctor to diagnose, show off fancy new toys such as the MC68HC11 with New Micro's Forth on board. Members will indeed gather if they can sense some purpose in doing so. In the case of the Forth class, for instance, some members' purpose is to get consultation help for $2.50 an hour that they used to pay $30 an hour to obtain. Isn't FIG for helping people not only with Forth in the abstract, but also with Forth in their specific application? In any event, this small weekly event has defied the mortality statistics of local chapter activity to run for about a year now, with only two blizzards and one Florida vacation having interrupted its continuity.

They still talk about the time three years back when Charles Moore came to town to show off the Novix [Forth chip]. One gets the sense that, more than anything, our chapter awaits the founding of a FIG Speaker's Bureau that would track Forthers willing to address local groups and to publish their travel itineraries for the benefit of interested chapters.

Speakers at FIG meetings do not have to be celebrities; a new face would be worth ten clever newsletters, in terms of drawing a crowd at Denver FIG. Fortunately for the continued vitality of our chapter, Gary Betts of the ANSI Forth Technical Committee (X3J14) is in the area and has agreed to address us. Charles Curley should be out for a visit sometime soon, and when Wil Baden comes to Colorado around January of next year there will be an eager and attentive audience awaiting his presentation. Any other Forthers visiting Colorado are invited to address the chapter on their doings; please give us some advance notice.
Our chapter bulletin board, the RCFB, is now on the ForthNet. ForthNet messages bounce around the continent from local BBS to local BBS, sometimes even coming back to reinsert themselves accidentally in the conversational threads of the board of origin. The Forth community has never had such a communications resource at its fingertips before. What is needed now is some creative use of same.

Wouldn’t it be nice if every chapter had a BBS on the ForthNet? In such a situation, no formal Speaker’s Bureau would be necessary. There is already a FIG Conference that is exchanged on the ForthNet. (Currently we are discussing just what is a FIG Chapter. You should log in just to see British Columbia FIG’s electronic newsletter!) Forthers can post their travel dates in the ForthNet FIG Conference, or in a separate Speaker’s Conference. We might actually get to meet one another. Hibernating chapters might have cause to dust off the gavel more often than quarterly.

But these are dreams one has only on rainy days...

(Code continued from page 9.)

```
DOZEN-EGGS 35 IS R1 CR CR % Assumes R1 has been preset.
WIPE." Point height is increasing. Equatorial radius is " R1.
6 IS #COLS
2 IS #ROWS
R1 2* 10 + IS HJUMP
200 #ROWS / 2/ 10 + IS VJUMP
#COLS 1+ 1 DO
#ROWS 1+ 1 DO
HJUMP J * IS D
VJUMP I * IS C
FI J * IS R2
F2 I * IS R3
OVAL

LOOP
LOOP ORIGIN;
CASE: COMMANDS DOZEN-EGGS NEST-OF-EGGS ONE-EGG BYE ;CASE
: DEMO ( -- )
BEGIN
CR ." 1 -- Dozen eggs"
CR ." 2 -- Nest of eggs"
CR ." 3 -- Draw an oval"
CR ." 4 -- Quit"
CR ." Your choice: " #IN 1- ABS 3 MIN 0 MAX
COMMANDS CR ." Press any key " KEY DROP WIPE
AGAIN ;

( Enter DEMO and a carriage return )
\------------------
16 1024 * CONSTANT 16K
16K 10 + SEGMENT PICTURE \ set up a buffer
: CLRBUF
PICTURE DUP @ 0 OVER 21@ 32 FILLLL
0 PICTURE 41! ;

\ Take a snapshot of screen. Save it in TEMP.PIC
: SNAP ( -- )
16K PICTURE 41! \ make sure entire screen is saved
C-OFF \ cursor off
CRT-BASE 0 PICTURE @ 0 16K CMOVE
\ SCREEN to PICTURE segment
PICTURE $" TEMP.PIC" PUT-FILE C-2 ; \ PICT. to TEMP.PIC file, cursor on
: QD PICTURE @ 0 CRT-BASE 0 16K CMOVE ; \ PICTURE to SCREEN
\ Look at snapshot kept in TEMP.PIC.
: LOOK ( -- )
CLRBUF
PICTURE $" TEMP.PIC" GET-FILE \ TEMP.PIC to PICTURE
QD ;

\ Illustration
: DEMO2 NEST-OF-EGGS SNAP WIPE CR ." Screen saved."
CR ." Press any key to display screen stored in file" KEY DROP LOOK ;
```
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
    - Ric Shepard
    - Box 1344
    - Kodiak, Alaska 99615
- **ARIZONA**
  - Phoenix Chapter
    - 4th Thur., 7:30 p.m.
    - AZ State University
    - Memorial Union, 2nd floor
    - Dennis L. Wilson
    - (602) 956-7578
- **ARKANSAS**
  - 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. Grevelle Ave.
    - Phillip Wasson
    - (213) 649-1428
  - North Bay Chapter
    - 2nd Sat., 10 a.m.
    - 12 Noon Tutorial, 1 p.m.
    - South Berkeley Public Library
    - George Shaw (415) 276-5953
  - Orange County Chapter
    - 4th Wed., 7 p.m.
    - Fullerton Savings
    - Huntington Beach
    - Noshir Jesung (714) 842-3032
  - Sacramento Chapter
    - 4th Wed., 7 p.m.
    - 1708-59th St., Room A
    - Tom Ghormley
    - (916) 327-2047
  - San Diego Chapter
    - Thursdays, 12 Noon
    - Guy Kelly (619) 454-1307
  - Silicon Valley Chapter
    - 4th Sat., 10 a.m.
    - H-P Cupertino
    - Bob Barr (408) 435-1616
  - Stockton Chapter
    - Doug Dillon (209) 931-2448
- **COLORADO**
  - Denver Chapter
    - 1st Mon., 7 p.m.
    - Clifford King (303) 693-3413
  - Central Connecticut Chapter
    - Charles Krajewski
    - (203) 344-9996
- **FLORIDA**
  - Orlando Chapter
    - Every other Wed., 8 p.m.
    - Herman B. Gibson
    - (305) 855-4790
  - Southeast Florida Chapter
    - Coconut Grove Area
    - John Forsberg (305) 252-0108
  - Tampa Bay Chapter
    - 1st Wed., 7:30 p.m.
    - Terry McNay (813) 725-1245
- **GEORGIA**
  - Atlanta Chapter
    - 3rd Tues., 6:30 p.m.
    - Western Sizzlen, Doraville
    - Nick Hennenfent
    - (404) 393-3010
- **ILLINOIS**
  - Cache Forth Chapter
    - Oak Park
    - Clyde W. Phillips, Jr.
    - (312) 386-3147
  - Central Illinois Chapter
    - Robert Illyes
    - (217) 359-6039
- **MARYLAND**
  - MDFIG
    - Michael Nemeth
    - (301) 262-8140
- **MASSACHUSETTS**
  - Boston Chapter
    - 3rd Wed., 7 p.m.
    - Honeywell
    - 300 Concord, Billerica
    - Gary Chanson (617) 527-7206
- **MICHIGAN**
  - Detroit/Ann Arbor Area
    - 4th Thurs.
    - Tom Chrapkiewicz
    - (313) 322-7862
- **MINNESOTA**
  - MNFIG Chapter
    - Minneapolis
    - (612) 588-9532
- **MISSOURI**
  - Kansas City Chapter
    - Robert Illyes (217) 359-6039
  - Fort Wayne Chapter
    - 2nd Tues., 7 p.m.
    - IP Univ. Campus, B71 Neff Hall
    - Blair MacDermid
    - (219) 749-2042
  - Central Iowa FIG Chapter
    - Robert Illyes (217) 359-6039
  - Fort Wayne Chapter
    - 2nd Tues., 7 p.m.
    - IP Univ. Campus, B71 Neff Hall
    - Blair MacDermid
    - (219) 749-2042
- **NEW JERSEY**
  - New Jersey Chapter
    - Rutgers Univ., Piscataway
    - Nicholas Lordi
    - (201) 338-9363
  - Fairfield FIG Chapter
    - 4th Day, 8:15 p.m.
    - Gurdy Leete (515) 472-7077
- **FAIRFIELD FIG Chapter**
  - 4th Day, 8:15 p.m.
  - Gurdy Leete (515) 472-7077
- **FLORIDA**
  - Orlando Chapter
    - Every other Wed., 8 p.m.
    - Herman B. Gibson
    - (305) 855-4790
  - Southeast Florida Chapter
    - Coconut Grove Area
    - John Forsberg (305) 252-0108
  - Tampa Bay Chapter
    - 1st Wed., 7:30 p.m.
    - Terry McNay (813) 725-1245
- **GEORGIA**
  - Atlanta Chapter
    - 3rd Tues., 6:30 p.m.
    - Western Sizzlen, Doraville
    - Nick Hennenfent
    - (404) 393-3010
- **ILLINOIS**
  - Cache Forth Chapter
    - Oak Park
    - Clyde W. Phillips, Jr.
    - (312) 386-3147
  - Central Illinois Chapter
    - Robert Illyes
    - (217) 359-6039
- **MARYLAND**
  - MDFIG
    - Michael Nemeth
    - (301) 262-8140
- **MASSACHUSETTS**
  - Boston Chapter
    - 3rd Wed., 7 p.m.
    - Honeywell
    - 300 Concord, Billerica
    - Gary Chanson (617) 527-7206
- **MICHIGAN**
  - Detroit/Ann Arbor Area
    - 4th Thurs.
    - Tom Chrapkiewicz
    - (313) 322-7862
- **MINNESOTA**
  - MNFIG Chapter
    - Minneapolis
    - (612) 588-9532
- **MISSOURI**
  - Kansas City Chapter
    - Robert Illyes (217) 359-6039
  - Fort Wayne Chapter
    - 2nd Tues., 7 p.m.
    - IP Univ. Campus, B71 Neff Hall
    - Blair MacDermid
    - (219) 749-2042
  - Central Iowa FIG Chapter
    - Robert Illyes (217) 359-6039
  - Fort Wayne Chapter
    - 2nd Tues., 7 p.m.
    - IP Univ. Campus, B71 Neff Hall
    - Blair MacDermid
    - (219) 749-2042
  - **NEW JERSEY**
    - Rutgers Univ., Piscataway
    - Nicholas Lordi
    - (201) 338-9363
Volume XI, Number 2 39 Forth Dimensions

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

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

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

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

Columbus FIG Chapter
4th Tues.
Kal-Kan Foods, Inc.
5115 Fisher Road
Terry Webb
(614) 878-7241

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

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

PENNSYLVANIA
Villanova Univ. FIG Chapter
Bryan Stueben
321-C Willowbrook Drive
Jeffersonville, PA 19403
(215) 265-3832

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

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
Cliff Penn (214) 995-2361

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

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

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

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

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

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

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

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

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

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

CANADA
BC FIG
1st Thurs., 7:30 p.m.
BCIT, 7300 Willingdon Ave.
BBY, Rm. 1A-324
Jack W. Brown (604) 596-9764
BBS (604) 434-5886

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

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

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

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

FINLAND
FinFIG
Janne Kotiranta
Arkkitehdinkatu 38 c 39
33720 Tampere
+358-31-184246

HOLLAND
Holland Chapter
Vin Van de Zande
Finmark 7
5831 JE Leusden

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

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

NORWAY
Bergen Chapter
Kjell Birger Faeraas
47-518-7784

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

SWEDEN
SweFIG
Per Alm
46/8-92963

SWITZERLAND
Swiss Chapter
Max Hugelshofer
Industrieberatung
Ziberstrasse 6
8152 Opikon
01 810 9289

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

SC/FOX SBC
Single Board Computer
User Manual

SC/FOX PCS
Parallel Coprocessor System
User Manual

$35 EACH
NOW AVAILABLE
FROM THE FORTH INTEREST GROUP

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