Curly Control Structure Set (I)

Forth Systems Comparison

PCYerk Classes

Minimal Forth Wordsets
Announcing the SC/FOX IO32 Board for FAST Forth I/O

**SC/FOX IO32 Board Features**
- The IO32 is a plug-on daughter board for either the SBC32 stand-alone or PCS32 PC plug-in single board computers.
- 5 MB/sec SCSI Port.
- Attach up to 7 SCSI Devices.
- 4 RS232 Serial Ports, up to 230K baud.
- 16-bit Bidirectional-Parallel Port, may be used as two 8-bit ports.
- 2 programmable counter/timers.
- Prototyping area on board.
- All bus signal brought out to pads.
- Full interrupt Support.
- Two 50-pin user application connectors.
- No jumpers, totally software configurable.
- Driver software source included.
- Single +5 Volt low-power operation.
- Full ground and power plane.
- 4 Layer, Eurocard-size: 100mm x 160mm.
- User manual and interface schematics included.
- Low chip count (8 ICs) for maximum reliability.
- Test routines for SCSI, parallel, and serial ports supplied in source code form.
- Plug together up to 6 IO32 Boards in a stack.

**Fast Data-Dispersion Program Example**
The program, SEND below, reads 1K blocks from a SCSI drive and transmits them out one of the IO32 board's four RS232 serial ports at 230K baud. SEND uses only IO32 facilities. Disk read speed is limited by SCSI drive speed.

```
EXPORT SEND
CREATE BUFR 2560 ALLOT
PUT ( #k)
1024 * BUFR BYTE +
BUFFR BYTE DO
1 C@
UERM
LOOP ( until done)
SEND ( block# #k)
230KB
BEGIN ?DUP WHILE ( while blocks remain )
2DUP 10 MIN ( max 10K in buf)
>R BUFR R@ SCSIRD ( read nK from SCSI)
R@ PUT ( and put to serial)
R@ - ( decrement remaining)
SWAP R> + SWAP ( up new starting block)
REPEAT ( repeat remaining test)
DROP ( discard blk# and exit)
```

For additional product and pricing information, please contact us at:
SILICON COMPOSERS INC 208 California Avenue, Palo Alto, CA 94306 (415) 322-8763
6 **Forth Systems Comparisons**
Guy M. Kelly

Award-winning paper dissecting and comparing nineteen Forths, including commercial, shareware, and public-domain systems. Provides detailed testing information, analysis of how to compare different Forths, discussion of critical non-performance-related factors, and timing results. Won the "Public Service" award at the 1991 FORML Conference.

22 **The Curly Control Structure Set**
Kourtis Giorgio

After an in-depth review of all the literature on Forth control structures, and following two years of development and testing, the author proposes a new set of control structures for you to try out. In search of performance, ease of use, generalization, flexibility, and teachability without sacrificing too much in terms of historical continuity? Think it can't be done, at least not better? Or just want to brush up on how control structures work? Open your mind and sharpen your wits...Part one of two.

38 **A FORML Thanksgiving**
Richard Molen

The annual Forth Modification Laboratory—FORML—is a long-standing Forth tradition. Join those who gathered on California's Monterey peninsula to discuss new proposals, Forth hardware, emulators, embedded systems, and the usual (and unusual) wide-ranging fare.

39 **PCYerk Classes**
Rick Grehan

Last issue's winner of the "Object-Oriented Forth" contest returns with supplemental code. Here you will find the Forth foundations of basic classes, storage classes, byte and word arrays, strings, and string arrays. Use this or another object-oriented Forth to explore the symbiosis of traditional Forth and classical object-style programming.

Departments

4 **Editorial** Coming attractions, call for contest papers, about this issue.

31 **New FIG Board Members** Mike Elola, Nicholas Solntseff, Jack Woehr

32 **Best of GENie** Dearth of feedback on ANS Forth; how minimal can Forth get?

36 **Volume XII Index** The complete subject index to *Forth Dimensions* contents published from May 1990 – April 1991.

37 **Advertisers Index** Sources of information about FIG, ANS Forth, classes, on-line Forth connections, and FIG chapters will return in the next issue.
Editorial

New Contest for Forth Authors!

With this issue, many of you will be due to renew your membership in the Forth Interest Group. This is a year to do so promptly and to make a gift membership or two for the office, a co-worker, or friend—here are a few things to look for in coming issues of Forth Dimensions:

1. A West-coast group of Forth adepts is producing a series of articles applying Forth to hands-on, hardware-software projects that you can do—a laboratory for increasing your Forth proficiency at the workbench. Vendors and developers will have more opportunities to contribute technical and industry information in ways that will show what they are doing successfully and where Forth is excelling in real-world application. We have scheduled tutorials about traditional tools like CREATE DOES> as well as the control structures that will be introduced in ANS Forth. More than ever we believe that, from beginner to expert, every Forth user and project manager will want to receive the vital information that will be appearing here.

2. Reader participation has always been a key element of this publication. Your contributions are the lifeblood of our pages, dramatically helping to chart our direction. We not only welcome your own articles and letters to the editor, we need them.

FD can now announce the third in a series of contests for Forth authors. The first called for entries about Forth hardware, and the winners were published in our issue XI/6. More recently, the winners of our object-oriented Forth contest appeared in issue XII/5. Drawing on feedback from Forth vendors, the theme of our current contest is Forth in large-scale applications.

This is our first call for papers about "FORTH ON A GRAND SCALE." This theme applies equally to projects requiring multiple programmers, and to applications or systems consisting of large amounts of code and/or of significant complexity. Papers will be refereed. To encourage entries, the author of the winning article will receive $500, the second-place $250, and the third-place $100. Articles will be evaluated for publication even if they do not win a cash prize.

You need not have been personally involved in the subject of your entry, just write about it in sufficient technical detail, and address the particular challenges that were faced and describe how (or whether) they were overcome. Chances are, if you think a subject might fit the theme of this contest, the judges will be anxious to include it in their evaluations—so get started soon. The deadline for contest entries is August 3, 1992. Mail a hard copy and a diskette (Macintosh 800K or PC preferred) to the Forth Interest Group, P.O. Box 8231, San Jose, California 95155; or mail the hard copy and upload an ASCII version to MARLIN.OA on GENie's e-mail service with an attached note describing the file and compression/archive format, if any. We all look forward to receiving your contribution!

At the other end of the scale we have minimal Forths. How small can you get and still have a language? What are the fewest required words in Forth? That is the on-line discussion excerpted in "The Best of GENie" this month. Elsewhere in this issue, you will find supplemental disk materials for the object-oriented Forth "PCYerk" by Rick Grehan, and a meaty discussion of control structures by Kourtis Giorgio that will be concluded in the next issue. Finally, Guy Kelly shares his FORML paper with FD readers. It is a significant piece of work that shows what goes into evaluating Forth systems, and we thank him for allowing us to publish it here. It demonstrates the difficulty of doing head-to-head product comparisons, and is the first substantial attempt we know of to do so thoroughly and objectively. Pay special heed to his warning that benchmark excellence alone does not mean that any single system will be the right one every purpose!

Martin Overson
Editor
MEET THAT DEADLINE !!!

- Use subroutine libraries written for other languages! More efficiently!
- Combine raw power of extensible languages with convenience of carefully implemented functions!
- Yes, it is faster than optimized C!
- Compile 40,000 lines per minute!
- Stay totally interactive, even while compiling!
- Program at any level of abstraction from machine code thru application specific language with equal ease and efficiency!
- Alter routines without recompiling!
- Use source code for 2500 functions!
- Use data structures, control structures, and interface protocols from any other language!
- Implement borrowed feature, often more efficiently than in the source!
- Use an architecture that supports small programs or full megabyte ones with a single version!
- Forget chaotic syntax requirements!
- Outperform good programmers stuck using conventional languages! (But only until they also switch.)

WAKE UP !!!

Forth is no longer a language that tempts programmers with "great expectations", then frustrates them with the need to reinvent simple tools expected in any commercial language.

HS/FORTH Meets Your Needs!

Don't judge Forth by public domain products or ones from vendors primarily interested in consulting - they profit from not providing needed tools! Public domain versions are cheap - if your time is worthless. Useful in learning Forth's basics, they fail to show its true potential. Not to mention being a-l-o-w.

We don't shortchange you with promises. We provide implemented functions to help you complete your application quickly. And we ask you not to shortchange us by trying to save a few bucks using inadequate public domain or pirate versions. We worked hard coming up with the ideas that you now see sprouting up in other Forths. We won't throw in the towel, but the drain on resources delays the introduction of even better tools. Don't kid yourself, you are not just another drop in the bucket, your personal decision really does matter. In return, we'll provide you with the best tools money can buy.

The only limit with Forth is your own imagination!

You can't add extensibility to fossilized compilers. You are at the mercy of that language's vendor. You can easily add features from other languages to HS/FORTH. And using our automatic optimizer or learning a very little bit of assembly language makes your addition zip along as well as in the parent language.

Speaking of assembly language, learning it in a supportive Forth environment turns the learning curve into a light speed escalator. People who failed previous attempts to use assembly language, conquer it in a few hours or days using HS/FORTH.

MEET THAT DEADLINE !!!

- Use subroutine libraries written for other languages! More efficiently!
- Combine raw power of extensible languages with convenience of carefully implemented functions!
- Yes, it is faster than optimized C!
- Compile 40,000 lines per minute!
- Stay totally interactive, even while compiling!
- Program at any level of abstraction from machine code thru application specific language with equal ease and efficiency!
- Alter routines without recompiling!
- Use source code for 2500 functions!
- Use data structures, control structures, and interface protocols from any other language!
- Implement borrowed feature, often more efficiently than in the source!
- Use an architecture that supports small programs or full megabyte ones with a single version!
- Forget chaotic syntax requirements!
- Outperform good programmers stuck using conventional languages! (But only until they also switch.)

WAKE UP !!!

Forth is no longer a language that tempts programmers with "great expectations", then frustrates them with the need to reinvent simple tools expected in any commercial language.

HS/FORTH Meets Your Needs!

Don't judge Forth by public domain products or ones from vendors primarily interested in consulting - they profit from not providing needed tools! Public domain versions are cheap - if your time is worthless. Useful in learning Forth's basics, they fail to show its true potential. Not to mention being a-l-o-w.

We don't shortchange you with promises. We provide implemented functions to help you complete your application quickly. And we ask you not to shortchange us by trying to save a few bucks using inadequate public domain or pirate versions. We worked hard coming up with the ideas that you now see sprouting up in other Forths. We won't throw in the towel, but the drain on resources delays the introduction of even better tools. Don't kid yourself, you are not just another drop in the bucket, your personal decision really does matter. In return, we'll provide you with the best tools money can buy.

The only limit with Forth is your own imagination!

You can't add extensibility to fossilized compilers. You are at the mercy of that language's vendor. You can easily add features from other languages to HS/FORTH. And using our automatic optimizer or learning a very little bit of assembly language makes your addition zip along as well as in the parent language.

Speaking of assembly language, learning it in a supportive Forth environment turns the learning curve into a light speed escalator. People who failed previous attempts to use assembly language, conquer it in a few hours or days using HS/FORTH.

WINTER SALE >>>
Code fragments and benchmarks for several of the Forths for the PC are outlined to illustrate various tradeoffs and their effect on performance.

The following list represents some of the Forths I have been able to study. They span a wide range of implementation tradeoffs and provide some insight into the results of these tradeoffs.

<table>
<thead>
<tr>
<th>Forth</th>
<th>Model</th>
<th>Author(s)</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBL</td>
<td>83</td>
<td>Green</td>
<td>public</td>
</tr>
<tr>
<td>eForth</td>
<td>X3J14</td>
<td>Muench &amp; Ting</td>
<td>public</td>
</tr>
<tr>
<td>F83</td>
<td>83</td>
<td>Laxen &amp; Perry</td>
<td>public</td>
</tr>
<tr>
<td>F-PC</td>
<td>83</td>
<td>Zimmer &amp; Smith</td>
<td>public</td>
</tr>
<tr>
<td>Fifth</td>
<td></td>
<td>Click &amp; Snow</td>
<td>share</td>
</tr>
<tr>
<td>HS/FORTH</td>
<td>*</td>
<td>Callahan</td>
<td>commercial</td>
</tr>
<tr>
<td>KForth</td>
<td></td>
<td>experimental</td>
<td>copyrighted</td>
</tr>
<tr>
<td>LaFORTH</td>
<td></td>
<td>Kelly</td>
<td>copyrighted</td>
</tr>
<tr>
<td>MMSFORTH</td>
<td>79</td>
<td>Miller et al.</td>
<td>commercial</td>
</tr>
<tr>
<td>MVP-FORTH</td>
<td>79</td>
<td>Haydon</td>
<td>public</td>
</tr>
<tr>
<td>PC-Forth</td>
<td>83</td>
<td>Kelly</td>
<td>public</td>
</tr>
<tr>
<td>PMFORTH</td>
<td>fig</td>
<td>Moreton</td>
<td>commercial</td>
</tr>
<tr>
<td>polyFORTH</td>
<td>83</td>
<td>Moore et al.</td>
<td>commercial</td>
</tr>
<tr>
<td>Pygmy</td>
<td>cmFORTH</td>
<td>Sergeant</td>
<td>copyrighted</td>
</tr>
<tr>
<td>riFORTH</td>
<td>cmFORTH</td>
<td>Illyes</td>
<td>copyrighted</td>
</tr>
<tr>
<td>UniForth</td>
<td>83</td>
<td>Hendon?</td>
<td>share</td>
</tr>
<tr>
<td>Upper Deck</td>
<td>83</td>
<td>Graves</td>
<td>commercial</td>
</tr>
<tr>
<td>UR/FORTH</td>
<td>83</td>
<td>Duncan &amp; Wilton</td>
<td>commercial</td>
</tr>
<tr>
<td>ZEN</td>
<td>X3J14</td>
<td>Tracy</td>
<td>copyrighted</td>
</tr>
</tbody>
</table>

*Includes overlays to convert to fig, 79, or 83 standard.

Some of these Forths are available in different packages including public, share, or commercial versions. The version tested had the status indicated. The non-commercial versions are typically available at no charge, the commercial versions are typically copyrighted. The model does not imply compatibility.

These Forths cover a range of categories and complexities, as Table One illustrates.

Segment Models

Assuming four logical segments (not including the stacks), there are 15 different models. The following lists these models and indicates their use by each of the Forths studied.

C+L+D+H  e, F83, La, MMS, MVP, PC, pygmy, ri, Uni
C+L+H D  polyFORTH
C+L D+H  PM, ZEN
C L+D H  BBL, HS/FORTH, UR/FORTH
C+D L H  F-PC
C L D H  KForth, Upper Deck

Not found:
C L+D+H
L C+D+H
H C+L+D
C+D L+H
C+H L+D
C L D+H
C D L+H
L D C+H
D H C+L

Descriptions

Brief descriptions of most of the Forths tested are included at the end of this paper (all assembly code is in a common format).

Benchmarks

While studying the various threading, stack, and segmenting methods it seemed that a set of simple benchmarks could help in evaluating the performance trade-offs. The benchmarks arrived at are specifically aimed at the attributes studied and do not necessarily correlate with real applications.

Threading

There are two aspects of threading in Forth to be evaluated. The efficiency of incrementing the Forth instruction-pointer and the efficiency of nesting (and unnesting).
The following threading benchmarks were used:

\begin{verbatim}
\emptyloop:\texttt{Empty} = XX
X (--) 30,000 0 DO LOOP ;
XX (--) 5 0 DO X LOOP ;
\end{verbatim}

\begin{verbatim}
\threading:\texttt{Thread} = YY - XX
CODE NC (--) NEXT, END-CODE
Y (--) 30,000 0 DO NC NC NC NC NC NC LOOP ;
YY (--) 5 0 DO Y LOOP ;
\end{verbatim}

\begin{verbatim}
\nesting1:\texttt{Nestl} = ZZ - XX
N: (--) ;
Z (--) ;
30,000 0 DO N: N: N: N: N: LOOP ;
ZZ (--) 5 0 DO Z LOOP ;
\end{verbatim}

\begin{verbatim}
\nesting2:\texttt{Nest2} = WW - XX
W1 ; : W2 W1 ; : W3 W2 ; : W4 W3 ;
W5 W4 ; : W6 W5 ;
W (--) 30,000 0 DO W6 LOOP ;
WW (--) 5 0 DO W LOOP ;
\end{verbatim}

The two nesting benchmarks should be equivalent but can be very different depending upon any optimization applied.

---

### Table One.

<table>
<thead>
<tr>
<th>Forth</th>
<th>Threading</th>
<th>Width¹</th>
<th>Stack</th>
<th>Segments²</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBL</td>
<td>direct</td>
<td>32/16s</td>
<td>in reg</td>
<td>N=C,m(L+D),n*H,S+B</td>
</tr>
<tr>
<td>eForth</td>
<td>direct</td>
<td>16/16</td>
<td>on stack</td>
<td>1 (sep. heads)</td>
</tr>
<tr>
<td>F83</td>
<td>indirect</td>
<td>16/16</td>
<td>on stack</td>
<td>1</td>
</tr>
<tr>
<td>F-PC</td>
<td>direct</td>
<td>16/16pp</td>
<td>on stack</td>
<td>3=C+D+S,mL,H</td>
</tr>
<tr>
<td>Fifth</td>
<td>subroutine?</td>
<td>32?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>HS/FORTH</td>
<td>indirect</td>
<td>16/16</td>
<td>in reg</td>
<td>N=n<em>C,n</em>(L+D),n*H,S</td>
</tr>
<tr>
<td>KForth</td>
<td>direct</td>
<td>16/16</td>
<td>in reg</td>
<td>5=C,L,D,H,S</td>
</tr>
<tr>
<td>LaFORTH</td>
<td>direct</td>
<td>16/16</td>
<td>in reg</td>
<td>2(2nd for text files)</td>
</tr>
<tr>
<td>MMsforth</td>
<td>indirect</td>
<td>16/16</td>
<td>on stack</td>
<td>1(non-DOS), 2=C+L+D,H</td>
</tr>
<tr>
<td>MVP-FORTH</td>
<td>indirect</td>
<td>16/16</td>
<td>on stack</td>
<td>1</td>
</tr>
<tr>
<td>PC-Forth</td>
<td>indirect</td>
<td>16/16</td>
<td>on stack</td>
<td>1</td>
</tr>
<tr>
<td>PMFORTH</td>
<td>direct</td>
<td>16/16</td>
<td>on stack</td>
<td>2=C+L,D+H+S</td>
</tr>
<tr>
<td>polyFORTH</td>
<td>indirect</td>
<td>16/16</td>
<td>on stack</td>
<td>N=n*(C+L+H),(D+S),n*D</td>
</tr>
<tr>
<td>Pygmy</td>
<td>direct</td>
<td>16/16</td>
<td>in reg</td>
<td>1</td>
</tr>
<tr>
<td>riFORTH</td>
<td>subroutine</td>
<td>16/16</td>
<td>in reg</td>
<td>1</td>
</tr>
<tr>
<td>UniForth</td>
<td>indirect</td>
<td>16/16</td>
<td>on stack</td>
<td>1</td>
</tr>
<tr>
<td>Upper Deck</td>
<td>direct</td>
<td>16/16</td>
<td>in reg</td>
<td>5=C,L,D,H,S</td>
</tr>
<tr>
<td>UR/FORTH</td>
<td>direct</td>
<td>16/16</td>
<td>in reg</td>
<td>4=C,L+D,H,S</td>
</tr>
<tr>
<td>ZEN</td>
<td>direct</td>
<td>16/16</td>
<td>in reg</td>
<td>2=C+L,D+H+S</td>
</tr>
</tbody>
</table>

1. Width given as: stack-width/token-width; os indicates token is an offset into the code segment, pp indicates token is a 16-bit paragraph address.
2. Code, List, Data, Head, and Stack; m(L) indicates one meg. of paragraph space for tokens; n*(L+D) or n*H indicates n 64K segments for lists+data or heads.
Interpret-time benchmark.

Interpret-time: Loads (tests: WORD, NUMBER, and FIND etc.)

```
99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP
99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP
99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP
99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP
99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP
99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP
99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP
99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP
99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP
99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP
99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP
99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP
99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP
99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP
99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP
99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP
99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP
99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP 99 DROP
```

Note: Loads is strongly influenced by the search method and, in many Forths, by the number of words in the dictionary.

Results

Initial testing was done on a 20 MHz '386; however, if the code or list addresses were moved from non-word to word boundaries, the times were significantly improved. This effect was noticed using PC-Forth and only investigated for PC-Forth and PC/FORTH (a now discontinued product from LMI, which resisted attempts to force code or list addresses to non-word boundaries). A 4.77 MHz 8088 did not exhibit this behavior and was used to obtain the results listed in Tables Two and Three.

The benchmarks arrived at are specifically aimed at the attributes studied and do not necessarily correlate with real applications.

### Table Two

<table>
<thead>
<tr>
<th>Forth</th>
<th>Type</th>
<th>Empty</th>
<th>Thread</th>
<th>Nest1</th>
<th>Nest2</th>
<th>Prims</th>
<th>Sieve</th>
<th>Loads</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBL</td>
<td>D-R-L</td>
<td>4.0</td>
<td>5.9</td>
<td>41.4</td>
<td>41.4</td>
<td>33.1</td>
<td>49.0</td>
<td>4.3</td>
</tr>
<tr>
<td>eForth</td>
<td>D-S</td>
<td>4.3</td>
<td>--</td>
<td>42.2</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>F83</td>
<td>I-S-J</td>
<td>5.4</td>
<td>15.1</td>
<td>43.4</td>
<td>43.4</td>
<td>41.5</td>
<td>68.1</td>
<td>3.8</td>
</tr>
<tr>
<td>F-PC</td>
<td>D-S-P</td>
<td>3.1</td>
<td>11.0</td>
<td>46.4</td>
<td>46.4</td>
<td>30.3</td>
<td>44.9</td>
<td>0.9</td>
</tr>
<tr>
<td>Fifth</td>
<td>S-?</td>
<td>7.1</td>
<td>--</td>
<td>20.9</td>
<td>20.9</td>
<td>70.4</td>
<td>97.2</td>
<td>--</td>
</tr>
<tr>
<td>HS/FORTH</td>
<td>I-R</td>
<td>5.5</td>
<td>9.8</td>
<td>34.2</td>
<td>33.8</td>
<td>28.5</td>
<td>48.8</td>
<td>0.7</td>
</tr>
<tr>
<td>KForth</td>
<td>D-R</td>
<td>2.9</td>
<td>5.8</td>
<td>25.8</td>
<td>25.8</td>
<td>21.6</td>
<td>36.2</td>
<td>21.9</td>
</tr>
<tr>
<td>LaFORTH</td>
<td>D-R</td>
<td>3.6</td>
<td>5.8</td>
<td>29.9</td>
<td>29.7</td>
<td>26.9</td>
<td>36.3</td>
<td>0.5</td>
</tr>
<tr>
<td>MMSFORTH</td>
<td>I-S</td>
<td>5.2</td>
<td>10.0</td>
<td>37.4</td>
<td>37.4</td>
<td>33.4</td>
<td>55.6</td>
<td>0.5</td>
</tr>
<tr>
<td>MVP-FORTH</td>
<td>I-S-J</td>
<td>14.2</td>
<td>19.8</td>
<td>53.3</td>
<td>53.3</td>
<td>50.8</td>
<td>---</td>
<td>8.5</td>
</tr>
<tr>
<td>PC-Forth</td>
<td>I-S</td>
<td>3.6</td>
<td>10.0</td>
<td>34.5</td>
<td>34.5</td>
<td>32.7</td>
<td>54.8</td>
<td>8.4</td>
</tr>
<tr>
<td>PMFORTH</td>
<td>D-S</td>
<td>5.5</td>
<td>9.6</td>
<td>43.5</td>
<td>43.5</td>
<td>39.8</td>
<td>70.7</td>
<td>15.1</td>
</tr>
<tr>
<td>polyFORTH</td>
<td>I-S</td>
<td>4.8</td>
<td>9.9</td>
<td>34.5</td>
<td>34.5</td>
<td>31.5</td>
<td>52.9</td>
<td>1.0</td>
</tr>
<tr>
<td>Pygmy</td>
<td>D-R</td>
<td>3.2</td>
<td>5.9</td>
<td>32.0</td>
<td>32.0</td>
<td>23.5</td>
<td>39.7</td>
<td>4.7</td>
</tr>
<tr>
<td>riFORTH</td>
<td>S-R</td>
<td>13.1</td>
<td>9.6</td>
<td>9.6</td>
<td>5.4</td>
<td>9.6</td>
<td>34.8</td>
<td>6.8</td>
</tr>
<tr>
<td>UniForth</td>
<td>I-S</td>
<td>3.6</td>
<td>10.7</td>
<td>35.9</td>
<td>35.8</td>
<td>33.3</td>
<td>---</td>
<td>2.7</td>
</tr>
<tr>
<td>Upper Deck</td>
<td>D-R</td>
<td>4.2</td>
<td>5.9</td>
<td>32.1</td>
<td>32.1</td>
<td>23.8</td>
<td>39.8</td>
<td>0.5</td>
</tr>
<tr>
<td>UR/FORTH</td>
<td>D-R</td>
<td>2.1</td>
<td>5.8</td>
<td>31.9</td>
<td>32.0</td>
<td>24.3</td>
<td>38.2</td>
<td>0.5</td>
</tr>
<tr>
<td>ZEN</td>
<td>D-R</td>
<td>3.6</td>
<td>6.8</td>
<td>34.1</td>
<td>33.8</td>
<td>27.4</td>
<td>44.6</td>
<td>4.4</td>
</tr>
</tbody>
</table>

All times in seconds, all measurements on a 4.77 MHz 8088 PC.
### Table Three.

<table>
<thead>
<tr>
<th></th>
<th>Type</th>
<th>Empty</th>
<th>Thread</th>
<th>Nest1</th>
<th>Nest2</th>
<th>Prims</th>
<th>Sieve</th>
<th>Loads</th>
</tr>
</thead>
<tbody>
<tr>
<td>rFORTH</td>
<td>S-R</td>
<td>13.1</td>
<td>9.6</td>
<td>9.6</td>
<td>5.4</td>
<td>9.6</td>
<td>34.8</td>
<td>6.8</td>
</tr>
<tr>
<td>Fifth</td>
<td>S-R</td>
<td>7.1</td>
<td>---</td>
<td>20.9</td>
<td>20.9</td>
<td>70.4</td>
<td>97.2</td>
<td>---</td>
</tr>
<tr>
<td>KForth</td>
<td>D-R</td>
<td>3.0</td>
<td>5.8</td>
<td>25.8</td>
<td>25.8</td>
<td>21.6</td>
<td>36.2</td>
<td>21.9</td>
</tr>
<tr>
<td>LaFORTH</td>
<td>D-R</td>
<td>3.6</td>
<td>5.8</td>
<td>29.9</td>
<td>29.7</td>
<td>26.9</td>
<td>36.3</td>
<td>0.5</td>
</tr>
<tr>
<td>UR/FORTH</td>
<td>D-R</td>
<td>3.0</td>
<td>5.8</td>
<td>31.9</td>
<td>32.0</td>
<td>24.3</td>
<td>38.2</td>
<td>0.5</td>
</tr>
<tr>
<td>Pygmy</td>
<td>D-R</td>
<td>3.2</td>
<td>5.9</td>
<td>32.0</td>
<td>32.0</td>
<td>23.5</td>
<td>39.7</td>
<td>4.7</td>
</tr>
<tr>
<td>Upper Deck</td>
<td>D-R</td>
<td>4.2</td>
<td>5.9</td>
<td>32.1</td>
<td>32.1</td>
<td>23.8</td>
<td>39.8</td>
<td>0.5</td>
</tr>
<tr>
<td>ZEN</td>
<td>D-R</td>
<td>3.6</td>
<td>6.8</td>
<td>34.1</td>
<td>33.8</td>
<td>27.4</td>
<td>4.6</td>
<td>4.4</td>
</tr>
<tr>
<td>HS/FORTH</td>
<td>I-R</td>
<td>5.5</td>
<td>9.8</td>
<td>34.2</td>
<td>33.8</td>
<td>28.5</td>
<td>48.8</td>
<td>0.7</td>
</tr>
<tr>
<td>polyFORTH</td>
<td>I-S</td>
<td>4.8</td>
<td>9.9</td>
<td>34.5</td>
<td>34.5</td>
<td>31.5</td>
<td>52.9</td>
<td>1.0</td>
</tr>
<tr>
<td>PC-Forth</td>
<td>I-S</td>
<td>3.6</td>
<td>10.0</td>
<td>34.5</td>
<td>34.5</td>
<td>32.7</td>
<td>54.8</td>
<td>8.4</td>
</tr>
<tr>
<td>UniForth</td>
<td>I-S</td>
<td>3.6</td>
<td>10.7</td>
<td>35.9</td>
<td>35.8</td>
<td>33.3</td>
<td>---</td>
<td>2.7</td>
</tr>
<tr>
<td>MMSFORTH</td>
<td>I-S</td>
<td>5.2</td>
<td>10.0</td>
<td>37.4</td>
<td>37.4</td>
<td>33.4</td>
<td>55.6</td>
<td>0.5</td>
</tr>
<tr>
<td>BBL</td>
<td>D-R-L</td>
<td>4.0</td>
<td>5.9</td>
<td>41.4</td>
<td>41.4</td>
<td>33.1</td>
<td>49.0</td>
<td>4.3</td>
</tr>
<tr>
<td>eForth</td>
<td>D-S</td>
<td>4.3</td>
<td>---</td>
<td>42.2</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>F83</td>
<td>I-S-J</td>
<td>5.4</td>
<td>15.1</td>
<td>43.4</td>
<td>43.4</td>
<td>41.5</td>
<td>68.1</td>
<td>3.8</td>
</tr>
<tr>
<td>PMFORTH</td>
<td>D-S</td>
<td>5.5</td>
<td>9.6</td>
<td>43.5</td>
<td>43.5</td>
<td>39.8</td>
<td>70</td>
<td>15.1</td>
</tr>
<tr>
<td>F-PC</td>
<td>D-S-P</td>
<td>3.1</td>
<td>11.0</td>
<td>46.4</td>
<td>46.4</td>
<td>30.3</td>
<td>44.9</td>
<td>0.9</td>
</tr>
<tr>
<td>MVP-FORTH</td>
<td>I-S-J</td>
<td>14.2</td>
<td>19.8</td>
<td>53.3</td>
<td>53.3</td>
<td>50.8</td>
<td>---</td>
<td>8.5</td>
</tr>
</tbody>
</table>

Sorted by nesting time.
1. Type: Indirect, Direct, or Subroutine threading,
   stack-top in Register, or on Stack,
   1-meg. Lists, 1-meg. lists on Paragraphs, Jump to NEXT.

---

### Timing

In general the results were as follows (fastest to slowest):
- subroutine threading; top-of-stack in register,
- direct threading; top-of-stack in register,
- indirect threading; top-of-stack in register,
- indirect threading; top-of-stack in memory,
- direct threading; top-of-stack in memory.1

1. Expected to be third, not last (PMFORTH was the only example).

---

**To obtain the maximum advantage from Forth, one should understand the rationale for its structure and its inherent strengths and weaknesses.**

---

### "16-bit"

<table>
<thead>
<tr>
<th>Threading</th>
<th>Nesting</th>
<th>Primitives</th>
<th>Sieve</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subroutine</td>
<td>10</td>
<td>5/10</td>
<td>10</td>
</tr>
<tr>
<td>Direct</td>
<td>6-07</td>
<td>25-34</td>
<td>22-27</td>
</tr>
<tr>
<td>Indirect</td>
<td>10</td>
<td>34-37</td>
<td>28-33</td>
</tr>
<tr>
<td>I (JMP NEXT)</td>
<td>15-20</td>
<td>43-53</td>
<td>42-50</td>
</tr>
</tbody>
</table>

### "16-bit paragraphs"

| F-PC (Dir) | 11      | 46        | 30    | 45   |

### "32-bit"

| BBL (Dir) | 06      | 41        | 33    | 49   |
| Fifth (Sub) | --    | 21        | 70    | 97   |

---

Forth Dimensions 9 March 1992 April
Optimization

Two of the Forths allowed optimization of user specified words. The results obtained using the optimizers are shown in Table Four.

Comments

Several aspects of these Forths make direct comparison difficult. Most of them do not automatically optimize their code nor do they directly span multiple segments. However, riFORTH does automatic optimization; polyFORTH has multiple C+1+H spaces; BBL, F-PC, and Fifth have up to one meg. of list space; F83 and MVP-FORTH have a central NEXT; MVP-FORTH and PMFORTH have inefficient versions of NEXT; KForth does high-level parsing; eForth interprets files via a serial link; and LaFORTH uses a 64K text buffer.

Further Tests

Because of the differences mentioned above, a test-set of five different versions of Forth were produced. They were all derived from riFORTH (a subroutine-threaded Forth available in a minimum number of screens). The versions (including riFORTH) were:

<table>
<thead>
<tr>
<th>Name</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-R-S</td>
<td>Subroutine in Register</td>
</tr>
<tr>
<td>D-R-M</td>
<td>Direct in Register</td>
</tr>
<tr>
<td>D-S-M</td>
<td>Direct on Stack</td>
</tr>
<tr>
<td>I-R-M</td>
<td>Indirect in Register</td>
</tr>
<tr>
<td>I-S-M</td>
<td>Indirect on Stack</td>
</tr>
<tr>
<td>D-R-S</td>
<td>Direct in Register</td>
</tr>
</tbody>
</table>

The versions were optimized for speed at the expense of size. All models used an in-line NEXT and in-line nest, LIT, etc., where possible. The benchmark results (sorted by nesting time) are given in Table Five.

Note that for riFORTH, Nest2 is almost twice as fast as Nest1 while Thread and Nest1 take the same time. This is because riFORTH is subroutine-threaded and has built-in optimization. Referring to the nesting benchmarks, the “code” no-op NC, and the “colon” no-ops ;N and W1 all compile as return instructions. However, W2 is compiled as a jump to W1, W3 as a jump to W2, etc., thus doing five jumps and a return inside the W loop instead of six call-return pairs. Also note that the Prims are executed much faster for riFORTH than the other versions (because riFORTH drops adjacent XCHG BP, SI pairs from “code macros” as it compiles them into the list field of a colon definition), while the Sieve (which uses a high-level DO LOOP) is only slightly faster.

The apparent anomaly among the other versions is D-R-S, the only one of the five that is not multi-segment. It nests more slowly but does Prims and Sieve faster than D-S-M because nest, LIT, and VARIABLE cannot be as highly optimized for speed.

Table Four. Results using optimizers.

<table>
<thead>
<tr>
<th>Forth</th>
<th>Type</th>
<th>Empty</th>
<th>Thread</th>
<th>Nest1</th>
<th>Nest2</th>
<th>Prims</th>
<th>Sieve</th>
<th>Loads</th>
</tr>
</thead>
<tbody>
<tr>
<td>HS/FORTH optimized</td>
<td>I-R</td>
<td>5.5</td>
<td>9.8</td>
<td>34.2</td>
<td>33.8</td>
<td>28.5</td>
<td>48.4</td>
<td>0.7</td>
</tr>
<tr>
<td>UR/FORTH optimized</td>
<td>D-R</td>
<td>3.0</td>
<td>5.8</td>
<td>31.9</td>
<td>32.0</td>
<td>24.3</td>
<td>38.2</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Table Five. Performance of test-set versions of Forth.

<table>
<thead>
<tr>
<th>Name</th>
<th>Empty</th>
<th>Thread</th>
<th>Nest1</th>
<th>Nest2</th>
<th>Prims</th>
<th>Sieve</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-R-S</td>
<td>13.1</td>
<td>9.6</td>
<td>9.6</td>
<td>5.4</td>
<td>9.6</td>
<td>34.8</td>
</tr>
<tr>
<td>D-R-M</td>
<td>2.9</td>
<td>5.8</td>
<td>25.8</td>
<td>25.8</td>
<td>21.5</td>
<td>36.2</td>
</tr>
<tr>
<td>D-S-M</td>
<td>2.9</td>
<td>5.8</td>
<td>25.8</td>
<td>25.8</td>
<td>21.5</td>
<td>42.3</td>
</tr>
<tr>
<td>D-R-S</td>
<td>2.9</td>
<td>5.8</td>
<td>31.9</td>
<td>32.3</td>
<td>23.4</td>
<td>37.9</td>
</tr>
<tr>
<td>I-R-M</td>
<td>3.6</td>
<td>10.0</td>
<td>33.8</td>
<td>33.5</td>
<td>28.8</td>
<td>47.7</td>
</tr>
<tr>
<td>I-S-M</td>
<td>3.6</td>
<td>10.0</td>
<td>33.8</td>
<td>33.5</td>
<td>32.1</td>
<td>53.2</td>
</tr>
</tbody>
</table>
The following lists the versions of NEXT, nest, EXIT, literal, CONSTANT, VARIABLE, @, !, and + used in these models.

### Code Fragments

<table>
<thead>
<tr>
<th></th>
<th>D-R-M</th>
<th>D-S-M</th>
<th>I-R-M</th>
<th>I-S-M</th>
<th>D-R-S</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EXIT</strong></td>
<td>MOV SI,[BP]</td>
<td>see D-R-M</td>
<td>see D-R-M</td>
<td>see D-R-M</td>
<td>see D-R-M</td>
</tr>
<tr>
<td></td>
<td>INC BP</td>
<td>&lt;--</td>
<td>&lt;--</td>
<td>&lt;--</td>
<td>&lt;--</td>
</tr>
<tr>
<td></td>
<td>INC BP</td>
<td>&lt;--</td>
<td>&lt;--</td>
<td>&lt;--</td>
<td>&lt;--</td>
</tr>
<tr>
<td></td>
<td>NEXT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>nest</strong></td>
<td>---</td>
<td>see D-R-M</td>
<td>see D-R-M</td>
<td>see D-R-M</td>
<td>JMP nest</td>
</tr>
<tr>
<td></td>
<td>DEC BP</td>
<td>&lt;--</td>
<td>&lt;--</td>
<td>&lt;--</td>
<td>DEC BP</td>
</tr>
<tr>
<td></td>
<td>DEC BP</td>
<td>&lt;--</td>
<td>&lt;--</td>
<td>&lt;--</td>
<td>DEC BP</td>
</tr>
<tr>
<td></td>
<td>MOV [BP],SI</td>
<td></td>
<td></td>
<td></td>
<td>MOV[BP],SI</td>
</tr>
<tr>
<td></td>
<td>MOV SI,addr</td>
<td></td>
<td></td>
<td></td>
<td>ADD AX,3</td>
</tr>
<tr>
<td></td>
<td>NEXT</td>
<td></td>
<td></td>
<td></td>
<td>MOV SI,AX</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NEXT</td>
</tr>
<tr>
<td><strong>NEXT</strong></td>
<td>LODSW</td>
<td>see D-R-M</td>
<td>LODSW</td>
<td>LODSW</td>
<td>see D-R-M</td>
</tr>
<tr>
<td></td>
<td>JMP AX</td>
<td>&lt;--</td>
<td>XCHG AX,DX</td>
<td>XCHG AX,BX</td>
<td>(LODSW</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>JMP [DX]</td>
<td>JMP [BX]</td>
<td>JMP AX)</td>
</tr>
<tr>
<td><strong>CON</strong></td>
<td>PUSH BX</td>
<td>see D-R-M</td>
<td>see D-R-M</td>
<td>see D-R-M</td>
<td>see D-R-M</td>
</tr>
<tr>
<td></td>
<td>MOV BX, #</td>
<td>&lt;--</td>
<td>&lt;--</td>
<td>&lt;--</td>
<td>&lt;--</td>
</tr>
<tr>
<td></td>
<td>NEXT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>VAR</strong></td>
<td>see CON ^</td>
<td>see CON ^</td>
<td>see CON ^</td>
<td>see CON ^</td>
<td>ADD AX,3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PUSH BX</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XCHG AX,BX</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NEXT</td>
</tr>
<tr>
<td><strong>LIT</strong></td>
<td>see CON ^</td>
<td>MOV BX, #</td>
<td>LODSW</td>
<td>LODSW</td>
<td>LODSW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUSH BX</td>
<td>LODSW</td>
<td>LODSW</td>
<td>LODSW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XCHG AX,BX</td>
<td>NEXT</td>
<td>XCHG AX,BX</td>
<td>XCHG AX,BX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NEXT</td>
<td></td>
<td></td>
<td>NEXT</td>
</tr>
<tr>
<td><strong>@</strong></td>
<td>ES:</td>
<td>POP BX</td>
<td>ES:</td>
<td>POP BX</td>
<td>ES:</td>
</tr>
<tr>
<td></td>
<td>NEXT</td>
<td>PUSH [BX]</td>
<td>NEXT</td>
<td>PUSH [BX]</td>
<td>NEXT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>!</strong></td>
<td>ES:</td>
<td>POP BX</td>
<td>ES:</td>
<td>POP BX</td>
<td>ES:</td>
</tr>
<tr>
<td></td>
<td>NEXT</td>
<td>NEXT</td>
<td>NEXT</td>
<td>NEXT</td>
<td>NEXT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>+</strong></td>
<td>POP AX</td>
<td>POP AX</td>
<td>POP AX</td>
<td>POP AX</td>
<td>POP AX</td>
</tr>
<tr>
<td></td>
<td>ADD BX,AX</td>
<td>ADD BX,AX</td>
<td>ADD BX,AX</td>
<td>ADD BX,AX</td>
<td>ADD BX,AX</td>
</tr>
<tr>
<td></td>
<td>NEXT</td>
<td>NEXT</td>
<td>NEXT</td>
<td>NEXT</td>
<td>NEXT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Forth Dimensions** 11 **March 1992 April**
**Observations**

Ignoring the various anomalies, the spread in performance among the Forths for most benchmarks is about a factor of two (about a factor of 1.5 among the test-set versions.) This seems a small gain considering both the efforts that have gone into the various implementations and the resulting lack of internal consistency from one implementation to the next. (It was, however, easier to handle these inconsistencies when writing the various versions of the benchmarks than to handle the inconsistencies among different versions of the various Forths.)

**Specifics**

The data for the D-R-M, D-S-M, I-R-M, and I-S-M versions yields the following ratios:

<table>
<thead>
<tr>
<th>Ratio</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indirect/Direct (Dir)</td>
<td>1.7:1</td>
</tr>
<tr>
<td>Indirect/Reg (Dir)</td>
<td>1.2:1</td>
</tr>
<tr>
<td>I-S-M/D-R-M</td>
<td>1.5:1</td>
</tr>
<tr>
<td>D-R-S/D-R-M</td>
<td>1.1:1</td>
</tr>
</tbody>
</table>

These ratios indicate that changing from indirect-threading to direct-threading in the multi-segment version provides about a 30% speed-up, while changing from top-of-stack on the stack to top-of-stack in a register provides about a 10% speed-up. Changing both provides about a 50% speed-up.

The segment model affects performance in the case shown above by about 5–10% because the D-R-S version does not permit the best possible optimization of the Forth virtual machine for speed (as shown by the code fragments on the preceding page).

A more significant reason for segmentation is that it provides separation of the components of a Forth word and can provide more memory in which to program. For example, separating the headers from the rest of the words can provide more program space or can make an application smaller and much harder to disassemble.

Another reason for segmentation is that more and more operating systems restrict the use of data and code in the same memory “hunk.” These systems normally restrict read/write access to data structures in the code hunk, making an application either use separate hunks for code and data or use the operating system to overcome such restrictions (with possible performance penalties).

**Opinions**

Selecting one Forth over another for a typical gain of 50% in performance may be the wrong reason to make the choice. Changing from an 8086 to a faster member of the family, changing an algorithm, or using the optimizers available with several of the Forths can result in gains of from three to more than 30.

The following considerations would seem at least as important:

- quality and completeness of the implementation,
- availability and appropriateness of additional modules,
- availability and quality of support including documentation,
- transportability of source and ease of use,
- application-size supported.

Notice that price is not in the above list. If you are going to use the Forth for a commercial application, even the highest priced commercial Forth is inexpensive if it has features that are important to your application and will allow you to finish your project significantly faster than you otherwise would.

A particular consideration these days is the size of the application supported. Most commercial applications are big and growing bigger, especially those that have to run under most of the current graphical user interfaces. The typical single-segment Forth, even with overlays, is hard pressed to support the bloated programs that seem to be required. (Even embedded systems are getting larger, although minimizing their size is still very important.)

Most of the Forths reviewed do not easily support large programs and among those that do, there are a variety of trade-offs that need to be considered. Some of the Forths that seem to support large programs have limitations on the space available for code and/or data, others do not. Some require significantly more memory for a given application than others. The segmentation information and the code-fragments presented for the Forths provide some insight as to the advantages and limitations of the various Forths.

Another consideration that is becoming more important, at least in the PC world, is the ease with which foreign libraries and facilities (DLLs, OLE, etc.) can be accommodated. Most of the Forths reviewed have no built-in capability, a few do. If this is an important consideration, one should investigate the support for interfacing to other programs and libraries that may be available.

Most of the Forths reviewed claim to support multitasking. If this is an important feature, be warned that the support provided is usually minimal. Further, almost none of these Forths provide useful multiuser support.

**Forth**

For those wishing to evaluate Forth, important considerations include ease of use (including DOS interface and available editors), standardization, and adherence to available Forth texts.

Another consideration that is important when considering a Forth is whether you are going to approach it as a black box, or whether you are interested in understanding its internal structure. To obtain the maximum advantage from Forth, one should understand both the rational for its structure and its inherent strengths and weaknesses. This requires at least some understanding of the internals of the version being used and becomes more important as an application becomes more complex. The Forths reviewed...
range from simple to very complex and the documentation of their structure ranges from nonexistent to well detailed.

Further, some provide complete source code and some do not (although you can usually obtain it for a fee). At the most advanced level, those that supply source provide it either as native Forth code with a metacompiler or as assembly code for use with a standard assembler. Be warned, most metacompilers are difficult to master at best and you usually require some understanding of them to follow the accompanying Forth source.

Finally, remember... Forth can never (well, hardly ever) be too small or too fast—especially for all those big and slow applications.

**Forth Assemblers** (an aside)

How to move the contents of memory (pointed to by the contents of register BX) into a register (AX in this case):

- MOV AX, [BX]  opcode destination source
- AX, [BX] MOV  destination source opcode
- AX [BX] MOV.  same order, trailing period
- [BX] AX MOV   source destination opcode
- [BX] AX MOV,  same order, trailing comma
- [BX] AX LDA,  same order, different opcode
- 3) 0 MOV,    same order, different register "names"
- 3) 0 LDA     same order, different opcode

and there are probably more (and you haven't seen how to index yet!).

The code fragments are all given in a standard format. This does not reflect the flavor of the assembler mnemonics of the various Forths studied (as hinted at above) but does make it easier to understand and compare the examples.

### How to open a file and load a program in the various Forths.

<table>
<thead>
<tr>
<th>Forth</th>
<th>Case</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBL</td>
<td>screen-file</td>
<td>CACHE-NAME 30 EXPECT &lt;cr&gt; BBLBENCH &lt;cr&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 CACHE-NAME 8 + ! &lt;cr&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OPEN-CACHE 1 LOAD &lt;cr&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(my old version did not have USING)</td>
</tr>
<tr>
<td>eForth</td>
<td>text-lines</td>
<td>via serial channel</td>
</tr>
<tr>
<td>F83</td>
<td>screen-file</td>
<td>OPEN F83BENCH. 1 LOAD &lt;cr&gt;</td>
</tr>
<tr>
<td>F-PC</td>
<td>text-file</td>
<td>FLOAD BENCH &lt;cr&gt;</td>
</tr>
<tr>
<td>Fifth</td>
<td>text-file</td>
<td>L SIEVE.FIV &lt;cr&gt; C &lt;cr&gt;</td>
</tr>
<tr>
<td>HS/FORTH</td>
<td>text-file</td>
<td>FLOAD HSFBENCH &lt;cr&gt;</td>
</tr>
<tr>
<td>LaForth</td>
<td>text-file</td>
<td>LA LA.HI &lt;cr&gt; (from DOS) LT RUN &lt;cr&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BT MT TEXT LABENCH^Z BT OPEN . (handle)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BT size handle READ TP +1 &lt;cr&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 LT DROP TP @ XC! LT RUN &lt;cr&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(couldn't find a better way - must be one?)</td>
</tr>
<tr>
<td>MMSFORTH</td>
<td>screens</td>
<td>400 LOAD &lt;cr&gt; (non-DOS, see MMSBENCH)</td>
</tr>
<tr>
<td>MVP-FORTH</td>
<td>screens</td>
<td>342 (or 171) LOAD &lt;cr&gt; (see MVPBENCH)</td>
</tr>
<tr>
<td>PC-Forth</td>
<td>screen-file</td>
<td>INCLUDE PCBENCH &lt;cr&gt; or</td>
</tr>
<tr>
<td></td>
<td>text-file</td>
<td>INCLUDE PCBNECH &lt;cr&gt;</td>
</tr>
<tr>
<td>PMFORTh</td>
<td>screen-file (in PMfile)</td>
<td>OPEN B:PBENCH &lt;cr&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 SFLOAD &lt;cr&gt;</td>
</tr>
<tr>
<td>polyFORTh</td>
<td>screen-file</td>
<td>CHART PCBENCH 1201 LOAD &lt;cr&gt; or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 LOADUSING PCBENCH &lt;cr&gt;</td>
</tr>
<tr>
<td>Pygmy</td>
<td>screen-file</td>
<td>NAMEZ: PYGBENCH &lt;cr&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>600 PYGBENCH 2 UNIT &lt;cr&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 OPEN 1 LOAD &lt;cr&gt;</td>
</tr>
<tr>
<td>riFORTh</td>
<td>screen-file</td>
<td>RIFORTH RIBENCH &lt;cr&gt; (from DOS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 LOAD &lt;cr&gt; (screens start from 1)</td>
</tr>
<tr>
<td>UniForTh</td>
<td>screen-file</td>
<td>UNIFORTH UNIBENCH &lt;cr&gt; (from DOS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 LOAD &lt;cr&gt;</td>
</tr>
<tr>
<td>Upper Deck</td>
<td>text-file</td>
<td>CAPS ON RELOAD BENCH &lt;cr&gt;</td>
</tr>
<tr>
<td>UR/ForTh</td>
<td>screen-file</td>
<td>ASM USING LMIBENCH. 1 LOAD &lt;cr&gt;</td>
</tr>
<tr>
<td>ZEN</td>
<td>text-file</td>
<td>INCLUDE ZENBENCH. &lt;cr&gt;</td>
</tr>
</tbody>
</table>
Segments (max. size each):
Code(64K) Lists+Data(1 meg.) Heads(n*64K)
Stack+Block(64K)

Register use:
AX = W SI = IP CS = code
BX = tos(lsw) DI = 0 (litt) DS = lists+data (seg/off of)
CX = tos(msw) BP = RP ES = lists+data (32 bit addr)
DX = SP SP = SS SS = stacks

Next
LODSW XCHG SP,BP
JMP AX POP DS
Push SI
Push DS POP SI
XCHG SP,BP XCHG SP,BP
MOV DX,xxx_pfa(seg) NEXT
MOV SI,xxx_pfa(offset) MOV DS,DX NEXT

Nest (bigger, faster)

Unnest
XCHG SP,BP
XCHG SP,BP
PUSH SI
POP DS
Push DS
Pop SI
XCHG SP,BP
MOV AX,xxx_pfa(off) JMP DOCOL
MOV DS,AX
MOV SI,AX

@ not implemented

MOV ES,CX
MOV CX,ES:[BX+2] MOV ES:[BX]
MOV CX,ES:[BX] POP ES
POP BX
MOV CX,xxx_pfa(seg) ADD BX,AX
MOV CX,xxx_pfa(off) ADC CX,DX

Nest (slower, smaller)

Unnest
MOV DS,xxx_pfa(seg)
MOV AX,xxx_pfa(off)
JMP DOCOL

Notes: compiled tokens are offsets into the code segment.

1. Not for military use.

Constant (in-line)
Variable
PUSH BX POP DX
PUSH CX POP AX
MOV BX,#(lsw) MOV CX,xxx_pfa(seg)
MOV CX,#(msw) MOV BX,xxx_pfa(off)

Nest
NOP CALL NEST

Unnest
XCHG SP,BP
XCHG SP,BP
PUSH SI
POP DS
XCHG SP,BP
XCHG SI,BP
POP SI
NEXT

Literal
LODSW
PUSH AX

Next

Nest not implemented
Variable
NOP CALL NEST

doVAR

+ : UM+ DROP ;

Notes: All variables are user variables, UP is in memory, FOR NEXT loop instead of DO LOOP, CATCH and THROW are used in error recovery.

BBL
V1? 10/25/86, Green

Written by Roedy Green to use as the tool for rewriting Abundance (a vast database program and application). Source code for BBL is in assembler.

A direct-threaded 32-bit implementation with the top-of-stack in a register. A multiple-segment model which interprets from screen files.

March 1992

14

Forth Dimensions
F83
V2.4.0 3/24/87, Laxen & Perry

Written by Henry Laxen and Mike Perry to provide a working model of an 83 Standard Forth. Released with many enhancements over Fig-FORTH and available for 8080/8086 processors. Includes full source code and metacompiler in DOS screen files.

An indirect-threaded 16-bit implementation with the top-of-stack on the stack. A single-segment model which interprets from screen files.

Notes: Central NEXT.

Segments (max. size each):
Code+Lists+Data+Heads+Stack+Blocks(64K)

Register use:
AX = - SI = IP CS = all
BX = W DI = - DS = CS
CX = - BP = RP ES = CS
DX = - SP = SP SS = CS

Next ____________ Nest ____________ Unnest
LODSW INC BX INC BX MOV SI,[BP]
MOV BX,AX INC BX INC BP
JMP [BX] DEC BP INC BP
MOV [BP],SI INC BP
MOV SI,BX JMP NEXT

Notes: Central NEXT.

F-PC
V3.50 10/22/89, Zimmer & Smith

A massive effort (and implementation) by Tom Zimmer and Robert L. Smith (with support from a variety of other persons and groups). Many enhancements over F83 and a large set of contributed add-ons by other programmers. Has a very complete text-editor and hyper-text-like source-code and documentation browser. Very big and very complete, includes full source code and metacompiler.

A direct (segment) threaded 16-bit implementation with the top-of-stack on the stack. A multiple-segment model which interprets from text files.

Notes: colon definitions start on paragraph boundaries.
Segments (max. size each):
Code(n*64K) Lists+Data(n*64K) Heads(n*64K) Stack(64K)

Register use:
AX = - SI = IP CS = code
BX = tos DI = W DS = lists+data
CX = - BP = RP ES = heads/misc
DX = - SP = SP SS = stacks

Next_____ Nest_____ Unnest
LODSW INC BP MOV SI,[BP]
XCHG DI,AX INC BP DEC BP
JMP [DI] MOV [BP],SI DEC BP
LEA SI,[DI+2] NEXT

@_________ !_________ Literal
MOV BX,[BX] POP AX PUSH BX
NEXT MOV [BX],AX MOV BX,[SI]
POP BX INC SI
NEXT INC SI
NEXT

Constant____ Variable____ ±
PUSH BX PUSH BX
MOV BX,[DI+2] LEA BX,[DI+2]
NEXT ADD BX,AX
NEXT

Segments (max. size each):
Code(64K) Lists(64K) Data(64K) Heads(64K) Stacks(64K)
Tool(64K) Video(64K) Msgs(1K)

Register use:
AX = W SI = IP CS = code
BX = tos DI = - DS = lists
CX = - BP = RP ES = data
DX = - SP = SP SS = stacks

Next_____ Nest_____ Unnest
LODSW DEC BP MOV SI,[BP]
JMP AX DEC BP INC BP
MOV [BP],SI INC BP
MOV SI,pfa NEXT

@_________ !_________ Literal
MOV BX,ES:[BX] POP ES:[BX] PUSH BX
NEXT POP BX MOV BX,value
NEXT

Constant____ Variable____ ±
PUSH BX PUSH BX
MOV BX,value POP AX
NEXT MOV BX,addr
NEXT ADD BX,AX
NEXT

HS/FORTH
V4.24 8/9/91,
Harvard Softworks
commercial
A very complete commercial implementation of
Forth for the 8086 family of microcomputers. One of the
few Forts in this review that
provides compatibility with
the DOS linker. Source code
and meta-compilers available.
Multiple Forth segments in a
single DOS allocation.

An indirect-threaded 16-bit implementation with the
top-of-stack in a register. A
multiple-segment model which interprets from text or
screen files.

KForth
V0.9 9/28/91, Kelly
copyrighted
Currently an experimental model to investigate various aspects of threading and
segmentation. Current version is fast ("in-line") direct-threaded, multi-segment (in
multiple DOS segments).

A direct-threaded 16-bit implementation with the top-of-stack in a register. A mul-
tiple-segment model which interprets from screen files.

Notes: Colon-word PFA's, literals, constants, and variable addresses are compiled
"in-line" in the code segment.
LaFORTH
V4.0 9/24/87, Stuart & Smith
copyrighted

Experimental version by
LaFarr Stuart and Robert L
Smith. Has some very inter-
esting features (including
calling Forth from Forth and
interpreting a word-at-a-time
instead of a line-at-a-time).
Source code is in assembler.

A direct-threaded 16-bit
implementation with the top-
of-stack on the stack. A single-
segment model with an extra
segment for interpreting text
files.

Notes: the return stack is in
the ES segment and grows
"up."

Segments (max. size each):
Code+Lists+Data+Heads+Stack(64K)  Text(64K)

Register use:

AX = -  SI = IP  CS = code+lists+data+p-stack
BX = W  DI = RP  DS = CS
CX = -  BP = -  ES = CS+1000H txt-buf+r-stack
DX = -  SP = SP  SS = CS

Next____  Nest______  Unnest____

LODSW  NEST:  ADD AX, 3  SUB DI, 2
JMP AX  XCHG AX, SI  MOV SI, ES:[DI]
           STOSW  NEXT

@________  !________  Literal

POP BX  POP BX  LODSW
PUSH [BX]  POP AX  PUSH AX
NEXT  MOV [BX], AX  NEXT

Constant  Variable  ±

CALL @  CALL @  POP AX
          POP BX  ADD AX, BX
           PUSH AX  NEXT

MMSFORTH
V2.4 5/30/85,
Miller Microcomputer Svs.
commercial!

Commercial version of
Forth includes advanced full-
screen editor, many utilities.
Options include database,
word-processor, general
ledger, expert system, and
advanced utilities. Source-
code is in screen files in DOS
version and in direct blocks
(screens) in self-booting ver-
(which supports more
efficient Forth disk formats
such as 1K sector size). Most
source-code is supplied, full
source-code and metacom-
piler are available.

Indirect-threaded, 16-bit
implementation, top-of-stack
on stack, single-segment
model (DOS version uses a
separate Heads segment).
Interprets from direct blocks
(DOS version uses screen-
files.)

Segments (max. size each):
Code+Lists+Data+Heads+Stack+Blocks(64K) (non-DOS version)

Register use:

AX = -  SI = IP  CS = all
BX = W  DI = -  DS = CS
CX = -  BP = RP  ES = CS
DX = -  SS = SP  SS = CS

Next____  Nest______  Unnest____

LODSW  DEC BP  MOV SI, [BP]
XCHG AX, BX  DEC BP  INC BP
JMP [BX]  MOV SI, [BP]  INC BX
             INC BX  INC BX
             MOV SI, BX  NEXT
             MOV SI, BX

@________  !________  Literal

POP BX  POP BX  *
PUSH [BX]  POP [BX]  *
NEXT

Constant  Variable  ±

*  POP AX
   POP BX  ADD AX, BX
   PUSH AX  NEXT

Note: Non-DOS version of MMSFORTH was used in Excalibur's SAVVY, DOS version in
Lindberg System's OMNITERM-2 and Ashton-Tate's RAPIDFILE.

* High-level words, source code provided.
1. The above information is presented with the generous permission of A. Richard (Dick)
Miller of Miller Microcomputer Services.
MVP-FORTH
V1.0405.03 5/17/85, MVP  public
One of the first 79-Standard Forths. Based on the fig-FORTH 8086 implementation model. Source code in direct blocks (screens).

An indirect-threaded, 16-bit implementation with the top-of-stack on the stack. A single-segment model which interprets from direct blocks (screens).

Notes: NEXT and NEST seem to be a direct translation of the fig-FORTH1860 assembly code. Central NEXT.

PC-Forth
V1.56 9/09/87, Kelly  public
Written expressly for teaching purposes. Includes full source code and a simple, interactive metacompiler (which does not require mixing meta-commands in with the Forth source code). Also includes a very powerful screen editor (with overlay capability for use with other Forths), as well as many utilities. Available in both self-booting and DOS versions.

An indirect-threaded 16-bit implementation with the top-of-stack on the stack. A single-segment model which interprets from screens (direct or file) or text files.

March 1992
**polyFORTH**

*pF86/MSD, FORTH Inc.
commercial*¹

The mother of all Forths
(well, almost) by FORTH, Inc. Complete source code
with metacompiler, EGA/ VGA graphics, data-base,
floating point, screen editor, debugger and other support.
Full multi-user capability built in at the kernel level. Source
code and shadow screens in screen files.

Indirect-threaded, 16-bit implementation, top-of-stack
on stack, multiple-segment model. Interprets from screen
files. A 32-bit 386 protected-mode version is also available.

Note: the reported benchmarks were done on pF86/MSD (which is a single-segment version
dated 1/20/87). The newer, multi-segment version detailed above should produce the same
or only slightly different times.

¹. The above information is presented with the generous permission of Elizabeth Rather of
FORTH, Inc.

**Pygmy**

*V1.3 10/4/90, Sergeant
copyrighted*

Based on the Chuck
Moore cmFORTH model. The source code and the
metacompiler are in screen file.

A direct-threaded 16-bit implementation with the top-of-stack in a register. A
single-segment model which interprets from screen files.
riFORTH
V1? 1990, Illyes copyrighted

A minimalist Forth; fast and efficient. Full source code and metacompiler in about 15 screens. Does some optimization. Interesting!

A subroutine-threaded, 16-bit implementation with the top-of-stack in a register. A single-segment model which interprets from screen files.

riFORTH Copyright Robert F. Illyes, 1990. My thanks to Robert Illyes for publishing the source code for riFORTH. The availability of a complete subroutine-threaded Forth, in only 12 screens, made it possible to clone the five different versions used in this study.

Segments (max. size each):
Code+Lists+Data+Heads+Stacks+Blocks(64K)

Register use:
AX = - SI = SP      CS = all      IP = IP
BX = tos DI = -     DS = CS
CX = - BP = -       ES = CS
DX = - SS = RP      SS = CS

Next__ Neat____ Unneat
CALL xxx RET

\[\begin{align*}
\theta \quad & (XCHG \ SP, SI) \* \\
\text{ Literal} & (XCHG \ SP, SI) \* \\
\text{MOV BX, [BX]} & \text{POP [BX]} \\
\text{CS} & \text{value} \\
\text{Variable} & \text{addr} \\
\text{PUSH BX} & \text{ADD BX, AX} \\
\text{MOV BX, value} & \text{ADD BX, AX} \\
\text{AX} & \text{IP} \\
\text{DI} & \text{W} \\
\text{BP} & \text{SS} \\
\text{SS} & \text{Sp} \\
\text{Next} & \text{Unneat} \\
\text{MOV DI, pfa} & \text{PUSH AX} \\
\text{JMP NEST} & \text{POP AX} \\
\text{LODSW} & \text{MOV SI, [BP]} \\
\text{JMP AX} & \text{INC BP} \\
\text{DEC BP} & \text{INC BP} \\
\text{MOV [BP, SI]} & \text{NEXT} \\
\text{MOV SI, DI} & \text{NEXT} \\
\text{NEXT} & \text{ADD BX, AX} \\
\text{PUSH BX, [BX]} & \text{ADD BX, AX} \\
\text{doVAR: PUSH BX} & \text{ADD BX, AX} \\
\text{MOV BX, [DI]} & \text{ADD BX, AX} \\
\end{align*}\]

* When these words are compiled in-line, these instructions may be eliminated.


Segments (max. size each):
Code(64K) Lists(64K) Data(64K) Heads(64K) Stacks(64K)

Register use:
AX = W SI = IP      CS = code
BX = tos DI = W'     DS = lists
CX = - BP = RP       ES = data
DX = - SS = SP      SS = stacks

Next__ Neat____ Unneat
MOV DI, pfa RET
JMP NEST

\[\begin{align*}
\theta \quad & (XCHG \ SP, SI) \* \\
\text{ Literal} & (XCHG \ SP, SI) \* \\
\text{MOV BX, [BX]} & \text{POP ES} \\
\text{CS} & \text{addr} \\
\text{Variable} & \text{SP} \\
\text{PUSH BX} & \text{ADD BX, AX} \\
\text{MOV BX, value} & \text{ADD BX, AX} \\
\text{AX} & \text{IP} \\
\text{DI} & \text{W} \\
\text{BP} & \text{RP} \\
\text{SS} & \text{Sp} \\
\text{Next} & \text{Unneat} \\
\text{MOV DI, addr} & \text{PUSH AX} \\
\text{JMP doVAR} & \text{POP AX} \\
\text{LODSW} & \text{MOV SI, [BP]} \\
\text{JMP AX} & \text{INC BP} \\
\text{DEC BP} & \text{INC BP} \\
\text{MOV [BP, SI]} & \text{NEXT} \\
\text{MOV SI, DI} & \text{NEXT} \\
\text{NEXT} & \text{ADD BX, AX} \\
\text{PUSH BX, [BX]} & \text{ADD BX, AX} \\
\text{doVAR: PUSH BX} & \text{ADD BX, AX} \\
\text{MOV BX, [DI]} & \text{ADD BX, AX} \\
\text{NEXT} & \text{ADD BX, AX} \\
\end{align*}\]

Notes: When case sensitive, all supplied words are lower case.

1. The above information is presented with the generous permission of Peter Graves of Upper Deck Systems.
UR/FORTH
V1.1 3/11/90, LMI commercial

UR/FORTH is one of the few Forths in this review that is compatible with the DOS linker. It is well supported, with many extensions and a very good screen-oriented editor. Most source code is provided in screen files. Complete source is available.

A direct-threaded 16-bit implementation with the top-of-stack in a register. A multiple DOS-segment model which interprets from screen or text files. Also available in OS/2 1.x, 386 32-bit protected mode, and Windows implementations (which are compatible, at Forth language level, with the DOS version).

Note: supports binary overlays.

1. The above information is presented with the generous permission of Ray Duncan of Laboratory Microsystems, Inc.

2. Segment model for version tested, varies with implementation.

---

ZEN
V1.5a 4/2/91, Tracy copyright

Currently (Sept. 1991) ZEN is the only Forth in this review that is tracking the X3J14 basis document. It has fully ROMmable assembler source code and an interface to one of the standard programmers text editors.

A direct-threaded 16-bit implementation with the top-of-stack in a register. A multiple-segment model which interprets from screen or text files.
The Curly Control Structure Set

Kourtis Giorgio
Genoa, Italy

AUTHOR
(smilinig)
Hi! I've got a new proposal on a new complete set of control structures.

READER
(annoyed)
Another proposal? Do you know that this is the 134th Forth proposal on extensions, expansions, additions, etc., to control structures?

AUTHOR
(less smilinigly)
Yes, but mine...

READER
(smilinig)
Oh, yes! Sure, yours is better, includes as subcases all previous proposals, is original, has support for errors, is coherent, etc.

AUTHOR
(happy)
Exactly!

READER
(serious)
You know that the same thing has been claimed by 59 other articles?

AUTHOR
(aggressive)
Yes, I know, I have read every article. Many are very interesting and have been important for me. I copied everything that could be copied, I took every good idea that has appeared, I tried to unify solutions, I attempted to solve all the problems I was aware of,* I tried to render uniform the proposed set of words, I sacrificed strict historical continuity to improve teachability while avoiding conflicts among old and new syntaxes. And I have used them for more than two years, refining them until they were stable enough.

* except the interactivity of control-flow words.
These control-flow words can be used any number of times in any combination, and also can be used in secondaries called from within the control structure—allowing, for example, WHILE to be defined as

```forth
WHILE ( flag -- ) 0= WHEN ;
```

Depending on the implementation, this feature may be available or not. While not often used, it is a rarely available but interesting feature that can provide new possibilities.

Looking again at Figure One-a, we see that apart from LEAVE, WHEN, WHILE, AGAIN, and ?AGAIN, there are words that must be used in pairs:

```forth
WHEN{ ( flag -- ) WHEN) ( -- )
WHILE{ ( flag -- ) WHILE} ( -- )
```

In brief, if `xxxx` is any control structure,

```forth
xxxx{ ... WHEN{ someCode WHEN} ... xxxx}
```

is equivalent to

```forth
xxxx{ IF someCode LEAVE THEN ... xxxx}
```

and, similarly,

```forth
xxxx{ WHILE{ someCode WHILE} ... xxxx}
```

is equivalent to

```forth
xxxx{ ... 0= IF someCode LEAVE THEN ... xxxx}
```

By using a WHEN pair, we not only can test a condition to decide if the control structure must be left, we can also place between WHEN and WHEN the code that must be executed just before leaving the control structure. This way, any leave-test point can be equipped with its own, specific pre-exit code.

Using WHEN is, thus, the equivalent of writing WHEN { and WHEN} with no code between them.

To illustrate figuratively, let's say that:

```forth
xxxx{ WHEN{ someThing WHEN} xxxx}
```

may be informally written as

```forth
xxxx{ WHEN xxxx}
```

Other pairs of words that we can use inside an indexed control structure are:

```forth
OF{ ( value -- )
OF} ( -- )
```

```forth
WITHIN{ ( lower upper -- )
WITHIN} ( -- )
```

```forth
IN{ ( x1 x2 ... xN -- )
IN} ( -- )
```

These pairs make explicit reference to the index and are mainly used in the internals of a CASE structure (though it may be used in any indexed control structure). Look at Figure One-b for a definition of their workings.

Inside indexed control structures, three words are supplied to reference the index and its value:

```forth
I ( -- indexValue )
```

Leaves on the stack the value of the index.
TO-I (newValue --)
Stores into the index a new value.

STEP (valueToAdd --)
Adds to the index a value (stepping it). STEP is equivalent to I + TO-I.

Description of Curly Control Structures
Given the previous framework, now I am going to illustrate briefly the use of each of these control structures and how they work. Later, we will see some examples.

CONTROL{ ( -- )
CONTROL} ( -- )

The word CONTROL{ marks the beginning of the control structure. The word CONTROL} marks the end. They may be used as label points when jumping via words like WHEN, WHILE, AGAIN, ?AGAIN, and word pairs like WHEN { ... WHEN}, etc.

ANDIF structures are easily implementable with these ingredients. The word CONTROL{, if reached, exits the control structure.

REPEAT{ ( -- )
REPEAT} ( -- )

Equivalent to—but much more flexible than—the usual BEGIN UNTIL or BEGIN WHILE REPEAT structures. Any number of WHILES or WHENs may be used inside this structure, along with the pairs WHEN WHEN, etc.

CASE{ ( KeyValue -- )
CASE} ( -- )

Comparable to Eaker's CASE structure. CASE{ takes a number from the stack (the subject of our research) and puts it into the index, where it can be retrieved by I ( -- KeyValue )
Afterwards, by means of pairs like OF{ ... OF}, WITHIN [ ... WITHIN], IN[ ... IN], and others yet to be invented, and also by using WHEN[ ... WHEN] or WHILE[ ... WHILE], we can select and perform the desired action. If the CASE} word is reached, the control structure is left.

FOR{ ( initialValue -- )
FOR} ( -- )

General-purpose looping construct (unlike a FOR ... NEXT definite loop). FOR{ takes a number from the stack and puts it into the index. Afterwards, the index may be manipulated by words like I, TO-I, and STEP. The loop-termination condition must be handled explicitly using WHILE or WHEN.

This is an imitation of the C language's FOR construct. The desired model of loop (pre-increment, pre-decrement, post-increment, post-decrement, fixed or variable step, etc.), must be handled explicitly—checking and incrementing/decrementing at the beginning or end of the loop, depending on the desired behavior. If FOR} is reached, the loop is repeated.

TIMES{ ( #times -- )
TIMES} ( -- )

Similar to the DO ... LOOP construct. Takes a number from the stack (there exists an error condition if the number is negative) and puts it into the index.

a) Before every iteration, the value of the index is checked. If the control structure is left and execution continues after TIMES). Otherwise, I is decremented and execution continues after TIMES{ ( beginning step -- )
LOOP} ( -- )

Start specifies the initial value of the index, #times specifies the maximum number of times the loop must be done (it could end prematurely due to words like LEAVE, WHEN, and WHILE).

If #times is negative, there is an error condition. Step and #times are put on the return stack, and start is put into the index (also on the return stack).

a) Before every iteration, the #times is checked. If it is 0, the control structure is left and execution proceeds after LOOP). If #times is not 0, it is decremented and an iteration begins.

If and when LOOP} is reached, the step is added to the index, then the process is repeated from point a).

Thus,

10 ( beginning )
4 ( times )
Along with the loop structure are furnished the four parameter-modifying words END, END, SIZE, and BACK. These words allow you to specify, in many different ways, the order and the set of values that must be spanned by the index during the loop.

**Examples and Test Suites**

The above set of control-flow words has been presented by figures and somewhat by words. Examples are important for two reasons:
- Clarify obscure or dubious points.
- Given the fact that only the constructing elements have been shown, provide some interesting combinations of them. (Some, probably, haven't even been explored yet.)

**CONTROL**

Example: Test whether the three variables A, B, and C contain 0. (The so-called ANDIF construct.)

```
: ABC_allZero?
  CONTROL{ A @ 0= WHILE
    B @ 0= WHILE
    C @ 0= WHILE
    "A,B,C contain 0" CONTROL} ;
```

**REPEAT**

Example One: Traverse a list until a 0 list terminator is found.

```
( addr -- ) REPEAT{
  DUP @ WHILE REPEAT} ( lastAddr )
```

or, equivalently:

```
( addr -- ) REPEAT{
  DUP @ 0= WHEN @ REPEAT
}
```

Example Two: Given the address of a null-terminated string, leave the address of the first space or the EndAddr (if no spaces are found).

```
( addr -- ) REPEAT{
  DUP C@ ?DUP
  WHILE BL = WHEN 1+ REPEAT
}
```

**CASE**

Example One: Take a character code from the stack and qualify it.

```
: ?WhatCharItIs? ( char -- )
  CASE{ ( pop char from stack and put into index) ." The character with code" I ." is "
```

Note: The two words (( and )) may be defined to count the elements of a set of numbers. So, instead of writing

```
ASCII A ASCII Z []
WITHIN{ ." an upper-case letter" WITHIN}
ASCII a ASCII z []
WITHIN{ ." a lower-case letter" WITHIN}
\ The word [] means simply 1+
```

```
ASCII + ASCII - ASCII *
ASCII / ASCII ^ 5
\ 5 specifies that +, -, *, /, and ^ are 5.
IN{ ." an arithmetic operator" IN}
ASCII ( ASCII { ASCII { 3
IN{ ." an opening parenthesis" IN}
ASCII ) ASCII ) ASCII ) 3
IN{ ." a closing parenthesis" IN}
BL OF{ ." SPACE char" OF}
  0 OF{ ." NULL char" OF}
ASCII 0 ASCII 9 []
WITHIN{ ." a decimal digit" WITHIN}
[ HEX ] 80 100
WITHIN{ ." a graphical character. One of those " cr
.. with code between 128 included and 256 excluded" WITHIN}
I 100 >= I 0 < OR
WHEN{ ." outside the character range" WHEN}
I -20 AND
WHILE{ ." a control character" WHILE}
  \ -20 = not (11111B)
."
```

Example Two: Special use of the case structure which allows for subcases. Pairs used inside the outermost WITHIN, if entered, leave the CASE structure and execution continues after CASE) (just like external pairs).

```
: DISASSEMBLE ( inst -- )
  CASE{
    0000 4000 WITHIN{
      I MoveInst WITHIN
```
CASE ResultDisplay ;

FOR
Example: Type the powers of two smaller than 1.000.000
1 FOR{ I 1.000.000 < WHILE
I .
I 2* TO-I ( or I STEP ) FOR}

Example: Type the contents of a null-terminated list.
( StartAddr -- )
FOR{ I WHILE
I CELL+ @ .
I @ TO-I FOR}

Example: Do a loop that executes at least once (like DO LOOP). Only for illustration purposes.
10 FOR{ someCode
5 STEP I 90 <
WHILE FOR}
90 10 DO someCode
5 +LOOP

TIMES
: DROPS
( X1 X2 ... Xn n -- )
TIMES{ DROP TIMES} ;

: MULTIEXECUTE
( token @times -- )
TIMES{ DUP EXECUTE TIMES} DROP ;

: CountBack (from --)
TIMES{ I . TIMES} ;

Presenting
|LEAVING{ and }COMPLETED|

While the subset of words discussed above resolves many problems, until now some are still unresolved. Here I will present the problems and the solutions to them using extensions of the wordset.

The |LEAVING{ Clause

Let's suppose there are three variables V0, V1, and V2 that contain addresses of strings, and we want to know if at least two among them are equal. Using the set presented so far, here is a solution:

Figure Two. Compilation effects of CONTROL, REPEAT, FOR, and CASE.

CONTROL

| LEAVING{ leavc code CONTROL|

REPEAT

| LEAVING{ leavc code REPEAT|

FOR & CASE
CASE and FOR compile similarly to CONTROL and REPEAT, respectively, but with (SIMPLE ) replaced by (INDEXED ).
CONTROL{ V0 @ V1 @ $= WHEN{
   ." At least two equal" WHEN
   V1 @ V2 @ $= WHEN{
      ." At least two equal" WHEN
   V2 @ V0 @ $= WHEN{
      ." At least two equal" WHEN
   ." All different"

CONTROL}

This solution, however, is redundant and wasteful of space. At the expense of computational time, we could choose to check all three equalities by ORing them together at the end and using an IF ELSE THEN control structure. Here, I'll show a third solution that is neither redundant nor slow.

CONTROL{ V0 @ V1 @ $= WHEN
   V1 @ V2 @ $= WHEN
   V2 @ V3 @ $= WHEN
   ." All different"
}LEAVING{ ." At least two equal"

CONTROL}

The previous solution and its meaning may be explained in more general terms.

Let XXXX be the name of a control structure (CONTROL, TIMES, etc.). Let's call the pair of words like WHEN [. . . WHEN], WHILE [. . . WHILE], OR{ . . . OR}, etc. as specialized leaving points, while words like WHEN and WHILE are unspecialized leaving points. Thus, the code in Figure Seven is logically equivalent to that in Figure Eight.

In other words, suppose we have a control structure with a }LEAVING() embedded in it. If we transform all unspecialized leaving points into the corresponding specialized pair—inserting into the specialized pair the code contained between the original }LEAVING() and the end of the control structure, then deleting the code from }LEAVING() (exclusive) to the end of the control structure (exclusive)—we obtain new code that is logically equivalent to the original. (Again, refer to Figures Seven and Eight.)

Another example will better clarify these concepts. Suppose we have a null-terminated string and must scan it for the first occurrence of the character +, -, or...

-- Figure Three. TIMES and LOOP compilation effects. --

.. If such an occurrence is found, we must substitute a space for it and leave on the stack (SubstitutionAddr+1 true); otherwise, we must leave a (false) on the stack. Here is a solution using }LEAVING():

( StringAddr)
FOR{ I C @ WHILE{ false WHILE}
   I C @ ascii + = WHEN
   I C @ ascii - = WHEN
   I C @ ascii . = WHEN 1 STEP
}LEAVING{ bl I C! I 1+ true FOR)

Maybe the above would read more clearly if written informally as:

FOR{ I C @ WHILE{ false WHILE}
   I C @ ascii + = ORIF
   I C @ ascii - = ORIF
   I C @ ascii . = ORIF 1 STEP
}ORWHEN{ bl I C! I 1+ true FOR)

*The }COMPLETED/ Clause*

A similar problem is encountered in definite loops. A definite loop like TIMES or LOOP may end for two reasons: 1. Premature end due to "leavers" like WHEN, WHILE, LEAVE,
2. Exhaustion of the number of looping times specified.

Depending on the reason why execution left the loop, different behaviors can be requested. Let’s explain via example.

Suppose we must search an editor’s text for a specific character. We want the cursor to move during the search and, if the character is found, have it point to that character; otherwise, we want to reset the cursor to its original position, not simply leaving it at the end of the text.

Below is a word to do that, using the new word \COMPLETED\.

: CharSearch ( char -- true | false )
cursor @ swap \ keep previous position
TextEnd @ cursor @ -
\ # of chars to end of text
TIMES{ dup ( ... char char )
NextCharGet ( char char textChar )
\ Move the cursor on
\ while furnishing the \ pointed char.
= WHEN{ 2drop true WHEN}
\COMPLETED{ drop ( initialCursorPosition)
Cursor! false
TIMES} ;
\ Compare with the flowchart in Figure Five.

\LEAVING{ and \COMPLETED{ can be used together.
The next example is a variation of the previous one. Here we want to search our text for the first occurrence of \, [\, or \. (See the flowchart in Figure Six.)

: CharSearch
cursor @ \ keep initial cursor position
textEnd @ cursor @ -
\ calculate # chars to end of text
TIMES{ charGet
dup ascii ( = when
dup ascii [ = when
dup ascii ] = when
drop

TIMES{ TIMES{
dup NextCharGet =

WHEN true
2drop true

false
DONE

\ COMPLETED{ drop cursor ! false TIMES}

\ COMPLETED{ Cursor! false
\ Restore initial cursor position.
\ LEAVING{ 2drop true
\ Drop initial cursor position...
\ ...and character under cursor.
TIMES} ;

\LEAVING{ and }COMPLETED{ can be used together.
The next example is a variation of the previous one. Here we want to search our text for the first occurrence of \, [\, or \. (See the flowchart in Figure Six.)

: CharSearch
cursor @ \ keep initial cursor position
textEnd @ cursor @ -
\ calculate # chars to end of text
TIMES{ charGet
dup ascii ( = when
dup ascii [ = when
dup ascii ] = when

\LAYOUT{Figure Four. Return-stack, control-structure frame.}

\begin{verbatim}
Figure Four. Return-stack, control-structure frame.

Figure Five. First CharSearch example.

Forth Dimensions
March 1992 April
\end{verbatim}
Figure Six. Using LEAVING with COMPLETED.

```
- cursor @
  textEnd @ cursor @

  times <
  charGet

  dup ascii ( =
  WHEN
    true
    false

  dup ascii [ =
  WHEN
    true
    false

  dup ascii [ =
  WHEN
    true
    false

  >LEAVING<
  drop

  >COMPLETED<
  drop cursor ! false

@times@ended?
```

tried, until now, to be both a TIMES loop and a LOOP loop. But when you try to do two things at once, you do them inefficiently. The drawbacks of the DO LOOP are:

- Counter-intuitive position of loop start and end (end start instead of start end). This has allowed the DO LOOP to work as a TIMES loop when used like 0 DO ... LOOP. (If the positions had been start end, we would have had to write 0 SWAP DO ... LOOP.)
- Slow execution when working like TIMES because, instead of decrementing and checking a flag like the TIMES construct, the DO LOOP must increment and check against a limit. (This drawback, coupled with the appearance of Forth processors, has led to the use of FOR NEXT in recent times.)
- Slow execution when working like DO +LOOP because the step is pushed and popped from the stack at every iteration of the loop without any valid reason.

Forth traditionalists could sustain that, in such a manner, it is possible to use a computed loop step that varies from one iteration to the next, but we can observe that real life cases have a constant loop step and, moreover, a much more flexible loop is furnished by the FOR { ... FOR} construct, which allows for any test at any step (or any new computed index value), and any model of loop pre-increment or post-decrement, etc.

Establishing the Best Input Characteristics for Loops

Definite loops are usually used for doing something a certain number of times, while allowing the index to assume a predefined set of values. The set of values may be specified by three out of the following four parameters.

- `start` First index value.
- `end` Last index value (or the value after the last one).
- `step` Index step, the constant difference between two successive index values.
- `times` Total number of different values assumed by the index during the loop, equal to the number of times the loop will be executed.

The possible combinations that can be used to specify the set of values the index will assume are:

- `start end step`
- `start end #times`
- `start #times step`
- `end #times step`

(The combination end #times step isn't worthy of discussion.)

Start end step is the combination chosen by Forth and other languages. Forth uses loops primarily to work on memory addresses, so start specifies the first address we have to work on, and end is the limit address. In such cases, it is sometimes useful to specify start size step instead, where size is the size of memory we want to work on (size = end - start).

Start end #times can be used when we want to sample a function in an interval given by start end with a certain resolution: #times. This combination would probably have to be implemented with floating-point numbers—its usefulness with integers is dubious.

Start #times step is used when we work on an array of elements of which we know the starting address, the number of elements, and the element size. This combination is the simplest and most efficient to implement. It will be our base for implementing all other kinds of loops. (We usually know the number of elements in an array and its first memory address.) Moreover, specifying start #times step doesn't allow room for misunderstanding.

Specifying start end step raises some subtle points to consider. Suppose the specified end is step aligned with the given start (as in 10 start 20 end 2 step). Do we mean that the loop must assume the end value, or must it stop at end - step? If we want to have the relation between end, start, and size expressed simply by size = end - start, we must deduce that the end value has to be excluded. Otherwise, the relation between end, start, and size must be written as size = end - start + step.

Suppose, on the contrary, that the specified end isn't step aligned with the given start (as in 10 start 21 end 2 step).
Is that an error condition? If not, how many times do we have to repeat the loop?

Let’s compare the choices of BASIC and Forth, and deduce the relationship between start, end, step, and #times.

Considering a positive step, in BASIC we write:

```
FOR I=10 TO 18 STEP 3 : PRINT I : NEXT I
```

loop is done 3 times

```
FOR I=10 TO 19 STEP 3 : PRINT I : NEXT I
```

loop is done 4 times

```
FOR I=10 TO 20 STEP 3 : PRINT I : NEXT I
```

loop is done 4 times

```
FOR I=10 TO 21 STEP 3 : PRINT I : NEXT I
```

loop is done 4 times

```
FOR I=10 TO 22 STEP 3 : PRINT I : NEXT I
```

loop is done 5 times

So the relation is 

```
#times = (diff - 1)/step + 1
```

where, if we round toward negative infinity, the result of the division is negative and so, correctly, the looping construct issues an error message. If we round toward zero, the loop will execute zero times without issuing the error message.

In standard Forth, a DO +LOOP works like:

```
19 10 DO I. 3 +LOOP loop is done 3 times
20 10 DO I. 3 +LOOP loop is done 4 times
21 10 DO I. 3 +LOOP loop is done 4 times
22 10 DO I. 3 +LOOP loop is done 4 times
23 10 DO I. 3 +LOOP loop is done 5 times
```

So the relation is 

```
#times = diff step + 1 where diff := end - start
```

In standard Forth, a DO +LOOP works like:

```
19 10 DO I. 3 +LOOP loop is done 3 times
20 10 DO I. 3 +LOOP loop is done 4 times
21 10 DO I. 3 +LOOP loop is done 4 times
22 10 DO I. 3 +LOOP loop is done 4 times
23 10 DO I. 3 +LOOP loop is done 5 times
```

So the relation is 

```
#times := diff/step*
```

That choice is equivalent to the Forth one for the case step = 1 (most common). Besides, that choice has some useful consequences when the step is not aligned to the start (see later examples) while maintaining historical continuity when the end is aligned to the start.

Additional Support Words for Definite Loops

Keeping the above discussion in mind, let’s consider the various ways we can specify an array on which we have to work:

```
StartAddress #elements SizeOfElements
StartAddress SizeOfArray SizeOfElements
StartAddress LimitAddress SizeOfElements
```

\ limit address is the first address not belonging to the array

```
StartAddress LastElementAddress SizeOfElements
```

So, the generic pattern is 

```
StartAddress ??? SizeOfElements.
```

Thus, let’s define three modifiers SIZE, END, and END that operate on three numbers and convert them to the standard format (start, #times, step):

(Continued on page 37.)
New FIG Board Members

In lieu of the usual "President's Letter," we offer the following statements made by the newest members of the Forth Interest Group's Board of Directors. They were installed last November at the FORML Conference, where the new board also held its first meeting. The board is now composed of the following individuals:

John Hall, President
Jack Woehr, Vice-President
Mike Elola, Secretary
Dennis Ruffer, Treasurer
David Petty
Nicholas Solntseff
C.H. Ying

The board welcomes comments from FIG members. John Hall will return in the next issue with his "President's Letter."

Mike Elola

"I started out my involvement in FIG as secretary of the business group that meets once a month to discuss the operation of FIG. I came to my first meeting at the request of Kim Harris, who considered me a good candidate to replace him as secretary. Partly out of respect for him, I agreed to become part of the business group.

"I considered my role as that of an observer for the first couple of years. Soon I overcame my initial skepticism with the business team members and their qualifications. By now, I have gained substantial respect for the leadership skills of the outgoing president, Robert Relling, as well as the current president, John Hall. (The president presides over the business meetings, and ends up having to referee some very delicate clashes during the long haul.)

"This experience gives me a background with FIG and its leaders, so I feel confident that I can contribute. My biggest concern for FIG has not really changed: I have always been concerned that our collective FIG energies might not be applied properly to obtain needed goals. Along with my help, I now feel that the business group has made considerable progress in setting priorities and focusing its energies. In the years that I have served, I have become especially aware of our limitations. Understanding and confronting those limitations is a vital leadership skill. Otherwise, we can easily squander our limited resources, both in terms of volunteer time and FIG reserves.

"Although our progress at learning to work within our limitations has come slowly, I am proud about the decisions we have made so far. Over the last year, we have spent considerably less money than some of the same services. More remarkably, we have not sacrificed the quality of those services. (Admittedly, some of the services have been cut. Others besides myself have had to step up our level of volunteer work to compensate.)

"No doubt FIG has persevered because of its determined leaders. I'd like to continue to serve FIG, now more than ever since I expect to be able to enjoy monitoring FIG's financial stabilization, if not recovery. Innovative ideas from all of our business team partners have been essential to help turn things around. Beyond specific measures we have taken, a long soul-searching period has contributed to our success. This has helped instill similar attitudes in most of the business team members, and increases my ability and eagerness to serve.

"Still, we cannot yet rest assured of our future, and I don't know if we ever will (this is not necessarily so bad). I personally feel that any stability we realize by small but steady efforts is more durable than stability or growth that is, perhaps, attainable by occasional concerted efforts involving greater risks.

"I also hope to moderate the efforts of others who would try to vitalize Forth with some kind of slick marketing shit. Steve Wozniak and the Homebrew Computer Club are now much further away from mainstream culture than they were at one time. A group such as FIG may need to remain in relative obscurity for the foreseeable future. Nevertheless, we should position ourselves comfortably. Acting out of desperation is not the way to inspire and keep the faith of our new and returning members."

Nicholas Solntseff

Dr. Nicholas Solntseff is of Russian emigré background and was born in Shanghai, China well before World War II. He was educated in English in Shanghai and completed his schooling at Sydney Technical High School in Sydney, Australia. Dr. Solntseff attended Sydney University, where he studied physics and obtained his B.Sc. in 1953 and Ph.D. in 1958. After a period of employment in England's nuclear engineering industry, he joined the University of London in 1963. Returning to Australia in 1967, Dr. Solntseff switched to Computer Science and taught at the University of New South Wales until 1970, when he moved to McMaster University (Ontario, Canada) after a year as Visiting Professor at the University of Colorado.

Dr. Solntseff has been involved with Forth since 1981, when he implemented his first fig-Forth on an Ohio Scientific microcomputer. He has been the convener of the South Ontario Chapter of the Forth Interest Group since its inception early in 1982. Dr. Solntseff's research interests include the implementation of a Forth-like language called Markov, as well as interfacing Forth with Microsoft Windows. For the last two years, Dr. Solntseff and his students have been working on human-interface techniques in Medical Expert Systems being developed in the Department of Clinical Epidemiology, McMaster University.

Jack Woehr

Jack Woehr learned Forth in 1986, and quit his factory (Continued on page 35.)
Discussion regarding the ANS Forth draft standard continued hot and heavy as we entered 1992. On January 16, the special invited guest in our on-line conference was Greg Bailey of Athena Programming and Technical Subcommittee chair on the X3J14 Technical Committee. Greg’s topic was "The Costs and Benefits of Adopting ANS Forth." If you were not present at Greg’s guest conference and have not yet captured the transcript, I highly recommend doing so.

There will be no further discussion of the ANS Forth effort in this column in this issue because, by the time you read this, the Technical Committee will have met in mid-February to vote. There may not be that much to do as late as mid-January the committee had received zero (that’s correct, zilch) comment. If you had specific comments or objections and failed to submit them, you have only yourself to blame. The opportunity was certainly there. I doubt if any standards effort has ever been so open to scrutiny.

So, are there any other hot buttons? You bet! Lots of them. Object-oriented programming and embedded systems still enjoy lively exchanges, but maybe one of the hottest topics on GENie and ForthNet is minimal Forth kernels that also perform. Witness the topic opened by fellow sysop Elliott Chapin on January 1, 1992. This excerpt, taken on January 17, only runs for 2 1/2 weeks and I already know of at least two outstanding replies.

If you aren’t participating in the guest conferences and in these discussions, you are missing out on a lot of the fun associated with being a Forther. Consider joining us soon.

**Topic 26: Minimal Forth**

How small can a working Forth be? Why try?

The minimal-Forth question has started up again; small wordsets are more than an intellectual exercise. Some processors are very small. Small kernels ease porting.

From: Ralf E. Stranzenbach
Subject: List of Forth words? Hi,

I’m searching for a list of Forth words that is required to be implemented in assembly language to create a reasonable, but very small in size, Forth environment.

Is there anyone who has assembled a list containing those primitives and, possibly, the implementation of the ‘higher-level words’?

—Ralf

I’ve sat in on a couple of informal discussions on this subject. The general consensus is that about ten words are sufficient:

**Stack ops:**

DUP (create a stack element)
DROP (destroy a stack element)
SWAP (move stack element)

**Arithmetic/logic:**

LITERAL (constants, etc.)
NAND (sounds silly, but you can synthesize anything else out of it)

**Address-space access:**

P (! and @ (or C! and C@, if you wish)

P! and P@ (for I/O space port access, if your CPU has such a thing...)

**Dictionary extension:**

CREATE

This is a very sparse list! Even the rawest bootstrap system would probably define a richer set of primitives than this. For example, + and * and / would be a lot nicer if they were primitives, rather than colon definitions made of tens or hundreds of primitive ops. Similarly, words like FIND are very nice to have!

A more sensible minimum set of primitives may be found in the eForth model; no doubt an implementation of it is available for most any CPU you’re interested in. Also look at the current ANS Forth proposal; the Core wordset will give you a good idea as to what functions are needed (although lots of them won’t be primitives in an implementation such as you describe).

Discussions of the "angels-on-the-heads-of-pins" variety continue as to which primitives belong on the short list. For example, if you have R8 you could synthesize DUP.

Rob Chapman once proposed a set of primitives for a Forth machine that had two kinds of arithmetic/logical ops; the first kind returns the value of the result, the second kind returns the resulting carry bits.

Regards, Milan J. Merhar

From: Doug Philips
Subject: Looking for a small PD-Forth for the 8086

Ralf E. Stranzenbach writes, "I've heard about a small Forth named MINI4T41. Does anyone know where to get it?"

I found it in FIG's on-line library on GENie. It is now available via e-mail from FNEAS. To get it, send a message to:

fneas@willett.pgh.pa.us

with the following body:

Send MINI4T41.ARC path your-email-address:RELATIVE-to-the-INTERNET-goes-here
You must supply an Internet relative e-mail address with the path command.

—Doug
Preferred:
dwp@willett.pgh.pa.us
Okay: [pitt,seil]willett@dwp

From: Nick Janow
Subject: Minimal Forth
Elliot C writes:
"The minimal-Forth question has started up again; small words sets are more than an intellectual exercise. Some processors are very small. Small kernels ease porting."

Minimal kernels might also be valuable on large processors. If the kernel and a program can fit in the cache, it will really scream along.

Nick Janow@mindlink.bc.ca

From: Andy Valencia
Subject: Looking for a
Philips writes:
"In other strings I read the word POSTPONE. What is this? How is it implemented?"

: COMPIL
R> DUP CELL+ >R @ , ;
: POSTPONE
BL WORD FIND
DUP 0= IF <not found code> THEN 0< IF COMPILE COMPILE THEN , ; IMMEDIATE

COMPILE is not part of ANS Forth and this definition here is exactly the wrong thing, because it is rather tricky and dependent on a threaded-code Forth. POSTPONE (nice word, awful name) is not what a Forth programmer does with the things, he doesn’t want to do (first postpone them, and then wait...). It postpones the compile time behavior, thus it is COMPIL for non-immediate words, and [COMPILE] for immediate.

"What Forth words need to be coded in assembly to have a good performance. (My estimation 30 to 50 will do.)"

Arithmetics:
+ - AND OR XOR UM*/ UM/ MOD
.Stack:
DUP OVER SWAP ROT DROP
Return-Stack:
>R EXECUTE
Memory:
@ ! C@ C! MOVE FILL
Tests:
0= <K < U<
Dictionary
FIND or an appropriate basic of it
Inner interpreter:
NEXT DOCOL (DOES>
Control flow:
When CLONE puts together the new image, it has to perform many relocations, since everything is moved around relative to the start of the program (as opposed to the start of the JForth image).

This means that CLONE must be able to tell the difference between a compiled number and a compiled address (as ['] might produce).

For this purpose, I've implemented ALITERAL.

The concept that a compiled number must be able to be differentiated from a compiled address is not normally needed in Forth, but for sophisticated functionality such as CLONE, it is necessary.

The answer to the above question is that there is no standard way of compiling in an address vs. a literal number... and there should be.

From: Rob Chapman
Subject: Min. Forth and good performance
I once had similar high-level definitions for:

```
: BRANCH ( -- )
  R> @ >R ;
: 0BRANCH ( n -- )
  0= R @ CELL -
  R - AND R> +
  CELL + >R ;
```

In this case I got faster branches but slower conditional branches.

0BRANCH and 0= represent the classic chicken-and-egg syndrome. In the above definition, we depend on 0= as a primitive. However, if we wish to define 0= in Forth:

```
: 0= ( n -- f )
  IF -1 ELSE 0 THEN ;
```

then we need conditional branching as a primitive.

Other amusements dredged up from the cellar [are given in Figure One.]

—Rob

From: E.RATHER [Elizabeth] Horrors!

From: Mitch Bradley
Subject: Min. Forth and good performance
Mike Haas writes,

"When [JForth's] CLONE puts together the new image, it has to perform many relocations, since everything is moved around relative to the start of the program (as opposed to the start of the JForth image)."

"This means that CLONE must be able to tell the difference between a compiled number and a compiled address (as ['] might produce)."

"For this purpose, I've implemented ALITERAL."

My Sun Forth and Forthmacks systems, which are fully relocatable and support an "application stripper" program that does the same thing as JForth's CLONE, have the same problem and solve it in the same way. My equivalent of ALITERAL is named '.."

"The answer to the above question is that there is no standard way of compiling in an address vs. a literal number... and there should be."

You can do it in ANS Forth by using POSTPONE with ['] or by using something like:

```
S" ['] FOO" EVALUATE
```
possibly with a string that is constructed at run time.

Mitch.Bradley@Eng.Sun.COM

---

**Figure One.**

```forth
: EXECUTE ( tick -- ) >R ; ( not that portable though! )
: ROT ( a b c -- b c a ) >R SWAP R> SWAP ;

( ==== Inner Interpreters ==== )
: (VAR) ( -- addr ) R> ;
: (CONST) ( -- n ) R> @ ;
: (NEXT) ( -- ) R> R> ?DUP IF 1 - >R @ ELSE CELL + ENDIF >R ;
: (LIT) ( -- n ) R> @ + >R ;
: (DO) ( limit \ index -- ) SWAP R> SWAP R> SWAP R> >R ;
: (LOOP) ( -- ) R> 1 + DUP R <
  IF >R DUP @ + ELSE R> 2DROP CELL + ENDIF >R ;
: (+LOOP) ( n -- ) R> SWAP DUP R> + SWAP 0< OVER R < XOR
  IF >R DUP @ + ELSE R> 2DROP CELL + ENDIF >R ;

( ==== Comparisons via divide and conquer ==== )
: < ( n \ m -- flag ) 2DUP XOR 0< IF DROP ELSE - ENDIF 0< ;
: > ( n \ m -- flag ) 2DUP XOR 0< IF NIP ELSE SWAP - ENDIF 0< ;
: U< ( n \ m -- flag ) 2DUP XOR 0< IF NIP ELSE - ENDIF 0< ;
: U> ( n \ m -- flag ) 2DUP XOR 0< IF DROP ELSE SWAP - ENDIF 0< ;

For the really esoteric:

( ==== Unsigned multiplication and division ==== )
: quot< ( n \ q -- q ) 2* OR ;
: rem<cm ( r \ m -- r ) 0< 1 AND SWAP 2* OR ;
: div? ( n \ r -- n \ ?rn- \ f ) OVER - DUP 0<
  IF OVER + 0 ELSE 1 ENDIF ;
: /MOD ( m \ n -- r \ q ) SWAP 0 OVER rem<cm SWAP 2* ( n \ r \ m/q )
  F FOR >R div? SWAP R rem<cm SWAP R> quot< NEXT
  >R div? >R NIP R> R> quot< ;
: / ( n \ m -- quot ) /MOD NIP ;
: MOD ( n \ m -- rem ) /MOD DROP ;
: * ( n \ m -- nm* ) ( unsigned ) 0 SWAP
  F FOR DUP >R 0< IF OVER + ENDIF 2* R> 2* NEXT
  0< IF + ELSE NIP ENDIF ;
```

---

March 1992 April Forth Dimensions 34
I have sought and accepted admission to the Board of Directors of the Forth Interest Group in order to contribute to the preservation of an organization which has proved so useful in the past decade, an organization which hopefully shall continue to render effective service in the coming decade.

"The Forth community is a besieged minority. The Forth Interest Group has suffered a decline in recent years. Old members have left the group, and new members are slow to replace them. At the same time, Forth professionals are confronted with the paradox of Forth usage increasing while Forth market share is declining.

"Yet we believe that our heterodox model of computation presents a more holistic approach to the interaction of man and machine than the path taken by orthodox computer scientists. If our approach still has value, then the practices and institutions which have served us are worth the effort taken to maintain them.

"I have sought and accepted admission to the Board of Directors of the Forth Interest Group in order to contribute to the preservation of an organization which has proved so useful in the past decade, an organization which hopefully shall continue to render effective service in the coming decade.

"The Forth Interest Group has historically fulfilled two important roles, that of assisting Forth programmers. I know this from experience. I learned Forth in the course of many entertaining Saturdays spent at Wolf and Pruneridge Roads. Furthermore, each phase of my successful career in Forth has involved employment found either at a meeting of the Forth Interest Group or via the GEnie Forth Interest Group RoundTable.

"I call upon all enthusiastic exponents of the Forth approach to urge their Forth acquaintances to join or to renew their membership in the Forth Interest Group. FIG will try to keep up its end of the bargain by constantly improving the quality of Forth Dimensions and by increased attention to the needs of beginners, to local chapters, and to community activities for the promotion and benefit of Forth.

"It's time to vote with your feet: if you walk away from FIG, FIG will become merely a part of computer club history like the Homebrew Society. If you stay, and if you bring others, FIG will continue as your tutor, friend, and advocate. Now it's up to you."

—Jack Woehr
jw@well.uucp
JAX on GEnie
Sysof, RCFB 303-278-0364
FAX 303-422-9800

If you stay, and if you bring others, FIG will continue as your tutor, friend, and advocate.

Total control with LMI FORTH™

Call for Papers!

Forth Dimensions is sponsoring a contest for articles about "Forth on a Grand Scale"

Write about large-scale Forth applications, systems, or...

1st prize: $500
2nd prize: $250
3rd prize: $100

See editorial in this issue.

$ New Contest Announcement $
Volume XII
Index

A subject index to Forth Dimensions contents published from May 90 - April 91. Prepared by Mike Eldola.

architectures, threading models for Forth
Letter, vol 12, #1, pg 7
arithmetic operations
Extended-Precision Math Made Easy, vol 12, #4, pg 16
arrays
Arrays in Forth, vol 12, #6, pg 13
assembly language
Forth Assembler & Free Use of Labels, vol 12, #4, pg 23
bulletin boards
Best of GEne, vol 12, #5, pg 36
calculator simulation
Pencil-and-Paper Arithmetic, vol 12, #6, pg 22
chapters, Forth Interest Group
Talking it up in the Outfield, vol 12, #1, pg 32
compiling systems, Forth
Letter, vol 12, #1, pg 7
deferring compilation, compiler directives for
Forth: A 68000 Native-Code Forth (Part 5), vol 12, #6, pg 16
design, language-driven
Letter, vol 12, #3, pg 5
design philosophy, Forth
Dictionary Structures and Forth, vol 12, #2, pg 26
division, 8086
Reliable 8086 & 80286 Division, vol 12, #4, pg 5
division, floored
Best of GEne, vol 12, #5, pg 36
Positive-Divisor Floored Division, vol 12, #1, pg 14
dynamic memory management
Best of GEne, vol 12, #3, pg 37
Dynamic Memory Allocation, vol 12, #3, pg 17
Dynamic Virtual Memory Management, vol 12, #3, pg 7
Banking on the R65F11, vol 12, #1, pg 23
Stack Variables, vol 12, #1, pg 18
education of Forth programmers
Best of GEne, vol 12, #6, pg 24
Best of GEne, vol 12, #4, pg 30
error handling
Letter, vol 12, #5, pg 5
Best of GEne, vol 12, #3, pg 37
file operations, generic
Forth: A 68000 Native-Code Forth (Part 2), vol 12, #3, pg 34
floating point arithmetic
Pencil-and-Paper Arithmetic, vol 12, #6, pg 22
Forth: A 68000 Native-Code Forth (Part 3), vol 12, #4, pg 26
Forth Dimensions magazine
Letter, vol 12, #1, pg 6
Forth Interest Group
President's Letter, vol 12, #6, pg 5
Best of GEne, vol 12, #5, pg 36
Letter, vol 12, #2, pg 5
Forth leaders
Best of GEne, vol 12, #6, pg 24
history of Forth
Editorial, vol 12, #4, pg 4
information services
International GEne Access, vol 12, #6, pg 26
GEne for Beginners, vol 12, #5, pg 34
interfacing Forth to operating systems
Forth: A 68000 Native-Code Forth (Part 2), vol 12, #3, pg 34
interpreters, special purpose
Forth & the Three-Number Problem, vol 12, #4, pg 20
interrupts
Interrupt-Driven Communications, vol 12, #5, pg 18
Forth Interrupt Handling, vol 12, #5, pg 10
local variables
Forth: A 68000 Native-Code Forth (Part 3), vol 12, #4, pg 26
Stack Variables, vol 12, #1, pg 18
macros, text interpreter
Interpretation-Macro Techniques, vol 12, #1, pg 11
meetings, annual
Forth Day 1990, vol 12, #6, pg 30
meta-compiling
Meta-compilation Made Easy, vol 12, #6, pg 31
Meta-compile by Defining Twice, vol 12, #2, pg 31
microprocessors, Forth
Best of GEne, vol 12, #5, pg 36
music, ear training
Ear Training, vol 12, #5, pg 24
number formatting operations
Upscale Number Input, vol 12, #2, pg 8
number input
Upscale Number Input, vol 12, #2, pg 8
object-oriented programming
Best of GEne, vol 12, #1, pg 28
optimizing compilers
Forth: A 68000 Native-Code Forth (Part 4), vol 12, #5, pg 28
Forth: A 68000 Native-Code Forth, vol 12, #2, pg 20
Extensible Optimizing Compiler, vol 12, #2, pg 14
printing of source code, pretty-printers
Letter, vol 12, #3, pg 6
promoting Forth
Best of GEne, vol 12, #6, pg 24
Embedded Control: Path to Forth Acceptance, vol 12, #1, pg 35
promoting the use of Forth worldwide
Soviet Forth-Info, vol 12, #6, pg 11
reviews and surveys
Best of GEne, vol 12, #2, pg 36
serial line communication routines
Interrupt-Driven Communications, vol 12, #5, pg 18
sieve-of-primes benchmark
Letter, vol 12, #2, pg 6
Re: Letter, vol 11, #5, pg 6
Letter, vol 12, #2, pg 5
simulations
From Clepsydras to Neural Nets, vol 12, #6, pg 6
stack effects testing
Testing Toolkit, vol 12, #3, pg 31
standards, ANSI
Best of GEne, vol 12, #5, pg 6
Letters, vol 12, #3, pg 37
Best of GEne, vol 12, #3, pg 37
target compiling using a hosted target
Interrupt-Driven Communications, vol 12, #5, pg 18
Smart RAM, vol 12, #3, pg 31
tutorials
Meta-compilation Made Easy, vol 12, #6, pg 31
GEne for Beginners, vol 12, #5, pg 34
Forth & the Three-Number Problem, vol 12, #4, pg 20
variables, headless
Anonymous Things, vol 12, #1, pg 8
variables, stack
Stack Variables, vol 12, #1, pg 18
vectored execution
Forth: A 68000 Native-Code Forth (Part 5), vol 12, #6, pg 16
Moreover, sometimes an array must be scanned in reverse order, although it's easier to specify the array by its starting address. Thus, let's define the additional modifier BACK to be used like

```
BACK LOOP { ... LOOP }
```

This has the effect of reversing the order of the values assumed by the index in the absence of BACK:

```
: BACK ( StartAddr #elements ElementSize -- LastAddr #elements ElementSize )
  DUP NEGATE >R
  OVER 1- *
  ROT + SWAP
  R> ;
```

With the above choices and definitions, we have a very flexible loop construct that accepts various input formats and greatly simplifies work with arrays.

Obviously, depending on the problem at hand, similar techniques may be used to extend the patterns accepted by the loop construct, making it possible to feed the construct with a format natural to the problem at hand. Moreover, the basic format is very easy to implement, fast to execute, and precise in meaning.

**Examples**

**Create and initialize a table:**

```
: LOGOSINIT
  LOGOS #LOGOS CELL
  LOOP{
    SIMPLELOGO I !
    LOOP} ;
```

**Search our table for a specified value and leave its address if it is found:**

```
: logosearch ( logo -- false | addr true )
  LOGOS #LOGOS CELL
  LOOP{
    dup I @ =
    WHEN{
      drop I true
      WHEN}
  COMPLETED{ drop false
    LOOP} ;
```

**Search in reverse order:**

```
: logoBackSearch ( logo -- false | addr true)
  LOGOS #LOGOS CELL BACK
  LOOP{
    dup I @ =
    WHEN{
      drop I true
      WHEN}
  COMPLETED{ drop false
    LOOP} ;
```

(Code, figures, and article continue in next issue.)
A FORML Thanksgiving

Richard Molen
Huntington Beach, CA

On November 25, 1991, just over forty dedicated Forthers flocked to Asilomar on California's Monterey peninsula to participate in the FORML conference. Some went to exchange ideas, some went to exchange addresses, all went to exchange experiences and to increase the collective knowledge base of the Forth community. This year there was something for just about everyone.

As we arrived at Asilomar, the air was cool and clear with a strong breeze. As we rushed in (late) to register and pick up our name badges and notebooks, we were intercepted with a big hug by Wil Baden who makes it a point to greet everyone at FORML in this fashion. It’s a nice way to start.

Asilomar is a beautiful retreat sprinkled with wind-swept Monterey pine and cypress trees. Weathered boardwalks cut through the struggling vegetation in the dune restoration project, leading to a beach of white sand and deep blue water—a living picture. Early morning finds deer foraging among the trees and a seascape full of life. In these surroundings, it was easy to relax and concentrate on the conference itself. As soon as we were registered, the lunch bell rang and we went to the cafeteria to eat. I was impressed by the simple elegance of the cafeteria. It wasn’t until after we were seated that I noticed there weren’t any menus. How nice it was to not worry about what to eat. “Could there be a GUI lesson here?” I mused. I enjoyed my French dip sandwich, as Guy Grote gave me bits of information to mentally munch on. He to be served.

The first presentation came from Mike Elora. His eyes lit up as he described HomeComing Forth, his implementation of a minimal Forth system on top of Apple’s HyperCard environment. HomeComing Forth is simple, with very nice debugging tools. For example, a definition’s object and source code are displayed side-by-side when editing, encouraging users to see what is actually compiled. Mike’s paper was quite tasty.

If you have ever crashed your system by leaving an unbalanced return stack, raise your hand, lower your head, or at least read on. Roland Kolinak’s paper describes and implements return stack security using a temporary stack for compile-time housekeeping. While this takes a little more compile time, it is well worth it for applications which allow users open access to the Forth environment. In addition, he added a prompt which displays the top three cells on the temporary stack when compiling a definition interactively. This allows a programmer to see what’s happening as the definition is compiling. Roland received the "most in keeping with FORML" award (a bottle of wine) for this effort.

Guy Kelly served up a very informative entree by speaking of his efforts to characterize tradeoffs in various Forth architectures. In his paper, he benchmarked and characterized 19 of the most common Forths. See article in this issue.—Ed. He also documented the sequences required to open a file and load a program, the various assembler syntaxes, and a brief on each of these Forths. In addition, he further isolated the effects of threading, segmenting, and register usage by manipulating each of these components using riFORTH as a base system. Guy found only a 2:1 performance ratio between the fastest and slowest of these versions. He concluded that other considerations often outweigh this performance gain. This certainly surprised me. His tests and riFORTH are available on GENie.

One of my favorite foods for thought is metacompilation. Guy dispels the mysteries of metacompilation and offers the metacompiler which he used for the benchmarking project. This metacompiler, also available on GENie, is capable of generating new Forth systems with various threading and memory-segmenting schemes. Guy’s papers are a must for anyone who wants to experiment with Forth architectures. It is easy to see why Guy Kelly won the “Public Service” award. Thank you, Guy.

Andrew McKewan spiced up the Motorola 6805 emulator with an optimizing Forth native-code compiler. This must be ambrosia for anyone working with the 6805 emulator. He commented that after he tossed out the
PCYerk Classes

Rick Grehan

Peterborough, New Hampshire

Rick Grehan is a senior editor at BYTE magazine and the technical director of BME Lab. He first encountered Forth over seven years ago when developing a music synthesizer control system built around a KIM-1. Since then, he has used Forth on 68000 systems (including the Macintosh), the Apple II, and the IBM PC. He has also done extensive work on the SC32 stack-based processor. Rick has a B.S. degree in physics and applied mathematics, and an M.S. degree in mathematics/computer science. His work on a PC version of the Yerk implementation won first prize in FD's object-oriented Forth contest.

The following code builds on the object-oriented Forth discussed in the last issue of FD.

```forth
\ ***************
\ ** BASIC CLASSES **
\ ***************

\ *******
\ ** object 
\ *******
:class object
  0 ivar dummy \ Used to get offset to ivar area

\ Return address to object's instantiation in variable
\ segment. You can use this to get an object's address
\ and store it in a variable for deferred binding.
\ E.G.:
\    variable frank
\    12 word_array bob
\    44 fill: bob
\    addr: bob frank !
\    2 get: { frank @ } .  FORTH RESPONDS>> 44 ok
\ :m addr: ( -- addr )
\     dummy 2- \ Return address to start of object's ivars region
\ :m ivar-addr: ( -- addr )
\     dummy
\ :m
\ :m length: ( -- n )
\     dummy 2- \ Get pointer to instantiation
\     @ \ Class address in token segment
\     @t \ Length
\ ;m
\ ;class
```

Frank Sergeant has written a three-instruction Forth for embedded system development on a budget. I don't have room here to elaborate in detail—well, only three words, I guess I do. The only words needed to start developing on an embedded system are X8, X1, and XCALL which fetch, store, and execute a routine on the target system, respectively. Frank described how he implemented these words in an MC68HC11 chip. Having used a Cadillac, four-word variation, I'd have to say that Frank is right on target.

Anyone who has done serious development can appreciate the usefulness of version control and file comparison. Wil Baden presented his tools, which he has ported many times, over many systems, over many years. Wil distributed 20 pages of code forming the basis of a text-file-based, source code control system. His implementation is capable of comparing and collating large files, and keeps all versions of a file in a compact format. It is a useful tool in any language. I found it interesting that some of those who used blocks did not see a need for such tools. Perhaps the modularity of blocks, combined with the fact that the majority of those using blocks used date stamping, reduces the need for such tools. Those interested in receiving a copy of this code on disk should contact Wil.

Guy Kelly called our attention to some of the tradeoffs of interpreting
source from text files. By adding some intelligence to parsing words, Guy simplified the definitions of words which use them (i.e., (, . (, .", \, and LOAD). By using block buffers, he eliminated the need for extra text-file line buffers, further simplifying the system. His system, also on GEnie, is simpler and more capable than a BASIS 17 system would be.

On the educational front, Dr. Tim Hendtlass of the Physics Department at the Swinburne Institute of Technology in Hawthorn, Australia, gave an excellent testimonial to the power of Forth in education. Tim described in detail the challenges of teaching interfacing (hardware) to classes of 60 students with various unrelated backgrounds and the dramatic change in their ability to learn interfacing when Forth was used. His paper also contains the exercises used to teach these students to solve simple instrumentation problems using both interrupts and multitasking. People learning to write interrupt service routines can really benefit from Tim's paper. Software for this paper is available on GEnie.

AI buffs and elderly people should take note of Dr. Hendtlass's paper on the development of a distributed, intelligent system which he calls Embedded Node Collectives. Each node collects information, uses an expert system and, sometimes, a neural network to digest this information, and communicates with the outside world in some fashion. These nodes have been used in several systems. The system he cites is one which helps elderly people care for themselves. The neural network—an input into the expert system—

```forth
\ ** STORAGE CLASSES **
\ *******************************

\ *******************************
\ ** ldarray -- 1 dimensional array
\ *******************************
:class ldarray <super object
  2 ivar nelems \ # of elements in the array
  2 ivar elemsize \ Size of each element in bytes

\ Allocate space for the array.
:m allocate: ( n -- )
  dup nelems ! \ Store # of elements
  elemsize @ * allot \ Set aside space in vars segment
;m

\ Set the elements size
:m setsize: ( n -- )
  elemsize !
;m

\ Initialize the array.
\ n is # of elements in the array
\ m is the element size
:m init: ( n m -- )
  setsize: self \ Set the element size
  allocate: self \ Allocate memory
;m

\ Return the # of elements
:m #elems: ( -- n )
  nelems @
;m

\ Return length of data area
:m length: ( -- n )
  length: self \ Header information
  #elems: self
  elemsize @ * \ Length of data portion
  + \ Add it all
;m

\ Do bounds checking for index
:m idx-check: ( i -- i )
  dup 1+ nelems @ > \ Check bounds
  if clear-o&mstacks \ Clear the stacks
    abort" Array bounds exceeded"
  endif
;m

\ Return the address of the array members start
:m array_addr: ( -- addr )
  elemsize 2+
;m
;class
```
learns a person’s habits. The expert system evaluates its inputs to determine what action is needed: a gentle prompting, a phone call for help, etc. I don’t think George Orwell would have cared much for this system, but it certainly can be instrumental in helping an elderly person to be more self-sufficient.

Thank you, Tim, not just for your presentation and software, but also for your recent neural network articles in Forth Dimensions.

Plenty of treats were to be had for the hardwired Forthers. BRAD RODRIGUEZ discussed his PISC-I (Pathetic Instruction Set Computer), which uses 1976 TTL technology, has a mere 2100 gates, and implements Forth in microcode. PISC-I adds a whole new dimension to the phrase “lean and mean.”

Dr. Ting showed us how we can create our own chips at the kitchen table by using the National Security Agency’s public-domain CMOSN macro cell library. So where’s the DIP? Well, Dr. Ting showed us that, too, by using the library to create a 40-pin Data Comparator Chip. With plenty of hand waving, which he promised us in his paper, JOHN Rmil discussed his QS2 (Quicksand 2) project proposal for a graduate-level VLSI design project at the University of California at Santa Cruz. It is a 16-bit microprocessor with classical RISC features, which has, among other things, a hardware-based threaded-code interpreter.

CHUCK MOORE demonstrated his MuP20 chip emulator software which displayed each layer of the chip in a different color. Using the seven-button interface, he scrolled through the chip,
displaying layers both individually and combined. Chuck pointed out that, since the chip can be emulated, the circuitry is tested and the making of the chip is anticlimactic. What caught my eye was the simplicity of the user interface. It seemed so simple I wondered if my four-year-old daughter could learn to use it. The emulator did what he needed—no more, no less. Trivial decisions which could distract him (or any user) from his thinking were all but eliminated. Thank you, Chuck.

Dr. Ting gave the recipe for primordial soup by specifying the modules needed to implement the Tiera Computer Organism System. He also supplied the code and theory for creating and killing simulated organisms, with a challenge to add the mutation-and-evolution components.

Jeff Fox won the “Programming Virtual Hardware” award with his simulation (in F-PC) of the MuP20 running eFORTH. As if running a simulation of Chuck Moore’s latest chip wouldn’t be enough fun, Jeff also simulated (with eFORTH and DesqView) parallel processing with the F20, an enhanced MuP20, using FORTH-Linda—a bulletin board style parallel-processor manager. This must mean that Jeff is simulating virtual machines based on the simulation of a virtual processor, which runs on a virtual machine (i.e., eForth and DesqView). Jeff’s paper describes his efforts in detail.

Another tasty dish was Dr. Ting’s talk about the Catalyst, his contribution to the Human Genome Project—the greatest reverse engineering project of all time. The Catalyst is an auto-

---

```forth
;m
\ Set value at index location
:m put: ( val i --)
  idx-check: self \ Check bounds
  2* \ Index -> offset
  array_addr: self \ Start of array
  + \ Add index
  ! \ Store
;m

\ Fill the array with value
:m fill: ( val --)
  array_addr: self \ Get address
  #elems: self \ # of elements
  0 do
    2dup i 2* + \ Form address
    ! \ Store value
  loop
  2drop \ Clear stack
;m

init: <<init-method
;class

*********
** String
*********
\ A string object consists of a maximum byte count, byte
\ count, and trailing null byte. The maximum count does
\ NOT include the preceding byte count and trailing null
\ byte.
: class string <super object
  1 ivar maxcount
  0 ivar thestring

\ Allocate space for the string
\ n is # of bytes to allocate
:m allocate ( n --)
  dup maxcount c! \ Save in max. count
  2+ \ For byte count & null byte
  allot
;m

\ Clear the string
:m clear:
  0 thestring !
;m

\ Store a string in the string object
\ addr must point to a packed, null-terminated string
:m put: ( addr --)
  \ See if the string will fit
  dup c@ maxcount
  >
  if clear-o&mstacks
    abort" String too long"
  endif
  thestring $!
;m
```
\ Copy contents of string object to destination address
\ Note usage of Upper Deck Forth's $! operator
:m get: ( addr -- )
  thestring swap $!
;m

\ Return the address of the first character of the
\ string and the byte count
:m count: ( -- addr n )
  thestring count
;m
allocate: <<init-method

;class

***************
** String array
***************
\ This array is a collection of pointers to string
\ elements.
:class string_array <super word_array

\ Allocate space for the array
\ n = # of elements
\ m = max. size for each string element.
:m allocate: ( m n -- )
  dup init: self \ Make space for it
  ['] string >body @ \ Need this to make string
  swap
  0 do
    here >r \ Save location
    2dup instantiate \ Make a string object
    r> i put: self \ Store addr of object
    loop
  2drop \ Clear stack
;m

\ Fetch a string at index i. Store in address addr
\ Lots of "get:" messages here. The one in the curly
\ brackets goes to an integer array, and fetches the address
\ of a string object. The get outside the curly brackets
\ sends a get to a string type.
:m get: ( i addr -- )
  swap
  get: { get: self } \ Fetch
;m

\ Place a string at element in index i.
:m put: ( addr i -- )
  put: { get: self }
;m
allocate: <<init-method
;class

Forth Dimensions 43 March 1992 April

mitated molecular biology workstation designed to automate the HGP bottleneck of preparing DNA fragments for analysis. At its core is a three-axes robot arm capable of delivering liquids to .001" accuracy. The software runs on a Macintosh and was written in polyFORTH using a simple round-robin tasker. DENNIS RUPEP spoke last. He pointed out the need for a common validation suite for testing whether a Forth system is compliant. His paper discusses the labeling and documentation aspects of this effort. Dennis is looking for people willing to work with him to develop the suite. In my opinion, a common test suite would force interpretation of the standard in areas where it is unclear and, in a sense, test the validity of the standard itself. Dennis can be contacted at Forth, Inc.

I could talk about the wine and cheese parties (which were fun), the impromptu talks (there were some gems), and the workshops (which were lively) that are the less formal parts of FORML, but I'm running out of room. The presentations were wonderful and I look forward to reviewing many of them in detail, but what I found to be at least as inspiring were the people themselves—their experience, their personalities, and their insights. The presentations will be published shortly in the conference proceedings, but this dimension of FORML will only be captured in the minds of those that attended.
NEW FROM THE FORTH INTEREST GROUP

More on Forth Engines

C.H. Ting
Editor

Seeing Forth

Forth in the context of the intellectual threads of our time.

Jack J. Woehr

More on Forth Engines

Vol. 14
RTX Pocket-Scope, eForth for muP20, ShBoom, eForth for CP/M & Z80, XMODEM for eForth.

Vol. 15
Moore: New CAD System for Chip Design, A portrait of the P20; Rible: QS1 Forth Processor, QS2, RISCing it all; P20 eForth Software Simulator/Debugger.

$15.00 each

Seeing Forth

by Jack Woehr

"... I would like to share a few observations on Forth and computer science. That is the purpose of this monograph. It is offered in the hope that it will broaden slightly the stream of Forth literature, which creek has been running a mite shallow of late. Failing that, perhaps it will serve the function of a cup of warm tea, to make the seeker of Forth literature feel warmer and a little more filled until something more nourishing comes along."

$25.00

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