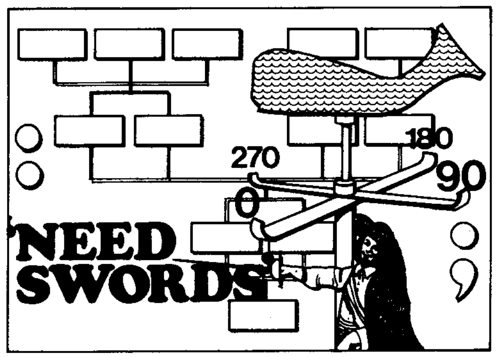
# FORTH Dimensions

Volume 5, Number 3
September/October 1983
$2.50

## 8088 RAMdisk




FEATURES



DEPARTMENTS

**FORTH Dimensions**

Published by FORTH Interest Group

Volume V, No. 3

September/October 1983

Editor
Marlin Ouverson

Publisher
Roy C. Martens

Typesetting/Production
LARC Computing, Inc.

Cover Art
Al McCahon

FORTH Dimensions solicits editorial material, comments and letters. No responsibility is assumed for accuracy of material submitted. Unless noted otherwise, material published by the FORTH Interest Group is in the public domain. Such material may be reproduced with credit given to the author and the FORTH Interest Group.

Subscription to FORTH Dimensions is free with membership in the FORTH Interest Group at $15.00 per year ($27.00 foreign air). For membership, change of address and/or to submit material, the address is: FORTH Interest Group, P.O. Box 1105, San Carlos, CA 94070.

# Letters to the Editor

**Unfinished Business**

Dear Editor,

I am a novice to Forth, but a programmer and analyst for small companies, especially in engineering and scientific work. I would like to see more and better information and programs for new Forth programmers.

Since Forth textbooks are scarce, I learn mostly from *Forth Dimensions*. Many programs are not easy because often they are not fully compatible with my software, or contain words that are not defined. In one program (from "A Techniques Tutorial on Defining Words," Vol. IV, No. 1), there is only one word, **WITHIN**, that I am not able to define at all. I have checked three different textbooks, but the program is still left unfinished.

I would like *Forth Dimensions* to evaluate Forth software like other magazines (*e.g., InfoWorld*). I had to buy three different Forths before finding one I feel comfortable with. I need more information on each piece of software, like which follows the Forth Standard, disk format supported, and options like eighty-column screen, lower-case, editor, etc. These would save my time and money.

Thank you for your time and patience with this comment about publication and evaluation.

Yours sincerely,

William A. Paine
11025 - 131st Ave. NE
Kirkland, WA 98033

*(Continued)*

Editorial

# Standard Fare

Once a month, a pioneering group of Forth aficionados meets to coordinate the considerable business of running a world-wide organization. That it is a not-for-profit affair does not make their duties less complex than those associated with any international business. That the board members are unpaid does not make them less committed, diligent, and effective as managing leaders.

The work of Forth Interest Group members has been largely responsible for the growing public acceptance of Forth. They have called attention to Forth as a practical language and, for more and more projects, as the language of preference. Whenever elements of the language have posed obstacles, they have contributed hours of labor to modify, argue, test, debate, and re-modify to create an improved Forth standard.

Forth-83 has been accepted as the official standard. Two articles in this issue provide a summary of some of the changes that have been introduced, and of some objections that have been raised. Our purpose in publishing these items is to show some of the changes that have been introduced and to let readers see at least part of the process (as well as the importance) of arriving at a new standard.

Of course, the people responsible for all this are just FIG members who get involved. There is always room for another contributor to this loose-knit band. Particularly welcome are articles, ideas, and letters to the editor from the many new members receiving *Forth Dimensions* this year. Let us know how we can help you, and let others know how Forth can help them!

Meanwhile, make good use of this issue and the ones to come. Articles and code are still being accepted for our issues on data acquisition, instrument control, and math. Utilities and useful applications are always welcome. Writers guidelines are available to authors (and potential authors) who send a self-addressed, stamped envelope to:

Editor
*Forth Dimensions*
Forth Interest Group
P.O. Box 1105
San Carlos, CA 94070

We look forward to hearing from each of you!

*—Marlin Ouverson*
*Editor*

### RPN Blues — Revisited

Dear FIG:

I have been trying to implement Forth on my system for two years now, but failed because of not having a good assembler for my system. In those two years I mostly did not work on the Forth system because of frustration. But now I had the opportunity to work with Forth on a friend's machine. Super!

There is one thing I think could provide an improvement in readability of Forth programs: do the control structures have to be in reversed polish notation, or wouldn't it fit in the Forth concept otherwise? How about control structures as below:

**FIG-Forth 79**

**DO** .... **(+)LOOP**

(cond) **IF** (true) **THEN**
(cond) **IF** (true) **ELSE** (false) **THEN**

**BEGIN** .... (cond) **UNTIL**

**BEGIN** (cond) **WHILE** (true) **REPEAT**

**Other version**

kept as it is

**IF** (cond) **THEN** (true) **ENDIF**
[**IF** is only documentary, **THEN** checks condition
**ELSE** is as before]

**BEGIN** .... **UNTIL** (cond) **FULFILLED**
[**BEGIN** is where to jump; **UNTIL** is documentary; **FULFILLED**
is formerly **UNTIL**]

**WHILE** (cond) **REPEAT** (true) **ENDWHILE**
[**WHILE** marks where to jump; **REPEAT** checks if cond is true;
**ENDWHILE** jumps to **WHILE**

What do you think about it?


Horst G. Kroker
HCH-V-Meissen Str. 37
Mainz LL2 6500
W. Germany


### Model Behavior

Dear FIG:

While working with a FIG-Forth system, I found a couple of things which may be of interest for inclusion in other compilers. First, there is a bug in the model's implementation of the logic associated with **?PAIRS** which allows the construct

**IF** ... **ELSE** ... **ELSE** ...
**ELSE** ... **THEN**

to be compiled without error. The execution of the resulting code is entertaining, but not particularly useful. I would suggest fixing it via the following changes to the model:

```
Screen 40:

: ?PAIRS  AND 0= 13 ?ERROR ;

Screen 73:

: ENDIF  ?COMP 6 ?PAIRS HERE OVER - SWAP ! ; IMMEDIATE
: DO     COMPILE (DO) HERE 8 ; IMMEDIATE
: LOOP   8 ?PAIRS COMPILE (LOOP) BACK ; IMMEDIATE
: +LOOP  8 ?PAIRS COMPILE (+LOOP) BACK ; IMMEDIATE

Screen 74:

: ELSE   2 ?PAIRS COMPILE BRANCH HERE 0,
         SWAP 2 [COMPILE] ENDIF 4 ; IMMEDIATE
```

Use of a bit-masked test thus allows **THEN** to follow either **IF** or **ELSE** but only allows **ELSE** to follow **IF**, which is what we want.

The other thing is a compiler speed-up enhancement. I had always wondered why the dictionary search scanned each entry character by character, even though the length was known, but just chalked it up to one of the mysteries of Forth that I'd figure out some day. It should be noted, incidentally, that I always use a **WIDTH** of thirty-one.

Just recently, however, I found out how things work with a width of less than thirty-one (I think) so I see the basic reason. However, I believe that things could be speeded up dramatically by just using the lower of **WIDTH** and the length found in the dictionary header as an increment to skip to the end of the name. I use a 6801-based FIG system which was so modified and the compile times went down by over thirty percent for one system (around 300 screens, would you believe).

Best regards,


Mike Armstrong
7502 S.W. 143rd Ave.
Miami, FL 33183


*(Continued on page 29)*

# FIG-Forth Vocabulary Structure

*Evan Rosen*
*Bayside, New York*

Vocabulary structure and linking in FIG-Forth is a clever and complex affair. FIG-Forth makes extensive use of the linked list in vocabulary management and creates a structure that allows the dumb primitive **(FIND)** to look in the right places without **(FIND)** ever realizing it. This note attempts to make both the creation and search processes a little clearer.

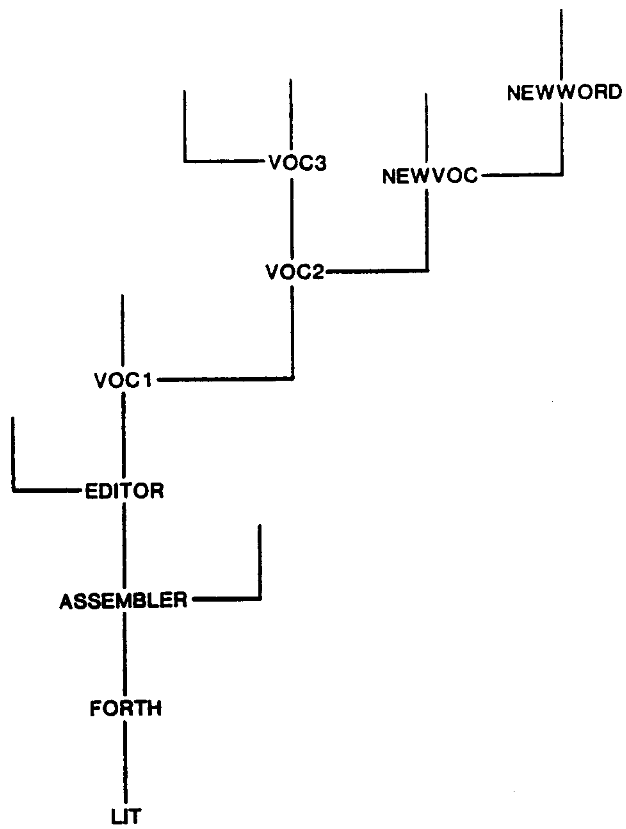
Before explaining how vocabulary structures work, let's talk about what they do.

You may recall that during typical dictionary searches, first the context and then the current vocabularies are searched. These two searches are performed in the same way. Take the context search as an example, first looking at the broad picture and then the details.

**Vocabulary Search (big picture)**

Assume we have the vocabulary "tree" shown in Figure One, and that the vocabulary **NEWVOC** has had a few words added to it. Assume that **NEWVOC** is the context vocabulary. When context is searched, first **NEWVOC** is searched, then **VOC2**, then **VOC1**, and then **FORTH**, assuming no match has been found. **VOC3** is not searched. Thus, the context vocabulary is actually composed of a sequence of vocabularies. The word "vocabulary" itself is, therefore, somewhat ambiguous in FIG-Forth usage.

After a few setup details, the actual search is done by the not-very-smart primitive **(FIND)**, which returns only on a match, or on finding a zero for the next name field address in the search. (The zero shows up in the name field of the first word in **FORTH**, usually **LIT**. Try **' LIT LFA ?**). Hence **(FIND)** has somehow to be guided in order to search all the right vocabularies. This is where Dummy Name Fields, containing the two bytes 81 and A0, come in. To understand the details we have to look at the structure of a vocabulary word.




**Schematic of Example Vocabulary Tree**

Note that lines emerging from the sides of vocabulary words *do not* represent real pointers. See Figure Two for actual configuration.

**Figure One**

| | | | | | |
|---|---|---|---|---|---|
| 86 NEWVOC | LINK | BDCFA(s) | 81 A0 | GRAFT | VOC-LINK |

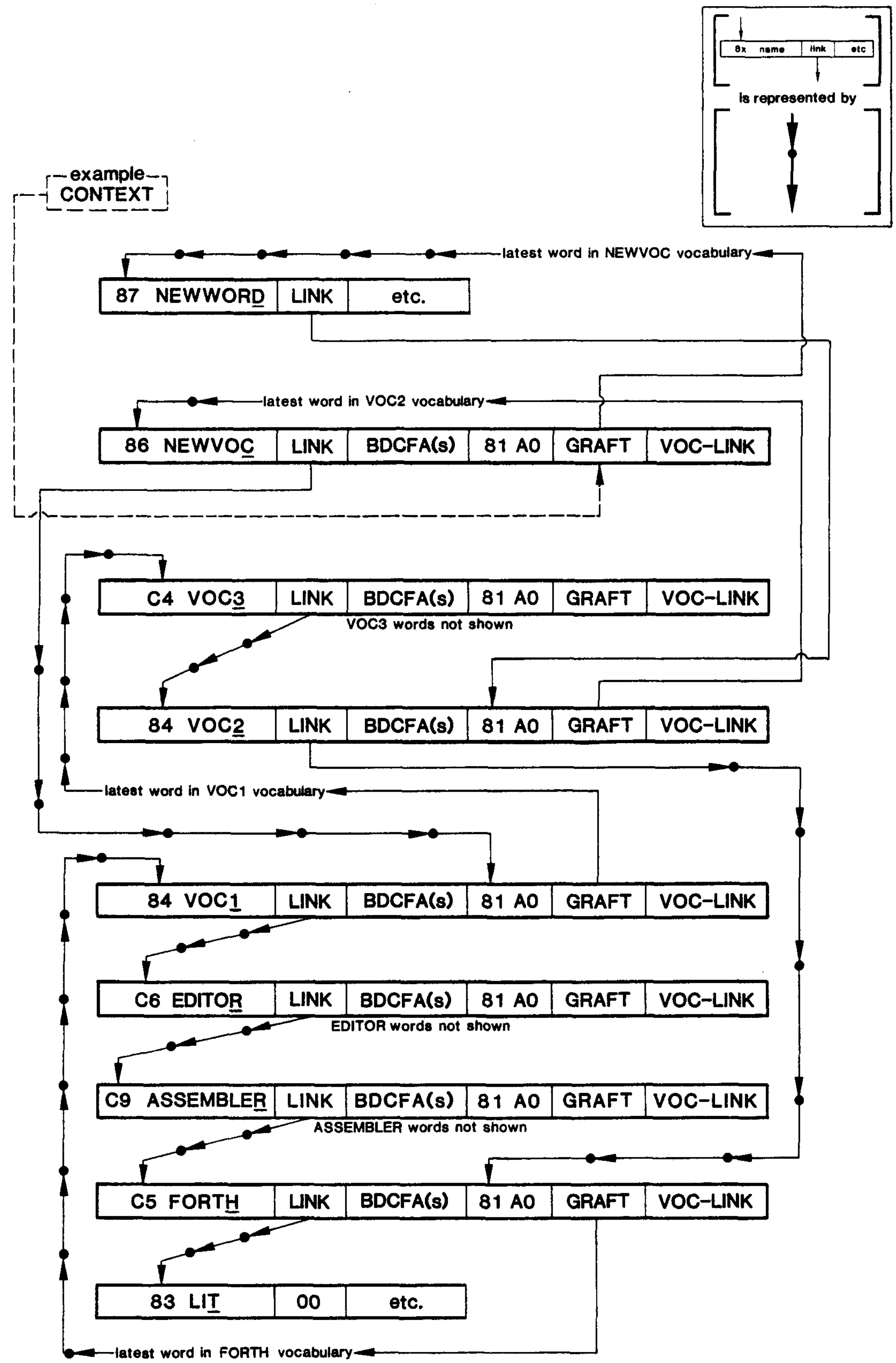
**Figure Two**

**Vocabulary Searches (detailed picture)**

Performing **VOCABULARY NEWVOC** will create the elements shown in Figure Two in the dictionary. Taking these items in order, we have:

**86**

The length byte of the new word, with high bit set, for detection by **TRAVERSE**. If **NEWVOC** were immediate, bit 6 would be set also, making this byte C6.




Detailed Structure of Example Vocabulary Tree

**Figure Three**

## NEWVOC

ASCII of the new word's name. High bit of last character will be set, with very occasional machine-dependent variations, *e.g.*, on 6502 systems.

## LINK

Link back to name field address of the previous word defined in the current vocabulary, *i.e.*, the vocabulary in which the word **NEWVOC** is defined. All normal so far.

## BDCFA(s)

The <**BUILDS...DOES**> code field address(es). This field is generally four bytes long, though the shorter, faster two-byte renderings are gaining prominence. This need not concern us now. You can tell how long this field is by looking for 81 A0 which will follow it.

## 81 A0

This is how A081H, will show up in the dictionary. It is a Dummy Name Field with name of length one (the 1 in 81) and actual ASCII name 20 with high bit set, to become A0. (ASCII 20H is a blank, which was chosen because it was rather unlikely to occur as an actual name in a working system. Back to this in a moment.)

## GRAFT

It's not clear if this field has another name, but calling it the Graft Field is useful for the moment, as this field helps in "grafting" the new vocabulary onto the vocabulary tree. The Graft Field in a vocabulary-name word like **NEWVOC** or **FORTH** is the actual field that is pointed to when we say something like, "**CONTEXT** points to **NEWVOC**."

Right after **NEWVOC** is defined, the Graft Field of **NEWVOC** points to the Dummy Header Field of the vocabulary in which it was defined. In the tree in Figure One, for example, **NEWVOC** was defined in the vocabulary **VOC2**. This is caused by the action of the <**BUILDS** part of the word **VOCABULARY**, when it performs

**CURRENT @ CFA ,**

(This is how the standard is written. **CFA** is misleadingly used as slang for **2** - and should be replaced.)

When the first word, call it **NEWWORD**, in the vocabulary **NEWVOC** is defined, its LF (Link Field) takes the value in the Graft Field, and the Graft Field takes a new value, namely, the NFA (Name Field Address) of **NEWWORD**. The trick is that this is accomplished in the usual way by **CREATE**, which looks at where **CURRENT** is pointing when **CREATE** is ready to set up the new links for **NEWWORD. CREATE** then gets the value in the Graft Field (which in our example points to the Dummy Name Field in **VOC2**), and puts this into the link field of **NEWWORD**. The graft has then been created. More in a moment.

## VOC-LINK

This points to the voc-link field of the previously defined vocabulary. For the bottom vocabulary, generally **FORTH**, this **(VOC-LINK)** will be 0 to indicate the end of the list. We're not going to talk about voc-links here.

Okay, now, let's see what happens when **(FIND)** unsuccessfully searches the context vocabulary, **NEWVOC**. Assume that some setup routine has properly arranged both the stack and the string that **(FIND)** will be trying to find. The address where **(FIND)** will start looking will be at the top of the stack. In this case it will be **CONTEXT @ @**, since **CONTEXT** points to the Graft Field of **NEWVOC** which points to the Name Field of the last word defined in **NEWVOC**. Then **(FIND)** starts looking.

When **(FIND)** reaches the first word defined in **NEWVOC**, which you recall was **NEWWORD, (FIND)** again fails to find a match and so looks in the Link Field of **NEWWORD** to find out where to search next. What is there, if you recall, is the address of the Dummy Name Field of **VOC2**. The unsuspecting **(FIND)** then looks at this field, where it sees the "name" 81 A0, again fails to match, and so goes to what it thinks is the link field corresponding to this

Dummy Name Field. What it finds instead is the Graft Field of **VOC2**. **(FIND)** then looks in this field, gets the pointer to the Name Field of the last-defined work in **VOC2**, and continues its search. This same tricking of **(FIND)** occurs at each intersection in the tree until **(FIND)** finally ends up at the base of the tree, in the **FORTH** vocabulary, at the Link Field of **LIT**, where it finds a zero and exits.

The term "vocabulary" has been carried over from pre-FIG Forths, where it had a meaning closer to what one would expect. A more descriptive name for the FIG-Forth version might well be **VOCABULARY-BRANCH**.

To review, in the current setup in FIG-Forth,

(1) Dictionary searches may repeatedly search various vocabularies within one search. For instance, the **FORTH** vocabulary is generally searched twice.

(2) Dictionary searches search all of each vocabulary-branch through which they pass, not just the part "below" the intersection. "Chronology" of definitions does not, *per se*, determine the search path.

Where does this lead us? The structure can be customized, to an extent, once it is understood: for instance, storing a zero into a word's link field can stop a search, or redirecting a link can alter the search pattern. Remember, though, that some definition for the word whose name is the null character must remain in the search chain, or the system won't know how to deal with the end of a line. The usual definition is next to that of **QUERY**, or,

```
HEX 8081 HERE
: X  R>  DROP ; !  IMMEDIATE
DECIMAL
```

compiled above where you're going to zero a link field should allow you to experiment from the terminal (but not from screens).

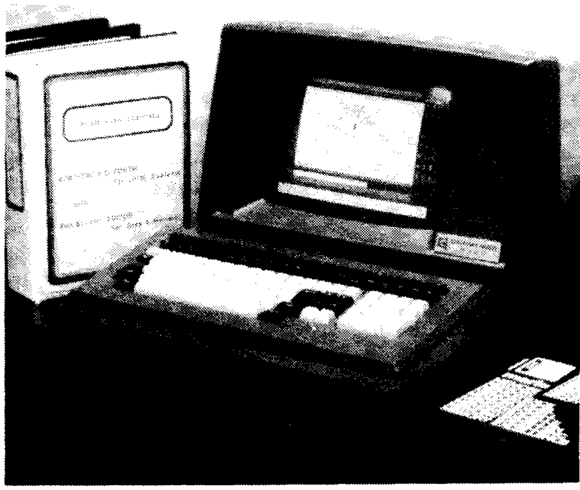
There are at least two major shortcomings to the present vocabulary organization:

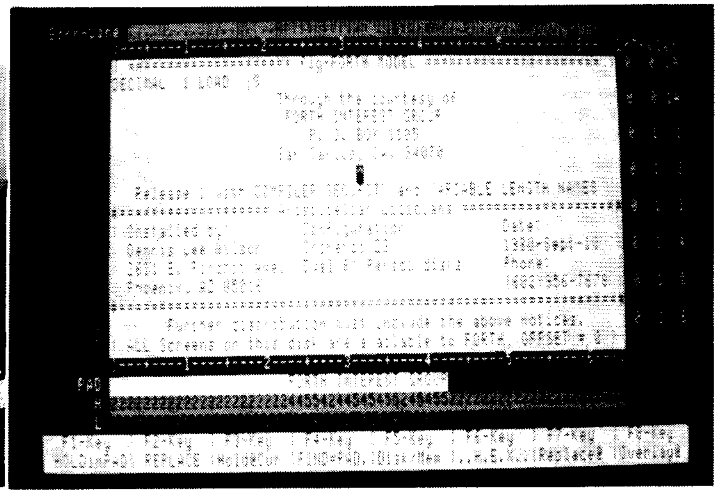
(1) No pointer is kept to the first word in a new vocabulary, only to the last; hence, rearranging the branches on the vocabulary tree is cumbersome.

(2) The search routine only looks at the "current" and "context" vocabularies, and is thus limited in regard to generalized search patterns.

In my next article, we'll look at some of the proposals for vocabulary structuring.

*Illustrative figures were kindly provided by Valpar International. —E.R.*





# 8080/Z80 FIG-FORTH for CP/M & CDOS systems
## FULL-SCREEN EDITOR for DISK & MEMORY

$50 saves you keying the FIG FORTH model and many published FIG FORTH screens onto diskette and debugging them. You receive TWO diskettes (see below for formats available). The first disk is readable by Digital Research CP/M or Cromemco CDOS and contains 8080 source I keyed from the published listings of the FORTH INTEREST GROUP (FIG) plus a translated, enhanced version in ZILOG Z80 mnemonics. This disk also contains executable FORTH.COM files for Z80 & 8080 processors and a special one for Cromemco 3102 terminals.

The 2nd disk contains FORTH readable screens including an extensive FULL-SCREEN EDITOR FOR DISK & MEMORY. This editor is a powerful FORTH software development tool featuring detailed terminal profile descriptions with full cursor function, full and partial LINE-HOLD LINE-REPLACE and LINE-OVERLAY functions plus line insert/delete, character insert/delete, HEX character display/update and drive-track-sector display. The EDITOR may also be used to VIEW AND MODIFY MEMORY (a feature not available on any other full screen editor we know of.) This disk also has formatted memory and I/O port dump words and many items published in FORTH DIMENSIONS, including a FORTH TRACE utility, a model data base handler, an 8080 ASSEMBLER and a recursive decompiler.

The disks are packaged in a ring binder along with a complete listing of the FULL-SCREEN EDITOR and a copy of the FIG-FORTH INSTALLATION MANUAL (the language model of FIG-FORTH, a complete glossary, memory map, installation instructions and the FIG line editor listing and instructions).

This entire work is placed in the public domain in the manner and spirit of the work upon which it is based. Copies may be distributed when proper notices are included.

| | USA | Foreign AIR |
|---|---|---|
| ☐ **FIG-FORTH & Full Screen EDITOR package**<br>Minimum system requirements:<br>80x24 video screen w/ cursor addressability<br>8080 or Z80 or compatible cpu<br>CP/M or compatible operating system w/ 32K or more user RAM<br>Select disk format below, (soft sectored only) ........ | $50 | $65 |

☐ 8" SSSD for CP/M (Single Side, Single Density)

Cromemco CDOS formats, Single Side, S/D Density

☐ 8" SSSD ☐ 8" SSDD ☐ 5¼" SSSD ☐ 5¼" SSDD

Cromemco CDOS formats, Double Side, S/D Density

☐ 8" DSSD ☐ 8" DSDD ☐ 5¼" DSSD ☐ 5¼" DSDD

***Other formats are being considered, tell us your needs.***

| | USA | Foreign AIR |
|---|---|---|
| ☐ Printed Z80 Assembly listing w/ xref (Zilog mnemonics) ........ | $15 | $18 |
| ☐ Printed 8080 Assembly listing ........ | $15 | $18 |
| TOTAL | $_____ | _____ |

Price includes postage. No purchase orders without check. Arizona residents add sales tax. Make check or money order in US Funds on US bank, payable to:

Dennis Wilson c/o
Aristotelian Logicians
2631 East Pinchot Avenue
Phoenix, AZ 85016
(602) 956-7678

# An Easy Directory System

*Will Baden*
*Costa Mesa, California*

A problem many have with Forth is remembering where things are located. The usual solution is to prepare a directory, *i.e.*, a screen with the names of things and the number of the screen which contains the thing named; or, if you are lucky enough to have a printer, you can list a hard copy of an index which contains the first line of every screen. Then you eyeball it for what you want.

A better way is to let Forth do the lookup for you. Define a word which will search the directory, find the screen number for you, and then push it on the stack. With this value, you can list it or do anything else you want with it.

**NAME** word **LIST**

This will list the screen where "word" is located.

A directory consists of alternating names (or subjects) and associated values in free format, beginning on the second line of each screen. The first line is reserved for heading and date. (To begin with line 0 instead of line 1, change **C/L** to **0** in **CONNIVE.**)

A fragment of a directory is shown in Figure One and will be used in our examples.

```
SCR # 78
( DIRECTORY        WWB/WWB 830714 )
INITIAL 30 STARTING-FORTH 32
KERNEL'S 96 SOLO 98 TRACE 89
FORMATTER 50      PRINT 66
DISCARD 100 RECREATE 100 RELOAD 100
PHONES 49
DOC-DIR  45 DOC-DIR-END 47

SORT 130 SORTED 130 ID< 131 NAMES 132
ALLNAMES 132 .NAMES 132
SWORDS 133
```

**Figure One**

Of course, the values in a directory do not have to be screen numbers. They could be anything that you want, *e.g.*, phone numbers, part numbers, or operating system constants.

Directories do not have to be in a neat order, and the user is responsible for maintaining them. Obviously, it is helpful if there is some semblance of order. For best performance, the most frequently used words should be at the beginning.

If you want to list the screen where the word **SWORDS** is located, then all you have to do is type **NAME SWORDS LIST** or you can say **VIEW SWORDS** and the result will be the same. **VIEW** is defined

```
: VIEW ( ---<name> )
  NAME LIST ;
```

This will list screen 133 where **SWORDS** is defined. The word **SWORDS** after **NAME** or **VIEW** is sought in the directory, the number 133 which follows the word **SWORDS** is pushed onto the stack, and screen #133 is listed. You can define other words like **VIEW** to perform any operation with the number on the stack.

If you want to load that screen, then another word named **NEED** can be used, *e.g.*,

**NEED SWORDS**

This will search the current working directory screen and then load the screen which has **SWORDS** defined on it. It checks to see if the word which is needed has been already defined; if so, it will not be loaded again. This word is very useful, since it can load other screens when and where they are needed. (See Figure Two.)

If any of the words needed are not already defined, they will be loaded before the rest of the screen. This way you can load the screens in any order as long as you have stated *on each screen* what words need to be loaded before that screen.

When screen 133 is loaded it checks for **SORT**, **ID<** and **NAMES**. If any of these is not defined it will load screen 130, 131 and/or 132 as appropriate. It then checks for **.NAMES** which just happens to be on the same screen as **NAMES** and so will always be already defined. "**NEED** something-else" may have been used on needed screens, and so forth.

This way all words will get their location from the directory. If at any time you move the screen to another location, just change the directory to show the proper screen number. Any screen which depends on the word whose screen location has changed will not be affected.

A directory is specified by two screen numbers: starting screen and ending screen. The system remembers these values for the current working directory. **DIR** will list the first screen of the current working directory.

To change the screen numbers of the current working directory, **ESTABLISH** can be used.

<starting-scr#> <ending-scr#> **ESTABLISH**

```
Screen 133 is

     SCR # 133
     ( "SWORDS" SORTED WORDS    WWB/WWB 820317 )

     NEED SORT   NEED ID<    NEED NAMES   NEED .NAMES

     : SWORDS ( -- )
       NAMES ( A,N )
       DUP CR .   ." NAMES DEFINED " CR
       2DUP SORTED ID<
       .NAMES ;
```

**Figure Two**

The defining word **DIRECTORY** at compile time takes two values from the stack which at run time will be used as arguments for **ESTABLISH**.

```
: DIRECTORY 2CONSTANT DOES>
  2@ ESTABLISH ;
```

You could define a word **DOC** which, when executed, would establish the current working directory for documentation as follows (assuming it to be on 45 through 47):

```
45 47 DIRECTORY DOC
```

You can even get the values from the current working directory

```
NAME DOC-DIR
NAME DOC-DIR-END
DIRECTORY  DOC
```

assuming that the current working directory has entries

```
DOC-DIR 45 DOC-DIR-END 47
```

**DOC** will change the current working directory to the documentation directory. Any number of working directories can be thus defined.

Every system will have a standard or default working directory. To get back to it we say **MAIN**. On this disk it is defined

```
78 82 DIRECTORY MAIN
```

The actual work of **NAME** is done by **SUBJECT**. **NAME** picks up the limits for the current working directory and executes **SUBJECT**. **SUBJECT** takes the next word in the input stream and puts it in **PAD**. It then does **CONNIVE**, which will look for that word on the screens indicated. Only the names are compared — the values are skipped over to make the search faster. **INTRIGUE** is used by **CONNIVE** to look up the word in **PAD** on a single screen. When and if the word in **PAD** is found, the next word will be interpreted.

**SUBJECT** can be used to define words similar to **NAME** and **VIEW** for special directories. A "help" system could be defined something like:

```
: HELP ( --- )
  HELP% 2@ SUBJECT LIST ;
```

With shadow screens it could be even easier.

```
: HELP ( --- )
  NAME >SHADOW LIST ;
```

```
SCR# 71
( you may already have some of these. )
-1 CONSTANT TRUE    0 CONSTANT FALSE
: DEFINED  ( --<name> a,f )  ( -' not -or- )
   -FIND  ( this is figforth "-FIND" )
   IF 64 ( precedence bit ) AND
      IF 1 ( it's immediate ) ELSE -1 THEN
      SWAP CFA SWAP
   ELSE HERE 0 THEN ;
: HAVE   ( --<name> f )  DEFINED SWAP DROP 0= NOT ;
( : word  word here ; )  ( homonym )
: COMPARE  ( a1,a2,n1 -- negative/zero/positive )
   OVER + SWAP
   DO COUNT I C@ - ?DUP
      IF SWAP 0= LEAVE THEN
   LOOP
   IF 0 THEN ;


SCR# 72
( easy directory system wwb/wwb 830714)
: MORE  ( -- addr,f )  ( bl word dup c@ -or- )
   BEGIN BL WORD DUP 1+ C@ BL OR BL -
      IF TRUE EXIT THEN
      BLK @ 1+ B/SCR MOD
   WHILE DROP  1 BLK +!  0 >IN !
   REPEAT FALSE ;
: INTERPRET-A-WORD  ( --<word> )
   ( interprets a word )
   DEFINED
   IF EXECUTE
   ELSE NUMBER ( dpl @ 0< if drop then )
   THEN ;
: CONTINUED  ( n -- )  ( b/scr * ) BLK !  0 >IN ! ;
73 LOAD   74 LOAD  ( directory system)
78 82 DIRECTORY MAIN   MAIN


SCR# 73
( easy directory system wwb/wwb 830714)
: INTRIGUE ( -- flag )  ( search the screen )
   BEGIN  MORE  ( addr,f)
      IF PAD  DUP C@ 1+  COMPARE ( 0 for equal )
         IF  BL WORD 0= ( 0) ELSE TRUE EXIT THEN
      THEN
   UNTIL  FALSE ;
: CONNIVE  ( scr1,scr2 -- )  ( <name> is in "PAD")
   BLK @ >IN @ >R >R   TRUE ROT ROT   1+ SWAP
   DO I ( b/scr * ) BLK !  C/L ( skip top line ) >IN !
      INTRIGUE IF NOT LEAVE THEN
   LOOP  ABORT" not in directory "
   INTERPRET-A-WORD
   R> R> >IN ! BLK ! ;
: SUBJECT ( scr1,scr2 --<name>)  ( find and execute )
   BL WORD COUNT PAD 2DUP C! 1+ SWAP CMOVE CONNIVE ;


SCR# 74
( easy directory system wwb/wwb 830714)
2VARIABLE DIR%
: %DIR          ( -- scr1,scr2 )  DIR% 2@ ;
: ESTABLISH   ( scr1,scr2 -- )  DIR% 2! ;
: DIRECTORY   ( scr1,scr2 -- )
   2CONSTANT DOES>  ( -- )  2@ ESTABLISH ;
: NAME   ( --<name> n -or- d )  %DIR SUBJECT ;
: VIEW   ( --<name> )                NAME LIST ;
: NEED   ( --<name> )
   >IN @ HAVE IF DROP ELSE >IN ! NAME LOAD THEN ;
: FOLLOW   (  --<name> )
   >IN @ HAVE IF DROP ELSE >IN ! NAME CONTINUED THEN ;
: RUN   ( --<name> )   >IN @ NEED >IN ! ;
: DIR   ( --<name> )  %DIR MIN LIST ;
: SUB   ( scr --<name> n -or d )  DUP SUBJECT ;


PROCEDAMUS   WWB   7/14/83
```

**End Listing**

A primitive phone list can be set up:

```
: .PH# ( DN --- )
  <# # # # # ASCII - HOLD
  #S #> TYPE SPACE ;

: REACH ( --- )
  [ NAME PHONES ] LITERAL
  DUP SUBJECT .PH# ;
```

Since the value of a name or subject is interpreted, **SUBJECT** could be used for menus. Assuming that **#MENU** is a screen with one- or two-character codes alternating with associated words to process them, something like the following could be done.

```
: MENU ( --- )
  #MENU LIST PROMPT
  QUERY 0 >IN !
  #MENU DUP SUBJECT ;
```

"**RUN** something" is equivalent to "**NEED** something something".

"**FOLLOW** name" has a somewhat similar relation to "**NEED** name" that "-->" has to "**LOAD**". It is used at the beginning of a screen to get to an earlier screen that will lead to the current screen. This allows an entire application, spanning several screens, to be loaded from a request for any one of the constituent words.

In the original conception, we thought that directories would be sparse, with only major entries like traditional directory screens. We soon found that all definitions could be put in the directory (and a utility to do this was developed). This makes **VIEW** act like its homonym, which vectors **CREATE**. An important difference is that our **VIEW** can list the source of words that are not yet defined.

Since June 1982, this system has been installed on Apple, Atari, CP/M and Heath Figforth, Micromotion Forth79, MVP Forth, and the Starting Forth dialect. The following utility words or their equivalent are required:

**MORE** returns (address,true) if there are more words in the input stream, (address,false) if the input stream is exhausted. Our definition should work for any system, even when **B/SCR** is not 1. On a standard system, the definition may be replaced with

```
: MORE ( --- a,f )
  BL WORD DUP C@ ;
```

(See Suralis and Brodie, "Checksum for Hand-Entered Source Screens," *Forth Dimensions*, Vol. IV, No. 3, p. 15.)

**INTERPRET-A-WORD** interprets the next word in the input stream. **NUMBER** is like the Starting Forth word, and returns a number or double number. In FIG-Forth you will have to remove the parenthesis marks from **DPL @ 0< IF DROP THEN**.

**DEFINED** returns a compilation address (which can be executed) and a true flag if the next word is defined, or a string address (which can be further massaged) and a false flag if the next word is unknown. It can be replaced with **-' NOT** in some systems.

**HAVE** returns **TRUE** or **FALSE** depending on whether you already have the next word or not. In Forth79 you may replace it with **FIND**.

All that is required of **COMPARE** is that strings at **HERE** and **PAD** can be compared for equality. **INTRIGUE** can be adapted to use the Starting Forth **-TEXT** or the FIG-Forth **-TEXT** instead.

**CONTINUED** is from the reference word set and goes to a screen with no return. It is used in the definition of **FOLLOW**. A "named -->" can be done

**NAME** word **CONTINUED**

If **B/SCR** is not 1 then remove the parentheses from around **B/SCR** * in **CONTINUED** and **CONNIVE**.

## First Screen of "FORMATTER" Directory

```
( FMT    K&P/WWB 830714 )
LINES 230 =? 230 DATE 230 .DATE 230 TODAY 230 PAGELEN 231
PAGEWIDTH 231 HUGE 231 MAXSTRING 231 CURPAGE 232 NEWPAGE 232
LINENO 232 PLVAL 232 M1VAL 232 M2VAL 232 M3VAL 232 M4VAL 232
BOTTOM 232 HEADER 232 FOOTER 232 FILLING 233 RJUST 233
LSVAL 233 SPVAL 233 INVAL 233 RMVAL 233 TIVAL 233 CEVAL 233
ULVAL 233 OUTP 234 OUTW 234 OUTWDS 234 DIRECTION 234
BETWEEN 234 CAP 234 ALLCAP 234 OUTBUF 234 DATE 235 .DATE 235
TODAY 235 PUTTL 235 PUTHEAD 236 PUTFOOT 236 PUTLINE 237 BR 238
PUTSPACE 238 PUTPAGE 239 GETPARAM 240 SETVAL 240 SETLS 241
SETCE 241 SETUL 241 GETTL 242 SETPAGE 242 SETSP 242 SETNE 242
SETIN 243 SETRM 243 SETTI 243 SETBOTTOM 244 SETPL 244
SETM1 244 SETM2 244 SETM3 244 SETM4 244 .FI 245 .NF 245
.BR 245 .JU 245 .RJ 245 .NJ 245 .LS 246 .CE 246 .UL 246
.HE 246 .FO 246 .BP 247 .SP 247 .NE 247 .IN 247 .RM 247
```

**Figure Three**

# A RAMdisk for 8086/8088 FIG-Forth

*John W. Irwin*
*Austin, Texas*

My IBM Personal Computer, with 320K of RAM, has far more memory than is used by most Forth programs. The desire to eliminate wear and tear on my diskette drives and diskettes prompted me to develop a RAMDisk application in Forth to make use of this resource. The program measures the unused RAM space remaining after Forth is loaded and makes the free space into a virtual diskette drive of that maximum capacity. The performance increase is impressive and the absence of the usual diskette commotion is welcome. By copying a set of screens to RAMDisk, program changes may be tried non-destructively and then copied back to the original screens when completely debugged.

## Non-Standard Words

My Forth, although similar to the FIG 8086 implementation, is a greatly expanded version with dictionaries for multi-tasking, full-screen, color, and provision for DOS-compatible, named disk files. The RAMDisk program presented here is a subset of my program; the omitted material pertains mainly to presenting the RAMDisk to the user as a DOS file.

This program makes use of several words from past issues of *Forth Dimensions*: the Kitt Peak **GODO**, and the modular programming words **INTERNAL**, **EXTERNAL** and **MODULE**. These definitions are presented for reference. The extended segment load and store words **EC@**, **EC!**, **E@** and **E!** in my system address the segment defined by a user variable **EB** (Extended Base). My input/output words are **IO@**, **IO!**, **IOC@**, and **IOC!**. The meaning is obvious and equivalent words may exist in other 8086/88 Forths or may be coded by the user. The assembler words used in **(MEM)** have obvious functions.

```
( Kitt Peak GODO and Modular Programming Words )


: (GODO)  2*  0 MAX  R @  4 -  MIN  R>  DUP DUP  @  + >R
          +  2+  @  EXECUTE ;

: GODO    COMPILE (GODO)  HERE  0 ,  2 ;  IMMEDIATE


: INTERNAL    CURRENT  @  @ ;      ( start private program section)

: EXTERNAL    HERE ;               ( end private program section)

: MODULE      PFA  LFA  ! ;        ( hide the private words)


R

SCR # 768
( RamDisk Program )          INTERNAL

0  CONSTANT  RAM#            ( total number of RamDisk buffers )
16384  CONSTANT  RAMBLK      ( first block number )
0  CONSTANT  RAMSEG          ( base segment address of RamDisk )

HEX

ASSEMBLE  CODE  (MEM)  12 INT  PSHAX  PSHCS  END-CODE

: RDmemsize  ( --- )  ( stores seg base, RAMSEG, and # bfrs )
     (MEM)  1000 +  DUP  ' RAMSEG !  >R  40 *  R> -
     11 /  ' RAM# ! ;
                                     DECIMAL     -->


SCR # 769
( RamDisk Program )          HEX

: RDclear  ( --- )  ( mark all buffers empty )
     RAMSEG  DUP  RAM#  11 *  +  SWAP
      DO
          I EB !        ( access a buffer )
          0 0  E!       ( mark buffer unused )
       11 +LOOP ;       ( to next buffer )

: RDseg  ( blk --- blk )  ( sets EB to buffer start )
     DUP  RAMBLK  -              ( get offset )
     11 *  RAMSEG  +  EB ! ;     ( calculate buffer address )

     DECIMAL  -->


SCR # 770
( RamDisk Program )          HEX

: RDwrite  ( bfr blk --- )       ( copy relative block )
     61 IOC@  3 OR               ( chirp beeper )
     61 IOC!
     61 IOC@  FC AND             ( quit beep )
     61 IOC!
     RDseg                       ( set buffer segment )
     0 E!                        ( write block number )
     102 2  DO
              DUP @  I E!   2+   ( copy block to RamDisk )
            2  +LOOP  DROP ;
                                          DECIMAL    -->
```

```
SCR # 771
( RamDisk Program )        HEX

: RDerr  CR ." RamDisk ERROR, improper function call"  QUIT ;

: RDread  ( bfr blk --- )        ( copy relative block )
    RDseg                        ( set buffer segment )
    0  E@  =                     ( see if data present )
    IF                           ( read actual buffer )
      102 2  DO                  ( copy RamDisk to buffer )
               I E@  OVER !  2+
             2 +LOOP  DROP
    ELSE                         ( RamDisk buffer not valid )
          100  BL  FILL          ( fill buffer with blanks )
    ENDIF ;
                                             DECIMAL    -->

SCR # 772
( RamDisk Program )

: RDR/W  ( addr blk f --- )  ( augments FORTH R/W )
    OVER  DUP                ( see if in RamDisk )
    RAMBLK 1-  >  SWAP  RAM# RAMBLK +  <  AND
    IF                       ( then memory read or write )
      1+  GODO
          RDerr  RDwrite  RDread  RDerr
         THEN
      R> DROP                ( drop return link to R/W )
    ELSE                     ( else real read or write )
      R>  SWAP  >R  >R       ( replace R> from R/W )
    ENDIF ;                  ( and return to R/W   )
                                                    -->

SCR # 773
( RamDisk Program )

: RDinit  ( --- )    ( initialize a phantom diskette in RAM )
    RDmemsize  RDclear  '  RDR/W    CFA  ' R/W ! ;

: RDparms   ( --- )    ( display RamDisk parameters )
    DECIMAL  CR  CR  ."  RamDisk "    CR  CR
    ." Capacity :  "  RAM#  4 / .  ." K-bytes"      CR
    ." Blocks   :  "  RAMBLK .
               ." - "          RAM# RAMBLK + 1-  .  CR
    ." Screens  :  "  RAMBLK  4 /  DUP  .
               ." - "          RAM# 4 / + . CR ;
                                                 -->

SCR # 774
( RamDisk Program )

EXTERNAL

: RD    ( 0 --- initialize RD,
          1 --- return initial block #,
          2 --- return number of blocks,
         >2 --- type information block )
    1+  GODO  RDparms  RDinit  RAMBLK  RAM#  RDparms  THEN ;

MODULE    FORTH

0 RD   3 RD       ( initialize RD and show user the size )
```

**End Listing**

## Program Interface

The RAMDisk program should be loaded before other applications. The only nucleus word affected is **R/W**. In order to make existing words, such as **FLUSH**, work with the RAMDisk, this program replaces the first parameter word of **R/W** (**R>**) with the execution address of **RDR/W**. **RDR/W** checks the block number to see if the call is to RAMDisk. If not, then the **R>** replaced at **'#R/W** is emulated and execution proceeds with the native **R/W** code. Since the RAMDisk program modifies a nucleus word, it should be "sealed" under **FENCE** to prevent accidental **FORGET**ing.

The assembler word **(MEM)** returns two values from which the available memory is determined. The PC-DOS call 12 **INT** returns the memory size in Kbytes. Pushing the CS register returns the beginning address of the Forth code segment. From these values, **GET-MEM-SIZE** determines space available for the RAMDisk so that the compiled application can be moved freely between machines with differing memory size. If Forth is loaded at a fixed location in a fixed memory-size machine, the word **(MEM)** can be replaced by a constant for memory size and a constant for the end of the Forth segment.

My Forth uses 256-byte blocks. Each block is assigned seventeen sixteen-byte 8086/88 "paragraphs". This wastes twelve bytes per block but simplifies buffer addressing. The extra segment pointer (**EB**) is set to the starting address of RAMDisk plus seventeen times the relative block number within the RAMDisk. The block identifier is then at offset zero and the block proper at offset two to 257 relative to the segment register (**EB**). This scheme is easily adapted to other buffer sizes. For systems with 1K blocks, each block is assigned sixty-five paragraphs.

## Program Functions

**RD** is the only RAMDisk word visible to the user. This word accepts a function flag as follows:

0 **RD** ( 0 --- )
clears RAMDisk, sets RAMDisk length.

1 **RD** ( 1 --- n )
returns initial block of RAMDisk.

2 **RD** ( 2 --- n )
returns number of blocks in RAMDisk.

>2 **RD** ( 3 --- )
types a summary block on the screen.

This function call interface is implemented by **GODO** and is very easy to interface to existing programs.

Clearing the RAMDisk ( 0 **RD** ) should be done before use and is done simply by writing zero to the block numbers in the RAMDisk. This operation checks the memory size and resets the RAMDisk origin and length, allowing the program to dynamically adapt to a different memory size and program load address each time it is used.

**RDR/W** is equivalent to FIG-Forth **R/W**. The stack at entry contains the standard **R/W** call ( addr blk# f ). If the block number is in the RAMDisk, a **GODO** is used to interpret the flag, else control is passed back to **R/W** with the flag moved to the R-stack.

**RDwrite** copies the Forth buffer into the RAMDisk buffer, including the block number. The update flag is never set in RAMDisk buffers since the RAMDisk represents the physical diskette. To provide some (very needed) feedback to the user that his **EDIT** is indeed being saved, a speaker click is emitted for each RAMDisk buffer write. This is done very simply in the two lines containing I/O words by gating and immediately de-gating a bit in the speaker port. These lines can be deleted for other systems or if feedback is not desired.

**RDread** compares the RAMDisk buffer block number to the requested block number to see if the buffer has been written. A valid buffer is indicated by a match while an invalid buffer contains the zero put there by initialization. If the buffer is valid, the RAMDisk buffer is copied to the Forth buffer, else the Forth buffer is blanked using **FILL**.

# In-word Parameter Passing

*Timothy Huang*
*Portland, Oregon*

This article and short program was stimulated by one of the L.A. FIG's handouts. Screen 312 is basically a copy from that. The word **&X** functions similar to **ASCII**, except that the parameter resides within the word, *i.e.*, the **X**.

The concepts from these words are so intriguing that I decided to explore them further (Screen 313). As normal Forth will not allow the parameter to be passed within the word's name itself, by adjusting the user variable **WIDTH**, one can play the game of passing the bulk through the name of the word. Screen 399 provides two examples of how **DUPX** will duplicate the **X**th stack entry to the top of the stack. This is similar to <**x**> **PICK**, except the index is included in the name. **DROPX** performs similar to the combination of <**n**> **ROLL DROP**.

The word **SWAPXY** (Screen 314) extends the same concept one step further in that it passes two single-digit parameters. It swaps the **X**th and the **Y**th stack entry. The word

**VSWAP** ( addr1 addr2 --- )

used in line 10 will swap the contents of two addresses <addr1> and <addr2>. This word can be defined as:

```
: VSWAP ( addr1 addr2 --- )
    2DUP     @  >R    @    SWAP  !  R>  SWAP  !  ;
```

Figure One displays some examples of usages.

```
SCR # 312
  \ &X CTRL-X $XX $XXXX                          TDH13DEC82
  DECIMAL
  1 WIDTH !
  : &X              \ put Ascii constant of X on stack
     HERE 2+  C@ [COMPILE] LITERAL   ; IMMEDIATE
  : $XX             \ put Hex constant of XX on stack
     BASE @  HERE 1+  NUMBER   ROT BASE !  [COMPILE] LITERAL ;
     IMMEDIATE
  : $XXXX           \ put 16 bit Hex constant of XXXX on stack
     [COMPILE] $XX    ; IMMEDIATE
  5 WIDTH !
  : CTRL-X          \ put Control character of X on stack
     HERE 6 +  C@  C/L -  [COMPILE] LITERAL    ; IMMEDIATE
  31 WIDTH !
  ;S
```

```
SCR # 313
  \ DUPX DROPX                                   TDH13DEC82
  DECIMAL
  3 WIDTH !
  : DUPX                        \ dup x_th item on stack
     HERE [ WIDTH @ ] LITERAL +  NUMBER  DROP
     [COMPILE] LITERAL PICK ;    IMMEDIATE
  4 WIDTH !
  : DROPX                       \ drop x_th item on stack
     HERE [ WIDTH @ ] LITERAL +  NUMBER  DROP
     [COMPILE] LITERAL ROLL DROP ; IMMEDIATE
  31 WIDTH !  ;S
```

```
SCR # 314
 0 \ SWAPXY                                      TDH13DEC82
 1 DECIMAL
 2 : X ( addr --- stack_addr )     \ get x_th cell address
 3    C@ ASCII  0 -  2*   SP@  2+  +  ;
 4 : Y ( addr --- stack_addr )     \ get y_th cell address
 5    1+ X    ;
 6 4 WIDTH !
 7 : SWAPXY                        \ swap the x_th & y_th entries
 8    HERE [ WIDTH @ 1+ ] LITERAL +  DUP
 9    X   SWAP   Y
10    VSWAP   ;   IMMEDIATE
11 31 WIDTH !
12 ;S
13
14
15
```

**End Listing**

```
9 8 7 6 5 4 3 2 1 CR S.
9 8 7 6 5 4 3 2 1 OK
DUP5 CR S.
9 8 7 6 5 4 3 2 1 5 OK
DROP CR S.
9 8 7 6 5 4 3 2 1 OK
DROP7 CR S.
9 8 6 5 4 3 2 1 OK
SP! OK
9 8 7 6 5 4 3 2 1 OK
SWAP37 CR S.
9 8 3 6 5 4 7 2 1 OK
SWAP29 CR S.
2 8 3 6 5 4 7 9 1 OK
SWAP15 CR S. 5 SPACES SWAP51 S.
2 8 3 6 1 4 7 9 5     2 8 3 6 5 4 7 9 1 OK
```

**Figure One**

# Stack-Oriented Co-Processors and Forth

*Dana Redington*
*Redwood City, California*

Ideally, a computer can be adapted to a wide variety of laboratory situations, provided that two conditions are met. The first condition requires using an appropriate, interactive environment. Here, Forth provides one of the best alternatives. The second condition usually requires extending software to meet current needs. Since Forth is intentionally extensible, this means molding the environment to fit the situation by enhancing the dictionary. Occasionally, increasing the vocabulary is not sufficient and more direct enhancements are needed in the form of hardware, as in floating-point computation.

This paper focuses on hardware enhancements to the Forth environment. It briefly reviews the structure of Forth, introduces co-processing, outlines the 8087 numeric processor with example words, and suggests the future of stacks in Forth.

**The Structure of Forth**

Forth is a unique, interactive language/environment. It is an example of what Loeliger (1981) calls a "threaded interpretive language." Additionally, Forth utilizes stacks, as do other languages (like UCSD Pascal). But, it is more than just an interactive, stack-oriented, threaded interpreter. The sum, in this case, is greater than the individual parts. Hofstadter's (1979) description of "strange loops" forming emergent phenomena is appropriate for describing what happens in Forth: "an interaction between levels in which the top level reaches back down towards the bottom level and influences it, while at the same time being itself determined by the bottom level." ( p. 709 ).

Using this analogy, the stack resides near the bottom level of Forth. The stack is a temporary place to store and transpose elements. Usually, stack elements are inferred to be 16-bit numbers even though other types of elements exist (*e.g.* character strings on a string stack or sprite planes on a graphics stack). The stack elements are stored one on top of the other where the most recently placed element on the stack is usually the first to come off, that is, a last-in-first-out stack. The stack is also a place to transform elements—using the (reverse polish) number sequence: "1 3 + ." yields "4 OK".

There are different types of stacks in Forth. The type is determined by the meaning of the stack elements—what function the elements serve. A data or parameter stack is used to store elements that usually represent data or the address of a variable. A return stack is used to store numbers that usually represent program flow-control parameters like the code field address of the next word to be executed.

Unfortunately, in Forth it becomes increasingly awkward to deal with numbers of larger sizes as in the "ripple of the carry bit" problem beyond 16 bits. The problem of larger numbers becomes more apparent on 16- and 32-bit computers. Examples include attempting to access memory beyond the 64K byte limit and dealing with numbers well beyond 16 bits as in quadruple-word arithmetic or floating-point computation. In such cases an alternative is necessary.

There are three primary alternatives to augmenting the Forth stack environment. They are, in increasing order of complexity: (1) simply devising Colon and/or Code definitions like writing floating-point routines in software and (2) adapting memory-mapped or ported hardware like a floating-point processor (such as adding a 9511 or 9512) with the necessary interface words, or (3) incorporating a co-processor specially equipped to deal with the desired stack elements. Incorporating a co-processor is the most interesting, but seldom used, alternative.

**Stack-Oriented Co-Processing**

Co-processing is a special form of multi-processing. The co-processor, as a guest, lacks some of the faculties of the host processor. The guest must rely on the host for some faculties such as memory segmentation and address generation. One advantage of co-processing is that almost no overhead is incurred in setting up the guest to execute an instruction when the host and guest are working in unison. A second advantage of co-processing is that the guest can be performing a complex calculation (like raising a number to the *i*th power) while the host is performing a few "housekeeping" chores; this is referred to as an asynchronous co-processing mode. A third advantage of co-processing is that host and guest can work together in what is called maximum synchronized co-processing; the host "waits" until the instant the guest has completed a computation before continuing with the instruction stream.

Obviously, a co-processor is a device that augments a processor by extending and/or redefining the host's capability. An ideal co-processor shares the host's resources. This shared resources approach also has a severe hardware limitation: a special co-processor must exist for a specific microprocessor. And in the case of Intel's 8086, a special 8087 Numeric Data Processor (NDP) is available. (The 16081 numeric co-processor exists for National's 16000 series and Motorola is developing an NDP for the 68000).

**The 8087**

The 8087 is a true co-processor. It augments the register and instruction sets of the host 8086(88) microprocessor; it enhances the variety of numeric data types; and it accelerates the 8086(88)'s numeric computation capabilities. The (5Mhz) 8087 was initially introduced in July 1980. A newer, faster (8 Mhz) 8087 is projected for January 1984. The 8087 has eight 80-bit registers, two pointers, a control register and a status register; supports seven data or number types; performs all computations on a temporary real format (80 bit); and has six principle instruction types. Detailed technical information on the 8086(88)/8087 can be found elsewhere.[1]

One of the unique features of the 8087 illustrates how co-processing instructions are interpreted. In a maximum synchronized mode, the NDP interprets the instruction stream along with the host processor. The NDP remains poised until a special command sequence is detected (ESC). When the special co-processor command is read, the 8087 interprets the subsequent commands, accessing memory as necessary and executing the interpreted commands. The 8086 waits until the NDP has completed the commands before continuing. In a true co-processing fashion the 8086(88) and 8087 interpret the same instruction stream containing embedded NDP commands.

The 8087 was designed with a stack structure: "the charter of the 8087 design team was first to achieve exceptional functionality and then obtain high performance." (*iAPX manual*, p. S.3). The 8087 is a Stack-Oriented Co-processor (maybe an "SOC"?). This structure makes it compatible not only with the 8086 but also with the architecture of Forth.

**Forth and a Stack-Oriented Co-processor**

The stack-oriented structure of the 8087 provides an easy incorporation into an 8086 Forth environment. All that is needed is a small 8087 assembler that contains the primitive commands to communicate with the NDP (this kind of assembler is described elsewhere[2]). The operation of a stack-oriented co-processor is identical to the operation of existing stacks; two examples help illustrate the simple extensibility of stack-oriented co-processing.

The first example involves a variant of integer computation. Usually, one integer is placed on the stack, then a second integer is placed on the stack and in reverse polish fashion a command is given to multiply the two numbers; for example,

```
1234 5 * .
```

would produce

```
6170 0 OK
```

My 8 Mhz 8086 Forth environment performs 3230 (integer multiply) operations per second, or 3.23 KOPS (pronounced "K-OPS").

In the co-processing case, a similar sequence is repeated. Again two numbers are placed on the stack. But, instead of issuing a * command, which would result in the 8086 multiplying the numbers, four additional steps must take place: (1) the first number must be moved to the co-processor's stack (the stack that physically resides within the 8087 chip; for this the word **W>F** defined in the 8087 assembler moves an integer or word from the data stack to the floating-point stack); (2) the second number must be moved to the 8087 stack, again using **W>F**; (3) the numbers must be multiplied using **F***: (the floating-point analog of *; and then (4) the resulting number must be returned to the Forth data stack using **F>W** (which converts a floating-point number to a 16-bit integer and transfers it to the top of the data stack). The typed sequence

```
1234 5 W>F W>F F* F>W .
```

would produce, again

```
6170 OK
```

In this case "more is less." Even though four steps are required, the 8086 + 8087 actually performs floating-point multiplication at 29.41 KOPS. The addition of a numeric co-processor increases the computational speed for integer multiplication by about nine times.

As a side point, the NDP is capable of performing in excess of 88,8000 floating-point multiplies per second. But the observed computational speed is slower due to the overhead needed to run Forth. (What if a Forth co-processor chip existed for **DOCOL, SEMIS,** and **NEXT**?)

The second example of stack-oriented co-processing involves extending the Forth environment. This example is an extension of the first one. A significant difference between Forth and other languages and/or environments is the Divide-Test-Conquer approach; first divide the application into easily testable parts, test each part, and then conquer the application. Having already tested the fast variant of an integer multiply, we can now define a new word and extend the vocabulary. For added clarity let us define the new word as:

```
CODE * (W>F) (W>F)
(F*) (F>W) NEXT;
```

The words in parentheses are the primitives of their "colon-level" counterparts. **CODE** * operates on the stacks in an identical fashion as:

```
W>F WF F* F>W
```

but at 45.45 KOPS.

Additionally, after this word has been defined, any subsequent applications of * uses the "newer" and faster definition. The definition for integer multiply has been literally redefined. This Forth feature is rarely found in other languages. For example, attempting to redefine the integer multiply in UCSD Pascal is next to impossible.

**The Future of Stack-Oriented Co-processors and Forth**

Stack-oriented co-processors provide a means of extending a Forth system. For now, numeric co-processors are easily applied. Their addition not only extends computations to include

1. Good references on the 8087 include: Duncan (1982), Field (1983), Palmer, *et al.*, (1980), Rash (1981), Simington (1983).

2. I resurrected an 8086/8087 assembler written in Forth by John Bumgarner on my Seattle Computer Products 8086/8087 (Gazelle) system running at 8 Mhz. John Bumgarner and myself are completing a draft of an article on "An Extensible Assembler for the 8087." The 8087 was placed in a copper "girdle" (only recommended for Forth artisans) to enhance heat dissipation at the increased clock frequency (thank you TZ). I have also been beta-testing a newer (non-girdled) 8 Mhz 8087 (thank you LM, JT, DC).

floatingpoint calculations but also enables microprocessor systems to rival larger mini and mainframe computers in number crunching ability. In the near future, other types of stack-oriented co-processors may become available, including string and graphics co-processors. Until a Forth processor is available, and maybe even after, a Forth co-processor chip also provides a means of extending a Forth system. In the co-processor approach, very powerful microprocessor systems can be built of various combinations of guest processors and host processors. Ultimately, each element type that requires a stack might have a custom co-processor.

The most important benefit of stack-oriented co-processors is their special ability to operate on an internal stack of predefined elements. These processors provide an effective balance of increased speed with a minimum of additional hardware and additional vocabulary. The interaction between Forth and stack-oriented co-processors forms a strange loop where the sum is conveniently greater than the computational parts.

*This paper has evolved from two earlier papers presented at the 4th Annual Forth Convention, San Jose, 9 October 1982 and at the Eighth Annual West Coast Computer Faire, San Francisco, 18-20 March 1983. Address communications to: D. Redington, Sleep Research Center, Department of Psychiatry and Behavioral Science, Stanford University School of Medicine, Stanford, California 94305.*

**References**


Duncan, R. "Intel's 8087 Numeric Data Processor." *Dr. Dobb's Journal*, 1982, 7(8), pp. 47-50.

Field, T. "The IBM PC and the Intel 8087 Co-processor, Part 1: Overview and Floating-Point Assembly-Language Support." *Byte*, 8(8), 1983, pp. 331-374.

Hofstadter, D. R. *Godel, Escher, Bach: an eternal golden braid.* New York: Basic Books. 1979.

Loeliger, R. G. *Threaded Interpretive Languages.* Peterborough, NH: Byte Books. 1981.

Palmer J., Nave R., Wymore C., Koehler R., & McMinn C. "Making Mainframe Mathematics Accessible to Microcomputers." *Electronics*, 8 May, 1980, pp. 114-121.

Rash B., "Application Note AP-113 — Getting Started with Numeric Data Processors." Intel Corporation, February 1981.

Simington, R. B. "The Intel 8087 Numerics Processor Extension." *Byte*, 8(4), 1983, pp. 154-172.

*iAPX 86,88 USER'S Manual.* Intel Corporation, 1981.

# Code and Colon Compatibility

*David Held*
*Hermosa Beach, California*

I recently developed an approach to a problem which may be of interest to others. I was developing a communications application in Forth-79 which would require some code definitions due to speed requirements. For the development process, however, I preferred to begin with colon definitions, planning to convert progressively higher-level words into code as the work progressed. Thus, I faced the problem of creating code definitions for words that might be called either as subroutines from other code words, or as Forth-compatible words from colon definitions.

The critical difference between the two is that a machine-language subroutine should end by returning to its caller (pop the stack for the caller's address), whereas a Forth code definition ends by jumping to **NEXT**. To resolve the dilemma, I used the techniques illustrated in Figure One by 8080 machine language.

These words would permit me to make calls to any machine-language subroutine (such as in my system's monitor); for example,

```
: SCROLL   F010 CALL  ;
```

would write **F010** over the zero in the definition of **(CALL)**, and would then execute **(CALL)**. Thus the monitor subroutine at **(F010)** would be executed, followed by a jump to **NEXT**.

Now we want to create subroutines that can be used either from colon definitions or code words. Here is a defining word which defines such subroutines. When invoked from colon definitions, the run-time behavior is similar to **CALL**, above. To use the subroutine from code definitions, merely "tick" its address and **CALL** it in assembly language. Figure Two shows the defining word **SUBROUTINE**:

For example, this word might be used to define subroutine **SUB1**, as follows:

```
SUBROUTINE SUB1
80  A  MV1
C010 STA
RET
```

Now we can invoke **SUB1** from Forth, as follows:

```
:  TEST1   SUB1  ;
```

Or, from a machine-language word, as follows:

```
CODE  TEST2
'  SUB1  CALL,
NEXT  JMP,
END-CODE
```

The advantage we gained by all this manipulation is that the definition of **SUB1** is unchanged, whether it is used by a colon or code word.

```
CODE (CALL)     ( a word which calls a subroutine, then jumps to NEXT )
        CD C,  0 ,         ( equivalent to CALL 0 )
        NEXT JMP,          ( equivalent to JMP NEXT )
        END-CODE

' (CALL) 1+   CONSTANT CALL-ADDR      ( a constant containing the address     )
                                      ( of the zero in the above definition.  )

: CALL      ( addr --- )  ( a word which calls the subroutine at addr       )
        CALL-ADDR    ,              ( write the desired subroutine address  )
                                    ( over the zero in definition of (CALL)
        (CALL) ;                    ( and execute (CALL)
```

**Figure One**

```
: SUBROUTINE    ( create a code subroutine useable either from colon )
                ( or code definitions.                               )

        [COMPILE] ASSEMBLER      ( invoke assembler vocabulary          )
        CREATE                   ( create a header for the subroutine   )
        DOES>  CALL  ;           ( this happens when subroutines so     )
                                 ( defined are invoked from FORTH       )
```

**Figure Two**

# CORDIC Algorithm Revisited

*Dave Freese*
*Cape May Court House, New Jersey*

I am employed as a senior engineering analyst and routinely use PL/I, Pascal, Fortran and BASIC at my place of employment. Until recently, Forth was just for hacking around at home, but it may just prove to be the answer to a "real" estate problem at work. I am developing special purpose wind speed/direction instruments for the U.S. Navy and have been frustrated by the "fat" code produced by all of the compilers at my disposal. None of the compilers have the option of pruning the object code by removing unwanted support code. When you are downloading the resultant code to ROMs, a trade-off must be made between code size and speed. Forth may just let me have the best of both worlds.

Which brings me to the subject of source code. The Volume IV, Number 1 issue of *Forth Dimensions* has found a permanent place on my desk. It contains some of the best material on fixed-point arithmetic that is available (at least with regard to Forth). The article on vector rotation using the CORDIC algorithm was particularly useful to me, as that type of conversion is routinely performed in wind speed/direction computation. The original code did not, however, meet my expectations with regard to accuracy. In particular, it failed to return correct values for rotations of 0, 45, 90 and 180 degrees. The accompanying listing of a double-precision version of the algorithm will provide the needed accuracy. It converts all input into double-precision numbers with the binary point between bits 15 and 16 (*i.e.*, the upper half of a number represents the integer part and the lower half represents the fractional part.

This was written for Z80 Forth by Laboratory Microsystems. This interpreter allows double number literals in colon definitions. Modify these entries for FIG-Forth or other interpreters which do not allow this extension. The conversion is fast due to the machine code divide-by-factor-of-two, 2SRA routine.

```
Screen # 27                         crc = 25114
 ( CORDIC ALGORITHM -- words for machine code )

HEX

 ( create header with CFA pointing to body of word )
 : :CODE ( -- )  BASE @  HEX  CREATE ;

 ( terminate body of word with a jump to NEXT )
 ( jp next )
 : ;NEXT  ( -- ) 0C3 C,  NEXT-LINK ,  SMUDGE
   BASE ! ;

 DECIMAL



 -->
```

```
Screen # 28                         crc = 5781
 ( CORDIC ALGORITHM -- double number words )
 ( 2DUP 2SWAP 2DROP D+ D- 2@ 2!  previously defined )
 : 2OVER ( d1 d2 -- d1 d2 d1 )  >R >R 2DUP R> R> 2SWAP ;
 : 2ROT ( d1 d2 d3 -- d2 d3 d1 ) >R >R 2SWAP R> R> 2SWAP ;
 : 2VAR ( -- ) <BUILDS 0 0 , , DOES> ;
 : 2CON ( -- ) <BUILDS , , DOES> 2@ ;
 : D< ROT 2DUP = IF ROT ROT DMINUS D+ 0< ELSE
     SWAP < SWAP DROP THEN SWAP DROP ;
 : D* ( d1 d2 -- d3 )  OVER 5 PICK U* 6 ROLL 4 ROLL * +
     2SWAP * + ;

 :CODE 2SRA ( d n -- d/2^n ) 0E1 C,  07D C,  0B7 C,
     0CA C,  NEXT-LINK ,    0E1 C,  0CB C,  02C C,  0CB C,
     01D C,  0E3 C,  0CB C,  01C C,  0CB C,  01D C,  0E3 C,
     03D C,  020 C,  0F3 C,  0E5 C,  ;NEXT
 -->
```

```
Screen # 29                         crc = 51411
 ( CORDIC ALGORITHM )     -->
 Zilog mneumonics for definition of 2SRA

          POP     HL              $1:  SRA    H
          LD      A,L                  RR     L
          OR      A,A                  EX     (SP),HL
          JP      Z,NEXT               RR     H
          POP     HL                   RR     L
                                       EX     (SP),HL
      cont'd next column               DEC    A
                                       JR     NZ,$1
                                       PUSH   HL
                                       JP     NEXT
```

```
Screen # 30                    crc = 16435
( CORDIC ALGORITHM )
( ALPHA[i] = 65536 * 32768 * arctan[1/2^i] / Pi )
2VAR ALPHAS -4 ALLOT
536870912. , , 316933406. , , 167458907. , ,  85004756. , ,
 42667331. , ,  21354465. , ,  10679838. , ,   5340245. , ,
  2670163. , ,   1335087. , ,    667544. , ,    333772. , ,
   166886. , ,     83443. , ,
( convert double to single with round up )
: D->S 32768 0 D+ SWAP DROP ;
: RVSUB1 >R 2ROT 2ROT 2OVER 2OVER R> 2SRA ;
: RVSUB2 >R 2ROT 2ROT 2SWAP R> 2SRA ;
: RVSUB3 >R 2ROT R> 4 * ALPHAS + 2@ ;
: *KN ( n -- d = n * 65536 * 0.60725293 )
    256 /MOD SWAP >R S->D 10188014. D*
    R> S->D 39797. D* D+ ;
-->
```

```
Screen # 31                    crc = 23246
( CORDIC ALGORITHM )
: ROTVECTOR ( n.y-old n.x-old n.ang -- n.y-new n.x-new )
  >R ( save angle )
  >R ( save n.x )   *KN ( convert n.y )
  R> *KN ( convert n.x ) 0 R> ( retrieve angle -> double )
  2DUP  0 0  D<  IF 0 16384 D+ 2ROT 2ROT DMINUS 2SWAP 2ROT
        ELSE  0 16384 D- 2ROT 2ROT 2SWAP DMINUS 2ROT  THEN
  14 0 DO
    2DUP 0 0 D< IF
         I RVSUB1 D- I RVSUB2 D+ I RVSUB3 D+ ELSE
         I RVSUB1 D+ I RVSUB2 D- I RVSUB3 D- THEN
    LOOP 2DROP ( drop d.angle )
  D->S  >R   D->S  R>  ;     -->
```

```
Screen # 32                    crc = 40282
( CORDIC ALGORITHM )
( single precision angular conversions )
: PIRADIANS  MINUS 32768 ROT ROT */ ;
: DEGREES 180 PIRADIANS ;
: POLAR-> ( rad ang -- y x ) 0 ROT ROT ROTVECTOR ;
;S
```

**End Listing**

Standards Corner

# Forth-83 Standard

*Robert L. Smith*
*Sunnyvale, California*

As many readers are aware, the Forth-83 Standard has been approved by the Forth Standards Team. By the time you read this, copies of the standard should be available from the Institute for Applied Forth Research, MicroMotion, or Mountain View Press. The majority of members of the Forth Standards Team are vendors or potential vendors of Forth systems and applications. The Forth-83 Standard represents a substantial input from the team members, the referees, and the Forth community. Literally hundreds of proposals were received and examined. In contrast to the past, there were two major meetings of the Standards Team as a whole, and very many meetings of the referees. The result is a document of substantial quality. The team rules require a two-thirds affirmative vote of the members to accept the new standard. The actual vote as of the time of this writing is twenty-two "yes" votes, one "no" vote, and three votes not received. We can see that the vote was quite decisively in favor of the new standard. In my opinion, the new standard offers a significant improvement over previous Forth standards.

Like all standards, Forth-83 is the result of many compromises and, therefore, not all readers will agree on the desirability of some of the features. It should be pointed out that the Forth Standards Team is not forcing anyone to adhere to the standard. In the forward to the standard, the following sentence appears: "A programmer or vendor may choose to strictly adhere with the standard, but the choice to deviate is acknowledged as beneficial and sometimes necessary." Certainly, if one has programs which work on an older standard or a non-standard system, there is no requirement that the old system be thrown away and the programs rewritten just because a new standard exists.

Let us briefly review some of the differences from the previous standard which may affect a more general acceptance of Forth-83. Most of the issues have been previously aired in this column for the purpose of information and to encourage public input.

The new **DO-LOOP** is somewhat different from previous **DO-LOOP**s. Most people now seem to prefer the new, circular-arithmetic **DO-LOOP**. Briefly, the advantages are (1) the index **I** now has a full 65K range, (2) there is no longer a need to have a separate **/LOOP** for unsigned indices, and (3) in most cases the new loop is faster than most older ones. A few vendors would prefer that

```
0 0 DO ... LOOP
```

would cause a null result. It appears that if that result is desirable, it could be obtained by using the new System Word Set and defining the desired function with a different name. Note that old code which used the construct

```
0 0 DO
```

would be incompatible in either case. A closely related issue is that of the new version of **LEAVE** which causes control to transfer to the end of the loop. There is an implementation issue here. There is not adequate space to discuss the issue thoroughly, but one vendor would prefer to have either a more complicated form (called **LEAVES**) or, alternatively, to allow only one occurrence of **LEAVE** within a given loop.

The default value of "true" for comparison operators now returns all bits set rather than just the low-order bit. In most cases, a comparison is followed by a test for non-zero, such as the word **IF**. In that case there will be no difference. If old code uses comparisons in conjunction with arithmetic operations, then some change will be required to work under the new standard. The simplest change is to follow the comparison with a negation operator. The new default value for "true" should be somewhat more useful than the old value. A related side benefit is that the word **NOT** is now available to mean "take the one's complement," whereas previously it was synonymous with **0=**. In most cases it is compatible with previous usage.

Historically, division in Forth has varied from system to system. According to Charles Moore, if a machine had a hardware divide, then its characteristics determined the division result. If division had to be done in software as with, say, the 8080, then floored division was usually chosen. However, only positive denominators were generally considered. In some cases **/** took signed arguments and **/MOD** took unsigned arguments. In Forth-83, the result of the **/** operation is the mathematical floor of the real number quotient. Alternatively, one may say that the quotient is more useful than the 79-Standard version. For example, one can readily perform an arithmetic right shift by dividing by an appropriate power of two. In hexadecimal arithmetic,

```
8712 100 /MOD
```

will yield a quotient of FF87 and a modulus of 12, so that the original number is readily split into 8-bit components. Under 79-Standard, the operation would yield FF88 with a remainder of FF12. A nice result is that now the right-shift operator **2/** is identical to **2 /**. In my opinion, the enhanced utility of the Forth-83 quotient and modulus function outweighs the disadvantage that some older code may need to be somewhat modified when negative arguments are employed.

In many cases, previous ambiguities have been resolved or clarified. There is at least one word, **]**, which is ambiguous in certain rare cases. It is important to realize that the new standard has not changed the meaning of this word from previous standards. The problem arises from alternative

*(Continued on page 30)*

# Forth-83: A Minority View

*Glenn S. Tenney*
*Belmont, California*

Having participated as a member of the Forth Standards Team (FST) and as a referee of the 83-Standard, I am strongly in favor of a new Forth Standard. Despite the many weeks I devoted to this, my professional conscience did not allow me to vote to accept the 83-Standard. For these reasons, I was asked to document my concerns as well as those of others. As you read this, please remember that many of the following are the concerns of others and do not necessarily represent my own views.

The Forth 83-Standard has recently been accepted and is now being published. Many people have been expressing varying degrees of concern over this new standard. After discarding a certain amount of gripes, these concerns fall into one of four categories: minor and non-technical; incompatibilities to the prior standard or existing systems; specific technical points; and general philosophical points. This is an overview of some of those concerns.

The new standard is almost completely reworded. This was done to make it more readable, yet this generated some fairly minor concerns, the most obvious being the process of locating underlying technical aspects that were actually changed. In fact, the new wording actually changed the published version of the standard so that **BASE** and **D<** reflect a technical change from the 79-Standard.

The new standard is incompatible with prior systems partly because the function of many words changed yet the word names remained the same. These functional changes range from obscure to obvious. This shows itself in the following ways:

- **PICK** and **ROLL** were changed, with no strong technical reason, from 1 origin to 0 origin, totally invalidating prior source code.
- Many changes were made, albeit for valid technical reasons, which are not always incompatible. These are the most dangerous, since when the incompatibility crops up it is often buried inside a previously functional definition.

The following changes affect many standard words:

  * The true flag (-1) returned from standard words provides some extra power, but can be incompatible if the flag is used in calculations.
  * Division is now floored towards negative infinity rather than towards zero.

83-Standard does not have state-smart required words. This seems to have grown from the problems using '(tick). In 79-Standard, **FIND** wasn't of much use so most people used the state-smart '(tick). This led to problems, since some state-smart words must be used cautiously when compiled within a colon definition. The solution adopted was to make all words state-dumb and to add words to handle some of the lost functions. **FIND** was later made useful, yet state-dumb words remain.

A side effect of the state-dumb "sweep" is that some words were made state-dumb that would have been better left state-smart regardless of the '(tick) decision. For example, there is no reason why **ABORT"** should be state-dumb. The system-dependent functions associated with **ABORT"** only appear within **ABORT"**. **ABORT"** could have been of great use outside of a colon definition, yet that has now been precluded.

Some current systems use mono-addressing (*e.g.* CFA or PFA), yet the standard now dictates dual-addressing (*e.g.* CFA and PFA). This tradeoff favors an easier implementation and was the subject of many hours of discussion. Some of the original dissentors still believe that a mono-addressing system is easier to use.

Disk (mass storage) I/O has been changed to disallow altering the data within a block buffer unless **UPDATE** is also used. Even if **EMPTY-BUFFERS** (which is no longer a required word) were used, the alterations might still be written to disk. Combined with the multi-tasking implications, **BUFFER** can no longer be used for a temporary scratch or data collection area. Although this codifies good programming practice, there are valid reasons why the standard should not be so restrictive.

A minor but grating problem is that the FST itself is always trying to allow a wide variety of actual or possible implementations. This has caused some problems and ambiguities within the standard. This is more an underlying political problem than a technical one, but the results affect the standard technically:

- The **]** (right-bracket) definition is ambiguous. Different systems are implemented such that the same supposedly transportable standard program produces different results on the different systems.

- Vocabularies have always been a problem. The standard attempts to resolve this issue, but actually leaves some gaping holes. Because search orders are not definitively stated, a standard program will have some difficulty using vocabularies and search orders.

Consumers will have great difficulty determining if a system is actually Forth-83. Although this has always been a problem, it will be worse this time because 79-Standard systems have been marketed. Implementors will have difficulty determining or choosing to make some of the subtle changes from 79-Standard.

The standard should codify existing practice. Instead, new concepts have been accepted without a complete test-bed existing with which to gain experience. In most cases, these new concepts are going to be proven correct, but what about the ones that aren't? Where will we be in four or five years if next month we find that one of these insufficiently tested concepts causes a problem?

It is difficult to judge the impact of some of these technical or philosophical concerns. Time and experience will tell whether the decisions made for Forth-83 were correct. Although it is difficult, we must temper our frenzy for having a new standard with a striving for reasonable perfection of the standard.

## Letters *(Continued from page 4)*

### 8080 Conditions

Dear FIG:

John Cassaday left conditional calls out of his 8080 assembler, which he published in *Forth Dimensions* (Vol. III, No. 6). I have written a single word that adds all the 8080 conditional calls to his assembler:

```
: IFCALL  SP@  0D  TOGGLE  C, , ;
```

The stack for proper use is:

<call-addr> <conditional> ---

Where <conditional> is any one of John's words **0=**, **CS**, **PE**, or **0<**, and an optional **NOT**. John's conditional words leave an 8080 conditional jump op-code on the stack. My word toggles bits in it to make it into the appropriate conditional call op-code. The call-address must be next down on the stack. A **LABEL** word can be used to define the entry point to the subroutine.

Using similar techniques, I have also written words for conditional return from subroutine and conditional jump:

```
: IFRET  SP@ 0A TOGGLE  C, ;
: IFJMP  SP@  08  TOGGLE  C, , ;
```

**IFJMP** needs a jump address on the stack, under the conditional, but **IFRET**, of course, needs only a conditional on the stack.

I think these words are yet more examples of an amazing property of Forth: the solution to a problem is usually less complicated than you think it will be!
Sincerely,

Paul E. Condon
6219 Rockwell St.
Oakland, CA 94618

### Yet Another Case Statement

The main feature of this **CASE** is the technique used. It is presented as an educational example of the power of Forth. In this example, **CASE** is a defining word which creates **TEST**. If you'll notice in the definition of **EXECUTES**, it looks like I'm jumping into the PFA instead of the CFA. This is because I am.

I was going to explain this, but after giving it some thought, I decided that it would be more beneficial to our fellow Figgers to present it as a puzzle. By the time you figure it out, you will understand **DOES**, the return stack, and the Forth compiler inside out.

As far as how this **CASE** compares to other **CASE**s: it compiles much smaller than most, it branches to **ENDCASE** upon finding a match, and executes about as fast. It will only execute one word after **EXECUTES** and that must be a word created in a colon definition. I don't use this **CASE** myself, nor do I think it best. It's just an example of the unusual. If you figure it out, it will help you understand Charles Moore's BASIC compiler (Vol. III, No. 6), which is even trickier. Good luck!

Marc Perkel
Perkel Software Systems
1636 N. Sherman
Springfield, MO 65803

### Searching for Graves

Dear FIG,

I read with interest the letter from Nick Francesco in *Forth Dimensions* (Vol. IV, No. 6). I share his feeling regarding the use of standard DOS files. I currently am using MicroMotion's Forth, which certainly is fine as far as the normal Forth operating systems go.

Mr. Francesco mentioned that William Graves' Forth II for the Apple II uses Apple's DOS. Looking through your section on system vendors, I could not find anything that looked like a potential Graves source. If you could supply me with more information regarding Graves Forth II, I would appreciate it. Or, if you do not have the information at hand, perhaps you could forward this letter to Mr. Francesco.

Thank you very much.
Sincerely,

James W. Patton
737 W. Davies Way
Littleton, CO 80120

*(Continued)*

```
SCR # B
   ( CASE EXAMPLE DEMONSTRATING FANCY BRANCHING, DOES>, AND
     THE POWER OF THE FORTH COMPILER )

   : SET-RETURN R> DUP @ >R 2+ >R ;
   : IFJUMP     OVER = IF DROP R> @ >R ELSE R> 2+ >R THEN ;
   : CASE       CREATE ] COMPILE SET-RETURN HERE 0 , DOES> >R ;
   : EXECUTES   COMPILE IFJUMP FIND 2+ , ; IMMEDIATE
   : ENDCASE    COMPILE DROP HERE SWAP !
                COMPILE EXIT [COMPILE] [ ; IMMEDIATE

   ( ALL WORDS ARE 79-STANDARD. FIND RETURNS THE CFA OF THE NEXT
     WORD IN THE INPUT STREAM. )

SCR # C
   ( CASE EXAMPLE )
   : PRINT-1 ." ONE " ;
   : PRINT-2 ." TWO " ;
   : PRINT-3 ." THREE " ;

  CASE TEST
         1 EXECUTES PRINT-1
         2 EXECUTES PRINT-2
         3 EXECUTES PRINT-3
       ENDCASE

   ( 1 TEST ONE OK
     2 TEST TWO OK
     3 TEST THREE OK
     4 TEST OK )
```

**CASE Statement**

## Forth Family Foiled

Dear Editor,

First, I heartily endorse your screen CRC words (*Forth Dimensions*, Vol. IV, No. 3).

Second, I am looking for a nice Forth for Apple III CP/M. I am aware of several systems for Apple II, but these require emulation mode, and this hampers operations significantly. The hard disk cannot be accessed with them and time is wasted loading the emulator.

Yesterday I told my two brothers-in-law, who already own Apple IIIs with CP/M cards, "No problem, there are lots of good Forths out there." Today I spent an hour on the phone calling Forth vendors and struck out. Is anyone out there catering to deluxe Apple III owners? They want a complete Forth system with full source code, some meta compiler, a screen editor, maybe strings and floating point, and perhaps some brief customizing documentation.

Love Forth,

Gary Nemeth
2727 Hampton Rd.
Rocky River, OH 44116

## ENCLOSE Encounters of the Second Kind

Dear Editor:

Reference *ENCLOSE Encounters*, in the Technotes section of Vol. V, No. 1. A line of code near the bottom of page thirty-four was omitted. It should have read,

```
HEX FIRST 2+ 400 BLANKS
```
(blanks a block buffer)
```
FIRST 2+ BL ENCLOSE
```

Thank you again for publishing the note.

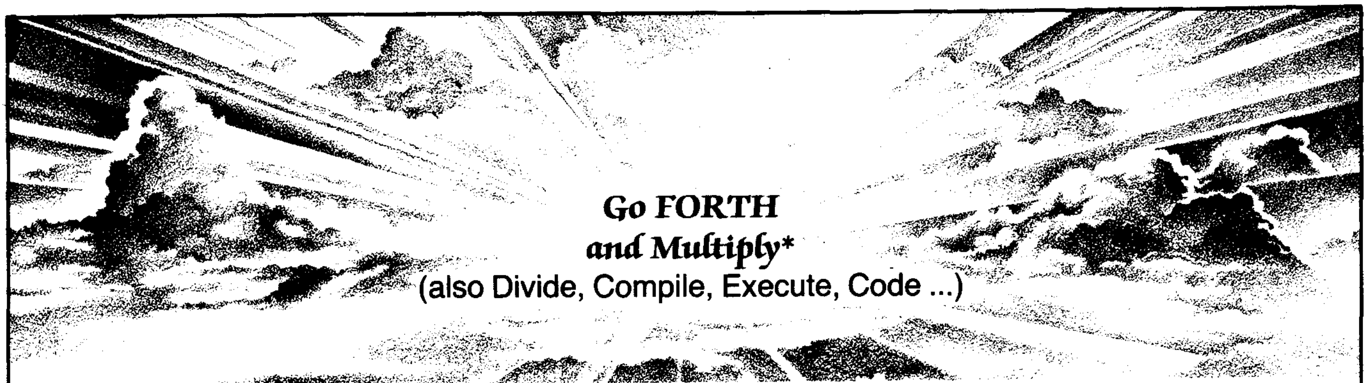
Sincerely yours,

Nicholas L. Pappas, Ph.D.
1201 Bryant St.
Palo Alto, CA 94301

*(Continued from page 26)*

techniques for compiling, and may occur when ] is compiled into a definition. This subject will certainly be a topic of discussion for the next meeting of the Forth Standards Team. Again, the definition of ] has not changed, and therefore should not be a reason to reject the new standard.

By adopting the new standard, the Forth Standards Team accepts Forth-83 as the current official FST standard, superceding all prior standards. It is important that we have a standard for writing transportable application code, as a basis for writing books and documents, and for teaching and communication.



# Meta Compiling III

*Henry Laxen*
*Berkeley, California*

Last time we talked about how to implement **CODE** words in the meta compiler, and saw how such words must operate in order to make meta : definitions work. We also saw how to define a symbol table for the definitions that are created during meta compilation by using the existing vocabulary structure. We also looked at how to create headers in the target address space. If any of these concepts are unfamiliar to you, I suggest you reread the previous two articles in this series, which discuss them in detail.

I would now like to talk about a few of the subtle issue that come up during meta compilation that must be handled by some means or another. Some of the subtle issues are how to handle forward references, and how immediate words such as ." are handled. Other similar issues arise, but we must leave some questions unanswered so that the reader can experience the joy of discovery.

The issue of forward references during meta compiling has, for some unknown reason, become almost a religious issue. The regular Forth interpreter treats forward references as an error condition, which has its pros and cons. Fortunately, it is almost always possible to write your Forth application in such a way that you can avoid forward references, hence one branch of the religion considers the problem solved, namely, don't use forward references. Unfortunately, in the meta compiling process, forward references are unavoidable, and we must develop a technique to handle them. Before I discuss a few solutions, I would like to present my view of the forward reference issue. The use of forward references is not sinful, immoral, illegal, or fattening. It should be discouraged but not banned. The problem that arises with forward references is that you can get yourself into big trouble. It destroys the bottom-up nature of Forth, and can cause you to retest previously working words because they make use of a forward reference which has changed. It also decreases the usefulness of program listings, if you are never sure of which way to turn the pages when you encounter an unfamiliar word.

Forward references also complicate the compiler, since it now must handle another class of objects (other than previously defined words and numbers). Most threatening, however, is that if forward references are abused, you can wind up with totally undecipherable spaghetti code. Just look at almost any Fortran program larger than 100 lines and written by a physicist, and you will see what I mean. The case for forward references is that sometimes you must have them. For example, if you are using recursion, and word A calls word B which calls word A, I am afraid a forward reference is somewhat unavoidable; if recursion is the natural solution to your problem, it would be silly not to use it. Also, error conditions are often more easily handled if forward references are allowed. You will often want a fatal error, which could occur at a relatively low level in your program, to call, say, the main menu routine, which obviously occurs at a very high level. That is the case in Forth, where **ABORT**, which is used at a very low level, calls the Forth interpreter, which is defined at a very high level.

Enough religion, let's take a look at some techniques for handling forward references during meta compilation. The simplest method to implement (and the hardest to control) is to simply use a place-holding word, and then patch it later when the resolving forward reference word is defined. Normally this word is called **GAP(** and it behaves like a comment, skipping the following text until the next ) and simply compiling a zero into the target image. The intervening text is usually the name of the word which will later be patched into this location. The problem with this approach is that you have to index some number of bytes, depending on the location of the gap, into the word that was defined, and patch it with another value. This approach is very inflexible and error prone, since if you ever change the definition in which the gap occurs, you must also change the place where it gets patched in a corresponding manner. There is no intelligence required, just conscientious effort, something humans are not well equipped for.

Another approach is to explicitly declare a forward reference before it takes place, and then resolve it somehow later when its target address is known. This is the Pascal approach, and is a pretty good compromise. At least you no longer have to count bytes into a word and hot patch it later. You can simply name the forward referenced word and define a mechanism that resolves it. This approach also allows you to have multiple forward references by linking them into a chain, and resolving the entire chain once the target address is known.

Finally, the last approach I will mention is that of handling forward references on the fly. I do not mean to imply that there are only three ways of doing this; there are many more, but three is enough for now. In order to handle forward references on the fly, we must modify the meta compiler's compiler. Instead of issuing an error message when an undefined word turns out not to be a number, we must define the word in question and remember the fact that it is a forward reference. Basically, all this entails is to change the compile loop to decide upon one of three cases instead of only two. Case one is that the word to be compiled already exists, in which case we simply compile it by executing it and letting it compile itself. Case two is that the word is a number in the current base, in which case we compile the code field for literal, followed by the value of the literal. Case three is that the word to be compiled is not already defined and is not a number, hence it must be a forward reference. In this case we must create an entry for it in the symbol table of forward references, compile a gap in the word currently being defined, and set up the run time of the forward reference to either link itself into a chain if it is not already resolved, or to compile itself if it is already resolved. Thus, forward references become basically transparent, except that they must be resolved somehow. This resolution can either be automatic as the word is actually defined, or explicit, requiring you to issue commands that will cause the resolution. Personally, I prefer the explicit method, since I am afraid of things happening behind my back, and it slightly discourages the use of forward references, which deep in my heart I know is right.

Enough about forward references, let's talk for a moment about immediate words. Immediate words present a special problem since they must be executed at compile time. They may do arbitrarily crazy things, and must do them in the target environment. For example, **[']** must look up the next word in the input stream and compile its code field as literal. Another example is **."** which must scan the input until another **"** is encountered, and then compile the runtime address for **(.")** which may not even be known yet, followed by the count-delimited string that was scanned. The usual mechanism used to implement immediate words is through a new defining word called **T:** which behaves just like Forth's : except that the definition it creates is placed in the target vocabulary, or symbol table. As you recall from last time, the main compiling loop looks up words in the symbol table and executes them. Words that are defined by **CODE** and : are placed in this symbol table, and when executed compile themselves. By using **T:** we can place words into this symbol table that do things other than compile themselves. For example **."** would have to first compile the run time for **."** namely **(."**, and then get the string and compile it into the target image. This is totally different behaviour from, say, the meta version of **DUP**, which simply compiles a pointer to the code field of **DUP** when it is executed. Thus, for each immediate word that passes through the meta compiling process, we must define a special case compiling word that "does the right thing" in the meta context.

Now I must apologize for not providing any code this time around. The problem is that all of the issues I discussed above are implemented in a very system-dependent manner; hence I would have to make a lot of assumptions about exactly how vocabularies work and how different system details operate. Rather than do that and provide code that would not run on any existing systems, I decided not to provide any code, but simply to discuss some of the remaining concepts involved in meta compiling. The best way to really learn about meta compilers is to write one. Hopefully, I have provided you with enough ammunition to attempt such an undertaking. Let me tell you that if you do, you will raise your level of Forth consciousness many levels, and I think it is an exercise well worth the effort.

Next time I will talk about multitasking, an issue many have heard about but few have seen. We will implement a very simple (and slow) high-level multi-tasker and discover its principles of operation. Until then, good luck and may Forth be with you!

 *The author is Chief Software Engineer for Universal Research, 150 North Hill Drive #10, Brisbane, CA 94005, specializing in the development of portable computers.*

## FORTH Vendors *(Continued from page 39)*

InnoSys
2150 Shattuck Ave.
Berkeley, CA 94704
415/843-8114

**Consultation & Training Only**
See System Vendor Chart
for others

Bartholomew, Alan
2210 Wilshire Blvd. #289
Santa Monica, CA 90403
213/394-0796

Boulton, Dave
581 Oakridge Dr.
Redwood City, CA 94062

Brodie, Leo
9720 Baden Ave.
Chatsworth, CA 91311
213/998-8302

Eastgate Systems Inc.
P.O. Box 1307
Cambridge, MA 02238

Girton, George
1753 Franklin
Santa Monica, CA 90404
213/829-1074

Go FORTH
504 Lakemead Way
Redwood City, CA 94062
415/366-6124

Harris, Kim R.
Forthright Enterprises
P.O. Box 50911
Palo Alto, CA 94303
415/858-0933

Intersystems Management
Computer Consultancy
Story Hill Rd. RFD3
Dunbarton, NH 03045
603/774-7762

Laxen, Henry H.
1259 Cornell Ave.
Berkeley, CA 94706
415/525-8582

McIntosh, Norman
2908 California Ave., #3
San Francisco, CA 94115
415/563-1246

Metalogic Corp.
4325 Miraleste Dr.
Rancho Palos Verdes, CA 90274
213/519-7013

Peschke, Manfred
Intersystems Mgmt. & Consult.
Story Hill Rd. RFD 3
Dunbarton NH 03045
603/774-7762

Petri, Martin B.
Computer Consultants
16005 Sherman Way
Suite 104
Van Nuys, CA 91406
213/908-0160

Redding Co.
P.O. Box 498
Georgetown, CT 06829
203/938-9381

Schleisiek, Klaus
Eppendorfer Landstr. 16
D 2000 Hamburg 20
West Germany
(040)480 8154

Schrenk, Dr. Walter
Postfach 904
7500 Karlstruhe-41
West Germany

Software Engineering
6308 Troost Ave. #210
Kansas City, MO 64131
816/363-1024

Softweaver
P.O. Box 7200
Santa Cruz, CA 95061
408/425-8700

Technology Management, Inc.
1520 S. Lyon St.
Santa Ana, CA 92705
714/835-9512

Timin, Mitchel
3050 Rue d'Orlean #307
San Diego, CA 92110
619/222-4185

RAM 22
CPU 68K
SYSTEM 8/16
CompuPro
CompuPro products are compatible with all S-100/IEEE 696 hardware. For performance, quality and reliability, contact your nearest Full Service CompuPro System Center today; call (415) 786-0909 extension 206 for location.
CompuPro
CompuPro, A GODBOUT COMPANY
3506 Breakwater Ct., Hayward, CA 94545

# FIG Chapter News

*John D. Hall*
*Oakland, California*

We now have forty-four chapters! The six new chapters are:

Rockwell FIG Chapter
Hoffman Estates, Illinois

Wichita FIG Chapter (FIGPAC)
Wichita, Kansas

Kansas City FIG Chapter
Kansas City, Missouri

Colombia FIG Chapter
Bogota, Colombia

Forth Interest Group — U.K.
London, England

Taiwan FIG Chapter
Taipei, Taiwan

**Orange County Chapter**

At the June 22 meeting, Dr. David E. Winkel of the University of Wyoming drew a full house. He spoke primarily about computer processor development and bit-slice design. He also introduced a bit-slice development system which he developed. There was a short discussion about an HP-75 Forth which one member is developing.

Wil Baden went through the fundamentals of conditionals at the July 6 meeting. Near the end of the meeting, Wil introduced non-compiling conditionals which allow one to occasionally compile. Also, Greg Stevenson developed a compiling buffer approach for conditionals. Bob Waters demonstrated Forth on a Timex-Sinclair, as advertised in Forth Dimensions. The Forth was direct compiling, multi-tasking, and had windowing capability.

On July 27, Wil Baden presented the **ONLY ALSO** vocabulary concepts of William F. Ragsdale. Many were already using the concept and were pleased with its performance. Wil added some

財團法人 資訊工業策進會贊助

FORTH

系統研習會

為發展最新電腦資訊技術，推廣最具威力FORTH語言。本會特邀請旅美學人，美國洛克布德飛機公司非破壞性檢驗主任丁陳漢蓀博士專程返國（七月二十日抵台，二十六日返美）主講最新電腦專題，機會難得，歡迎儘早報名參加，俾便準備講義及安排講習日期內午餐。

時間：七月廿一日至七月二十五日
每天　上午9:00～12:00（專題講座）
下午1:30～3:30（研討會）

報名地點：東方電腦中心（台北市重慶南路一段121號東方出版社三樓）
電話：3311316，3814907～08

研討會地址：資訊工業策進會研討室
台北市南京東路二段116號亞信大樓12樓

專題：
- Poly-FORTH及1983年FORTH新標準介紹
- Multitasking & multiprogramming
- Target compilation
- Case study（Control）
- Case study（Database）

費用：每人新台幣伍仟元（包括午餐及講義）

主辦　世界FORTH學會(FIG)台灣分會
（中華民國FORTH語言學會）

協辦　福世資訊科技公司
東方電腦中心
台北市重慶南路1段121號3樓
（衡陽路口東方出版社三樓）

**News from the FIG Chapter of Taiwan**

words that increased its performance. Zane Thomas is implementing a modem system for the Orange County Chapter to transfer screens. Bob Snook presented a short discussion on alternatives to **CASE** statements.

William Vock, who was visiting Greg Stevenson and collaborating with him on the development of software for the Epson QX-10 computer, presented a graphics package on August 3. Its performance was impressive. Mr. Vock is a graphics expert and knows the subject very well: he had done it with 8K of Forth.

**Rochester Chapter**

The Rochester (New York) FIG Chapter had its second meeting on June 25 at the University of Rochester. The group looked at the new 83-Standard draft, and Larry Forsley repeated the talk Bob Smith gave at the Rochester Conference on the specific differences between Forth-79 and 83-Standard. There was much discussion of the implications for ROMmed systems of the new mono-addressing rules. Specifical, does **BODY>** reference an address in RAM or ROM? It was suggested that copies of Bob's slides accompany distributed copies of the new standard.

**Nova Scotia Chapter**

The Nova Scotia Chapter held their first meeting on June 29. The group took a survey at this meeting and found (as in most chapters) that the members had a wide range of experience with Forth. Some people had no Forth experience at all, while others had written meta-compilers. They decided, for purposes of code exchange, that they would use 79-Standard. People were "volunteered" (a common Forth practice) to look into transferring screens to dissimilar computers and into collecting a list of all Forth information owned by various members in order to generate a master list. Graphics standards were also discussed and, as with any standards discussion, grew heated. As the bell rang, each side went to their respective corners. We will hear more about this in the future!

**Dayton Chapter**

At the June 25 meeting, Dr. Leonard Spialter gave a slide show of the Rochester conference. On July 12, the group decided to have a Forth tutorial and Dr. Spialter presented a flow chart for a "Day of the Week" program. The group spent the rest of the meeting programming it in Forth. Gary Granger gave a talk about Forth to the Columbus Ohio Heath Users Group on July 11, and a Forth talk and demo to the Timex-Sinclair Users group on July 19.

**Kansas City Chapter**

The first two meetings of the Kansas City Chapter had twenty to twenty-five attendees. The first meeting was spent getting acquainted and discovering who was doing what with Forth. At the second meeting, the group generated a list of topics of special interest for groups or programs. The highest items on the list were: program organization/coding style, graphics, data-base applications, and target compilers.

Support your local chapter!

*John D. Hall is the Chapter Coordinator for the Forth Interest Group and is a consulting programmer.*

## Chapters in Formation

Here are more of the new chapters that are forming. If you live in any of these areas, contact one of these people and offer your support in forming a FIG chapter.

Contact:

Michael Perry
1446 Stannage Ave.
Berkeley, CA 94702

Dick Turpin
3109 Breton Ave.
Davis, CA 95616

Samuel J. Cook
115 N. Washington Ave.
Batavia, IL 60510

Dr. Edward Newberger
2739 Elmwood Ave., Apt. 3
Kenmore, NY 14217

David Whitely
1163 West 550 North
Clearfield, UT 84015

Arnold Pinchuk
2130 Menasha Ave.
Manitowoc, WI 54220

T. William Rudolph
FIG-GRAPH East
592 Plymouth St.
Halifax, MA 02338

Tony Van Muyden
P.O. Box 7396
Edmonton, Alberta
T5E 6C8 Canada

Jack Hung
Comx World Operations
15/F Wo Kee Hong Bldg.
585-609 Castle Peak Rd.
Kwai Chung, N.T.
Hong Kong

Ravizza Donato
Sonnenbergstr. 34A
Uster 8610
Switzerland

Greg Stevenson
8002 Poinsettia Place
Buena Park, CA 90620

Glen Bowie
25746 North Player Dr., #Q-1
Valencia, CA 91355

Marc Perkel
Perkel Software Systems
1452 N. Clay
Springfield, MO 65802
(417) 862-9830

H. Marcus Bacon
704-1H E.I. DuPont
Savannah River Plant
Aiken, SC 29808

Richard Bloch
Eastern VA Center for MH Studies
Drawer A
Williamsburg, VA 23187

Wes Thomas
Jupiter Ace SIG
Frank Barth, Inc.
500-5th Ave.
New York, NY 10110

Scott Miles
Robotics
Christensen Diamond Products
2532 South 3270 West
Salt Lake City, UT 84119

Erick Ostergaard
COMPEX
2 Gertsvej
2300 Copenhagen S.
Denmark

Marc (Tamir) Weiner
Moshav Neve Ilan
D.N. Harei Yehuda 90850
Israel

# New Product Announcements

Forth Dimensions *welcomes press releases and product announcements, as well as reader letters regarding product performance. Addresses of the distributors and manufacturers mentioned in this column may be found in the Vendors List section.*

The latest Forth news from Little Rock is that Hawg Wild Software offers the **XFORTH XCHANGE** to original users of the XFORTH Forth-79 product. Questions, ideas and implementations should be sent to that company, who says their newest service will be "free and unrestricted."

Atari owners will be interested in **Power Forth** (for the 800/800XL, and 1200XL) from Elcomp Publishing. It is an extended FIG-Forth with editor and I/O routines. The utilities package includes decompiler, sector copy, Atari filehandling, graphics and sound, joystick program and player missile. The $39.95 price also covers two game demos and a mailing list application. Floating point with trig is an added $29.95, and the beginners' subset "Learn Forth" (requires 32K for disk or 16K for tape version) costs $19.95.

"UNIX-like word processing in Forth" is the claim made for **Forth-ms**. The licensed source code runs on Apple II computers using Epson printers with Graphtrax Plus, but can be configured by the user for other printers. Print spooling allows "simultaneous" use of printer and keyboard, and Greek letters and other symbols are available by typing a command followed by the name of the desired letter or symbol. The price (in single quantity) is $200 from Innovatia Laboratories.

The TDS900 is a **single-board Forth computer** with on-board screen editor, compiler and debug facilities. It uses FIG-Forth and provides simple interface to serial and parallel devices. All the user needs is a power supply, CRT and $395. If more than 12K RAM and 8K ROM is needed, up to 160K is available in increments of 20K per extra board. The computer and RAM boards use CMOS throughout, in single-Eurocard format. Information is available from Stynetic Systems, Inc. in the U.S. and from Triangle Digital Services, Ltd. in the U.K.

**Forth classes** will demonstrate how Forth can be used as an algorithm development tool and as a total programming environment. Problem solving will be emphasized by instructor Leo Brodie, author of *Starting Forth*. Students will apply design and problem-solving techniques in the design and coding of actual problems. East coast classes are planned for November, and the Los Angeles area will be covered in January.

"Any desired data file format" can benefit from **INDEX+**. By using its B-Tree ISAM utilities, Forth programmers can create and maintain keyed indexes in order to perform searches randomly, or sequentially in either direction. The program supports BLOCK disk I/O and the CP/M and MS-DOS interface by Laboratory Microsystems. Retail orders (the price is $125) should be sent to Laboratory Microsystems, for whose Forth systems INDEX+ is written; others should contact Business Computing Press.

Sylmar Software now offers FIG-Forth for the **Otrona Attache**. The two-disk set costs $50 and includes a full-screen editor and various utilities. A Towers of Hanoi version demonstrates the Attache's direct cursor operations. The user should obtain the FIG-Forth Installation Manual, which provides definitions for the Forth words.

# Fig Chapters

## U.S.

**• ARIZONA**

**Phoenix Chapter**
Call Dennis L. Wilson
602/956-7678

**• CALIFORNIA**

**Los Angeles Chapter**
Monthly, 4th Sat., 11 a.m.
Allstate Savings
8800 So. Sepulveda Boulevard
Los Angeles
Call Phillip Wasson
213/649-1428

**Northern California Chapter**
Monthly, 4th Sat., 1 p.m.
FORML Workshop at 10 a.m.
Palo Alto area.
Contact FIG Hotline
415/962-8653

**Orange County Chapter**
Monthly, 4th Wed., 7 p.m.
Fullerton Savings
Talbert & Brookhurst
Fountain Valley
Monthly, 1st Wed., 7 p.m.
Mercury Savings
Beach Blvd. & Eddington
Huntington Beach
Call Noshir Jesung
714/842-3032

**San Diego Chapter**
Weekly, Thurs., 12 noon.
Call Guy Kelly
619/268-3100 ext. 4784

**• COLORADO**

**Denver Chapter**
Monthly, 1st Mon., 7 p.m.
Call Steven Sarns
303/477-5955

**• ILLINOIS**

**Rockwell Chicago Chapter**
Call Gerard Kusiolek
312/885-8092

**• KANSAS**

**Wichita Chapter (FIGPAC)**
Monthly, 3rd Wed., 7 p.m.
Wilber E. Walker Co.
532 S. Market
Wichita, KS
Call Arne Flones
316/267-8852

**• MASSACHUSETTS**

**Boston Chapter**
Monthly, 1st Wed., 5 p.m.
Mitre Corp. Cafeteria
Bedford, MA
Call Bob Demrow
617/688-5661 after 7 p.m.

**• MICHIGAN**

**Detroit Chapter**
Call Dean Vieau
313/493-5105

**• MINNESOTA**

**MNFIG Chapter**
Monthly, 1st Mon.
1156 Lincoln Avenue
St. Paul, MN
Call Fred Olson
612/588-9532

**• MISSOURI**

**Kansas City Chapter**
Call Terry Rayburn
816/363-1024

**St. Louis Chapter**
Monthly, 3rd Tue., 7 p.m.
Thornhill Branch of
St. Louis County Library
Call David Douda
314/867-4482

**• NEVADA**

**Southern Nevada Chapter**
Suite 900
101 Convention Center Drive
Las Vegas, NV
Call Gerald Hasty
702/453-3544

**• NEW JERSEY**

**New Jersey Chapter**
Call George Lyons
201/451-2905 eves.

**• NEW YORK**

**New York Chapter**
Monthly, 2nd Wed., 8 p.m.
Queens College
Call Tom Jung
212/432-1414 ext. 157 days
212/261-3213 eves.

**Rochester Chapter**
Monthly, 4th Sat., 2 p.m.
Hutchison Hall
Univ. of Rochester
Call Thea Martin
716/235-0168

**Syracuse Chapter**
Call C. Richard Corner
315/456-7436

**• OHIO**

**Athens Chapter**
Call Isreal Urieli
614/594-3731

**Dayton Chapter**
Twice monthly, 2nd Tues &
4th Wed., 6:30 p.m.
CFC, 11 W. Monument Ave.
Suite 612
Dayton, OH
Call Gary M. Granger
513/849-1483

**• OKLAHOMA**

**Tulsa Chapter**
Monthly, 3rd Tues., 7:30 p.m.
The Computer Store
4343 South Peoria
Tulsa, OK
Call Art Gorski
918/743-0113

**• OREGON**

**Greater Oregon Chapter**
Monthly, 2nd Sat., 1 p.m.
Computer & Things
3460 SW 185th, Aloha
Call Timothy Huang
503/289-9135

**• TEXAS**

**Dallas/Ft. Worth Metroplex Chapter**
Monthly, 4th Thurs., 7 p.m.
Software Automation, Inc.
14333 Porton, Dallas
Call Marvin Elder
214/392-2802 or
Bill Drissel
214/264-9680

**San Antonio Chapter**
T.L. Schneider
8546 Broadway, Suite 207
San Antonio, TX 78217

**• VERMONT**

**Vermont Fig Chapter**
Monthly, 4th Thurs., 7:30 p.m.
The Isley Library, 3rd fl.
3rd Floor Meeting Room
Middleburynes, VT
Call Hal Clark
802/877-2911 days
802/452-4442 eves

**• VIRGINIA**

**Potomac Chapter**
Monthly, 1st Tues., 7 p.m.
Lee Center
Lee Highway at Lexington St.
Arlington, VA
Call Joel Shprentz
703/437-9218 eves.

## FOREIGN

**• AUSTRALIA**

**Australia Fig Chapter**
Contact: Ritchie Laird
25 Gibsons Road
Sale, Victoria 3850
051/44-3445

**FIG Australia Chapter**
Contact: Lance Collins
65 Martin Road
Glen Iris, Victoria 3146
03/29-2600

**Sydney Chapter**
Monthly, 2nd Fri., 7 p.m.
Morven Brown Bldg., Rm LG16
Univ. of New South Wales
Sydney
Contact: Peter Tregeagle
10 Binda Rd., Yowie Bay
02/524-7490

**• BELGIUM**

**Belgium Chapter**
Contact: Luk Van Loock
Lariksdreff 20
B2120 Schoten
03/658-6343

**• CANADA**

**Nova Scotia Chapter**
Contact: Howard Harawitz
P.O. Box 688
Wolfville, Nova Scotia B0P 1X0
902/542-7812

**Southern Ontario Chapter**
Monthly, 1st Sat., 2 p.m.
General Sciences Bldg, Rm 312
McMaster University
Contact: Dr. N. Solntseff
Unit for Computer Science
McMaster University
Hamilton, Ontario L8S 4K1
416/525-9140 ext. 2065

**Quebec Chapter**
Call Gilles Paillard
418/871-1960 or
418/643-2561

**• COLOMBIA**

**Colombia Chapter**
Contact: Luis Javier Parra B.
Aptdo. Aereo 100394
Bogota
214-0345

**• ENGLAND**

**Forth Interest Group -- U.K.**
Monthly, 1st Thurs.,
7 p.m., Rm. 408
Polytechnic of South Bank
Borough Rd., London
Contact: Keith Goldie-Morrison
15 St. Albans Mansion
Kensington Court Place
London W8 5QH

**• ITALY**

**FIG Italia**
Contact: Marco Tausel
Via Gerolamo Forni 48
20161 Milano
02/645-8688

**• NETHERLANDS**

**HCC-FORTH Interest Group Chapter**
F.J. Meijer
Digicos
Aart V.D. Neerweg 31
Ouderkerk A.D.
Amstel, The Netherlands

**• SOUTH AFRICA**

Contact: Edward Murray
Forthwith Computers
P.O. Box 27175
Sunnyside, Pretoria 0132

**• SWITZERLAND**

Contact: Max Hugelshofer
ERNI & Co. Elektro-Industrie
Stationsstrasse
8306 Bruttisellen
01/833-3333

**• TAIWAN**

**Taiwan Chapter**
Contact: J.N. Tsou
Forth Information Technology
P.O. Box 53-200
Taipei
02/331-1316

**• WEST GERMANY**

**West German Chapter**
Klaus Schleisiek
FIG Deutschland
Postfach 202264
D 2000 Hamburg 20
West Germany

## SPECIAL GROUPS

**Apple Corps FORTH Users Chapter**
Twice Monthly, 1st &
3rd Tues., 7:30 pm
1515 Sloat Boulevard, #2
San Francisco, CA
Call Robert Dudley Ackerman
415/626-6295

**Baton Rouge Atari Chapter**
Call Chris Zielewski
504/292-1910

**FIGGRAPH**
Call Howard Pearlmutter
408/425-8700

**MMSFORTH Users Groups**
Monthly, 3rd Wed., 7 p.m.
Cochituate, MA
Dick Miller
617/653-6136
(25 groups world-wide)

# FORTH System Vendors

## (by Category)

**(Codes refer to alphabetical listing e.g., A1 signifies AB Computers, etc.)**

### Processors

| | |
|---|---|
| 1802 | C2, C3, F3, F6, L3 |
| 6502 (AIM, KIM, SYM) | R1, R2, S2 |
| 6800 | C3, F3, F5, K1, L3, M6, T1 |
| 6801 | P4 |
| 6809 | C3, F3, L3, M6, T1 |
| 68000 | C3, C5, D1, E1, K1 |
| 68008 | P4 |
| 8080/85 | A5, C2, C3, F4, I5, L1, L3, M3, M6, R1, T3 |
| Z80/89 | A3, A5, C3, F4, I3, L1, M2, M3, M5, N1, T3 |
| Z80000 | I3 |
| 8086/88 | C3, F2, F3, L1, L3, M6 |
| 9900 | E4, L3 |

### Operating Systems

| | |
|---|---|
| CP/M | A3, A5, C3, F3, I3, L3, M1, M2, M6, T3 |
| CP/M-86 | C3 |

### Computers

| | |
|---|---|
| Alpha Micro | P3, S4 |
| Apple | A4, E1, E2, F4, I2, I5, J1, L4, M2, M6, M8, O2, O3 |
| Atari | E1, E2, M6, P2, Q1, V1 |
| Compaq | M5 |
| Cromemco | A5, M2, M6 |
| DEC PDP/LSI-11 | C3, F3, L2, S4 |
| Heath-89 | M2, M6, M7 |
| Hewlett-Packard 85 | |
| Hewlett-Packard 9826/36 | C5 |
| IBM PC | A8, C3, F3, L1, M5, M6, Q2, S8, W2 |
| IBM Other | L3, W1 |
| Kaypro II/Xerox 820 | M2 |
| Micropolis | A2, M2, S3 |
| North Star | I5, M2, P1, S11 |
| Nova | C6 |
| Ohio Scientific | A6, B1, C4, O1, S7, T2 |
| Osborne | M2 |
| Pet SWTPC | A1, A6, B1, C3, O1, S7, T2, T5 |
| Poly Morphic Systems | A7 |
| TRS-80 I, II, and/or III | I5, M2, M5, M6, S5, S6, S9 |
| TRS-80 Color | A3, A8, F5, M4, T1 |
| Vector Graphics | M2 |

### Other Products/Services

| | |
|---|---|
| Applications | P4 |
| Boards, Machine | F3, M3, P4, R2, S10 |
| Consultation | C3, C5, N1, P4, T3, W1 |
| Cross Compilers | C3, F3, I3, M6, N1, P4 |
| Products, Various | A5, B2, C3, C7, F3, I4, I5, S8, S12, W2 |
| Training | C3, F3, I3, P4, W1 |

# FORTH Vendors (Alphabetical)

The following vendors offer FORTH systems, applications, or consultation. FIG makes no judgment on any product, and takes no responsibility for the accuracy of this list. We encourage readers to keep us informed on availability of the products and services listed. Vendors may send additions and corrections to the Editor, and must include a copy of sales literature or advertising.

**FORTH Systems**

**A**

1. AB Computers
 252 Bethlehem Pike
 Colmar, PA 18915
 215/822-7727

2. Acropolis
 17453 Via Valencia
 San Lorenzo, CA 94580
 415/276-6050

4. Applied Analytics Inc.
 8910 Brookridge Dr., #300
 Upper Marlboro, MD 20870

5. Aristotelian Logicians
 2631 E. Pinchot Ave.
 Phoenix, AZ 85016

7. Abstract Systems, etc.
 RFD Lower Prospect Hill
 Chester, MA 01011

8. Armadillo Int'l Software
 P.O. Box 7661
 Austin, TX 78712
 512/459-7325

**B**

1. Blue Sky Products
 729 E. Willow
 Signal Hill, CA 90806

2. Business Computing Press
 2210 Wilshire Blvd.
 Suite 289
 Santa Monica, CA 90403
 213/394-0796

**C**

1. Capstone Computing, Inc.
 5640 Southwyck Blvd., #2E
 Toledo, OH 43614
 419/866-5503

2. Chrapkiewicz, Thomas
 16175 Stricker
 East Detroit, MI 48021

3. CMOSOFT
 P.O. Box 44037
 Sylmar, CA 91342

4. COMSOL, Ltd.
 Treway House
 Hanworth Lane
 Chertsey, Surrey
 England KT16 9LA

5. Consumer Computers
 8907 La Mesa Blvd.
 La Mesa, CA 92041
 714/698-8088

6. Creative Solutions, Inc.
 4801 Randolph Rd.
 Rockville, MD 20852
 301/984-0262

7. Curry Associates
 P.O. Box 60324
 Palo Alto, CA 94306

**E**

1. Elcomp Publishing, Inc.
 53 Redrock Lane
 Pomona, CA 91766
 (714) 623-8314
 Telex 29 81 91

2. Elcomp-Hofacker
 Tegernseerstr. 18
 D-8150 Holzkirchen
 West Germany
 08024/7331
 Telex 52 69 73

3. Emperical Research Group
 P.O. Box 1176
 Milton, WA 98354
 206/631-4855

4. Engineering Logic
 1252 13th Ave.
 Sacramento, CA 95822

**F**

1. Fantasia Systems, Inc.
 1059 The Alameda
 Belmont, CA 94002
 415/593-5700

3. FORTH, Inc.
 2309 Pacific Coast Highway
 Hermosa Beach, CA 90254
 213/372-8493

4. FORTHWare
639 Crossridge Terrace
Orinda, CA 94563

5. Frank Hogg Laboratory
130 Midtown Plaza
Syracuse, NY 13210
315/474-7856

6. FSS
P.O. Box 8403
Austin, TX 78712
512/477-2207

**H**

1. HAWG WILD Software
P.O. Box 7668
Little Rock, AR 72217

**I**

1. IDPC Company
P.O. Box 11594
Philadelphia, PA 19116
215/676-3235

2. IUS (Cap'n Software)
281 Arlington Ave.
Berkeley, CA 94704
415/525-9452

3. Inner Access
517K Marine View
Belmont, CA 94002
415/591-8295

4. Innovatia Laboratories
5275 Crown St.
West Linn, OR 97068

5. Insoft
10175 S.W. Barbur Blvd.
Suite #202B
Portland, OR 97219
503/244-4181

6. Interactive Computer
Systems, Inc.
6403 Di Marco Rd.
Tampa, FL 33614

**J**

1. JPS Microsystems, Inc.
361 Steelcase Rd., W.
Markham, Ontario
Canada L3R 3V8
416/475-2383

**K**

1. Kukulies, Christoph
Ing. Buro Datentec
Heinrichsallee 35
Aachen, 5100
West Germany

**L**

1. Laboratory Microsystems
4147 Beethoven St.
Los Angeles, CA 90066
213/306-7412

2. Laboratory Software
Systems, Inc.
3634 Mandeville Canyon
Los Angeles, CA 90049
213/472-6995

3. Lynx
3301 Ocean Park, #301
Santa Monica, CA 90405
213/450-2466

4. Lyons, George
280 Henderson St.
Jersey City, NJ 07302
201/451-2905

**M**

1. M & B Design
820 Sweetbay Dr.
Sunnyvale, CA 94086

2. MicroMotion
12077 Wilshire Blvd., #506
Los Angeles, CA 90025
213/821-4340

3. Microsystems, Inc.
2500 E. Foothill Blvd., #102
Pasadena, CA 91107
213/577-1477

4. Micro Works, The
P.O. Box 1110
Del Mar, CA 92014
714/942-2400

5. Miller Microcomputer
61 Lake Shore Rd.
Natick, MA 01760
617/653-6136

6. Mountain View Press
P.O. Box 4656
Mountain View, CA 94040
415/961-4103

7. MCA
8 Newfield Ln.
Newtown, CT 06470

8. Metacrafts Ltd.
Beech Trees, 144 Crewe Rd.
Shavington, Crewe
England CW1 5AJ

**N**

1. Nautilus Systems
P.O. Box 1098
Santa Cruz, CA 95061
408/475-7461

**O**

1. OSI Software & Hardware
3336 Avondale Court
Windsor, Ontario
Canada N9E 1X6
519/969-2500

2. Offete Enterprises
1306 S "B" St.
San Mateo, CA 94402

3. On-Going Ideas
RD #1, Box 810
Starksboro, VT 05487
802/453-4442

**P**

1. Perkel Software Systems
1636 N. Sherman
Springfield, MO 65803

2. Pink Noise Studios
P.O. Box 785
Crockett, CA 94525
415/787-1534

3. Professional Mgmt. Services
724 Arastradero Rd., #109
Palo Alto, CA 94306
408/252-2218

4. Peopleware Systems Inc.
5190 West 76th St.
Minneapolis, MN 55435
612/831-0827

**Q**

1. Quality Software
6660 Reseda Blvd., #105
Reseda, CA 91335

2. Quest Research, Inc.
P.O. Box 2553
Huntsville, AL 35804
800/558-8088

**R**

2. Rockwell International
Microelectronics Devices
P.O. Box 3669
Anaheim, CA 92803
714/632-2862

**S**

1. Satellite Software Systems
288 West Center
Orem, UT 84057
801/224-8554

2. Saturn Software, Ltd.
P.O. Box 397
New Westminister, BC
Canada V3L 4Y7

3. Shaw Labs, Ltd.
P.O. Box 3471
Hayward, CA 94540
415/276-6050

4. Sierra Computer Co.
617 Mark NE
Albuquerque, NM 87123

5. Sirius Systems
7528 Oak Ridge Highway
Knoxville, TN 37921
615/693-6583

6. Software Federation
44 University Drive
Arlington Hts., IL 60004
312/259-1355

7. Software Works, The
1032 Elwell Ct., #210
Palo Alto, CA 94303
415/960-1800

8. Spectrum Data Systems
5667 Phelps Luck Dr.
Columbia, MD 21045
301/992-5635

9. Stearns, Hoyt Electronics
4131 E. Cannon Dr.
Phoenix, AZ 85028
602/996-1717

10. Stynetic Systems, Inc.
Flowerfield, Bldg. 1
St. James. NY 11780
516/862-7670

11. Supersoft Associates
P.O. Box 1628
Champaign, IL 61820
217/359-2112

12. Sylmar Software
P.O. Box 44037
Sylmar, CA 91342

**T**

1. Talbot Microsystems
1927 Curtis Ave.
Redondo Beach, CA 90278

2. Technical Products Co.
P.O. Box 12983
Gainsville, FL 32604
904/372-8439

3. Timin Engineering Co.
C/o Martian Technologies
8348 Center Dr. Suite F
La Mesa, CA 92041
619/464-2924

4. Transportable Software
P.O. Box 1049
Hightstown, NJ 08520
609/448-4175

**V**

1. Valpar International
3801 E. 34th St.
Tucson, AZ 85713
800/528-7070

**W**

1. Ward Systems Group
8013 Meadowview Dr.
Frederick, MD 21701

2. Worldwide Software
2555 Buena Vista Ave.
Berkeley, CA 94708
415/644-2850

**Z**

1. Zimmer, Tom
292 Falcato Dr.
Milpitas, CA 95035

**Boards & Machines Only**
See System Vendor Chart
for others

Controlex Corp.
16005 Sherman Way
Van Nuys, CA 91406
213/780-8877

Datricon
7911 NE 33rd Dr., #200
Portland, OR 97211
503/284-8277

Golden River Corp.
7315 Reddfield Ct.
Falls Church, CA 22043

Triangle Digital Services Ltd.
23 Campus Road
London E17 5PG
England

**Application Packages Only**
See System Vendor Chart
for others

Curry Associates
P.O. Box 11324
Palo Alto, CA 94306
415/322-1463

*(Continued on page 32)*

## FORTH INTEREST GROUP

## MAIL ORDER

| | USA | FOREIGN AIR |
|---|---|---|
| ☐ Membership in FORTH Interest Group and Volume V of FORTH DIMENSIONS | $15 | $27 |
| ☐ Back Volumes of FORTH DIMENSIONS. **Price per each.** ☐ I ☐ II ☐ III ☐ IV | $15 | $18 |
| ☐ fig-FORTH Installation Manual, containing the language model of fig-FORTH, a complete glossary, memory map and installation instructions | $15 | $18 |
| ☐ Assembly Language Source Listings of fig-FORTH for specific CPU's and machines. The above manual is required for installation. Check appropriate box(es). **Price per each.** | $15 | $18 |

☐ 1802 ☐ 6502 ☐ 6800 ☐ 6809 ☐ VAX ☐ Z80
☐ 8080 ☐ 8086/8088 ☐ 9900 ☐ APPLE II ☐ ECLIPSE
☐ PACE ☐ NOVA ☐ PDP-11 ☐ 68000 ☐ ALPHA MICRO

| | USA | FOREIGN AIR |
|---|---|---|
| ☐ "Starting FORTH, by Brodie. BEST book on FORTH. (Paperback) | $18 | $22 |
| ☐ "Starting FORTH" by Brodie. (Hard Cover) | $23 | $28 |

☐ PROCEEDINGS: FORML (FORTH Modification Conference)
- ☐ 1980, $25USA/$35Foreign
- ☐ 1981, Two Vol., $40USA/$55Foreign
- ☐ 1982, $25USA/$35Foreign

ROCHESTER FORTH Conference
- ☐ 1981, $25USA/$35Foreign
- ☐ 1982, $25USA/$35Foreign
- ☐ 1983, $25USA/$35Foreign — Total $________

☐ STANDARD: ☐ FORTH-79, ☐ FORTH-83. $15USA/$18Foreign EACH. — Total $________

| | USA | FOREIGN AIR |
|---|---|---|
| ☐ Kitt Peak Primer, by Stevens. An in-depth self-study book. | $25 | $35 |

☐ MAGAZINES ABOUT FORTH: ☐ BYTE Reprints 8/80-4/81
☐ Dr Dobb's Jrnl, ☐ 9/81, ☐ 9/82, ☐ 9/83
☐ Poplar Computing, 9/83 $3.50USA/$5Foreign EACH. — Total $________

| | USA | FOREIGN AIR |
|---|---|---|
| ☐ FIG T-shirts: ☐ Small ☐ Medium ☐ Large ☐ X-Large | $10 | $12 |
| ☐ Poster, BYTE Cover 8/80, 16"x22" | $ 3 | $ 5 |
| ☐ FORTH Programmer's Reference Card. If ordered separately, send a stamped, self addressed envelope. | Free | |

TOTAL $________

NAME________________________ MS/APT________
ORGANIZATION________________________ PHONE(   )________
ADDRESS________________________
CITY____________ STATE______ ZIP________ COUNTRY________
VISA#________________ MASTERCARD#________________
AMERICAN EXPRESS#________________ Card Expiration Date________
(Minimum of $15.00 on Charge Cards)

Make check or money order in US Funds on US Bank, payable to: FIG. All prices include postage. No purchase orders without check. California residents add sales tax. 10/83

**ORDER PHONE NUMBER: (415) 962-8653**

**FORTH INTEREST GROUP * PO BOX 1105 * SAN CARLOS, CA 94070**

**FORTH INTEREST GROUP**
P.O. Box 1105
San Carlos, CA 94070

BULK RATE
U.S. POSTAGE
PAID
Permit No. 261
Mt. View, CA

ROBERT SMITH
2300 ST. FRANCIS DR
PALO ALTO, CA 94303

Address Correction Requested