

**FORTH**

Volume 8, Number 4

November/December 1986  
\$4.00

**Dimensions**

**Simple  
File Query**

**A Forth Standard?**

**Dual-CFA Definitions**

**Batcher's Sort**

**Getting Started with F83**

**Windows for the TI 99/4A**

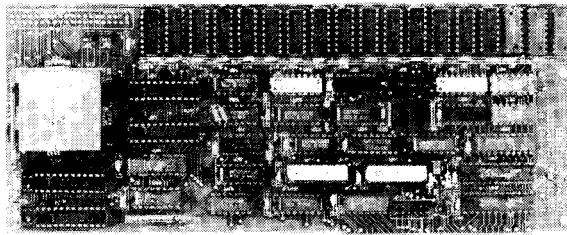
---

**A PC DROP-IN BOARD WITH:  
Novix Forth engine, 4 MIPS  
power, mini computer speed,  
parallel PC operation,  
1/2 Mbyte on board, multi-  
tasking capability, software  
included. \$1,495**

**YEAH, SURE.**

**It's finally here! The PC4000.**

Plugs into PC/XT or PC compatible. Comes with 4 Mhz clock. Upgrade to 5 Mhz by adding faster RAM and Clock. 16K of memory ported to PC bus for PC/PC4000 data transfer. Selectable 2K or 8K stacks or multiple stacks for multitasking.



PC4000

Runs the Sieve in Forth in .09 seconds—2170 times faster than the Sieve runs on the PC in PC-Basic. Includes SCForth software package for software development. RAM on board can be used to extend host memory space. C software available.

Contact:  
Software Composers  
Suite F  
210 California Avenue  
Palo Alto, CA 94306  
415-322-8763



---

SOFTWARE COMPOSERS

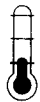
**Forth Dimensions**  
 Published by the  
**Forth Interest Group**  
 Volume VIII, Number 4  
 November/December 1986

Editor  
 Marlin Ouverson  
 Advertising Manager  
 Kent Safford  
 Production  
 Cynthia Lawson Berglund  
 Typesetting  
 LARC Computing

*Forth Dimensions* solicits editorial material, comments and letters. No responsibility is assumed for accuracy of submissions. 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 to the Forth Interest Group.

Subscription to *Forth Dimensions* is free with membership in the Forth Interest Group at \$30 per year (\$43 foreign air). For membership, change of address and to submit items for publication, the address is: Forth Interest Group, P.O. Box 8231, San Jose, California 95155. Administrative offices and advertising sales: 408-277-0668.

### Symbol Table



Simple; introductory tutorials and simple applications of Forth.



Intermediate; articles and code for more complex applications, and tutorials on generally difficult topics.



Advanced; requiring study and a thorough understanding of Forth.



Code and examples conform to Forth-83 standard.



Code and examples conform to Forth-79 standard.



Code and examples conform to fig-FORTH.



Deals with new proposals and modifications to standard Forth systems.

# FORTH

## Dimensions

### FEATURES

#### 13 Dual-CFA Definitions, Part Two by Mike Elola



The dual-CFA structure provides a new method for decomposing functions into smaller functions. Its value can be demonstrated in deferred and vectored definitions, and in definitions that dispatch multiple functions. This strategy can be the basis of a Forth programming philosophy aimed at compactness, brevity and programming ease.

#### 17 Simple File Query by Edward Petsche



This program allows the user to define and initialize a file, enter data, query on any combination of fields, delete records and change field values in records. It is based on data-base elements presented previously in *Forth Dimensions* and should work with most versions of Forth-83.

#### 28 A Forth Standard? by Glen B. Haydon

Forth does not differ from a natural language: it is evolving. And what is a standard language? Only after a word is used with a specific meaning for some time do dictionary editors accept it. This essay considers common use as a common-sense paradigm for Forth standards.

#### 34 Windows for the TI 99/4A by Blair MacDermid



This program plots algebraic functions in a choice of five windows on the display. It computes the coordinates of a plotted function, appropriately scaled to fit within the selected window. (Members of the Fort Wayne FIG Chapter implemented the ACM SIGGRAPH CORE Standard as a group project, from which this code was adapted later for publication.)

#### 37 Getting Started with F83 by Greg McCall



Sifting through F83's source shadow screens can be a bewildering first exposure to that system. This summary of the file words and file-editing facilities will ease your introduction. It explains how to open a second, read-only file and load screens from it without changing the **CURRENT** file.

#### 39 Batcher's Sort by John Konopka



An alternative to the sometimes quirky Quicksort was discovered by K.E. Batcher — slightly slower, but more robust and with consistent sorting times. If you'd rather not complicate your Quicksort code to handle special cases, Batcher's may be just the sort for you.

### DEPARTMENTS

- 5 Letters
- 12 Editorial: "Conventions"
- 27 Crossword
- 31 Advertisers Index
- 42 FIG Chapters

Visit the **MACH 2 Product Support RoundTable™** on **GENie™** !!

# MACH 2

## MULTI-TASKING FORTH 83 DEVELOPMENT SYSTEM

The **MACH 2 FORTH 83 Multi-tasking Development System** created by Palo Alto Shipping Company provides a fresh approach to FORTH programming and the FORTH language. The foundation of **MACH 2** is a subroutine threaded FORTH with automatic macro substitution. This state-of-the-art implementation of the FORTH language allows **MACH 2** to take full advantage of the powerful 680X0 microprocessors; therefore execution times of programs written in **MACH 2** are comparable to the execution times of programs written in the traditional compiled languages.

**MACH 2's** integrated programming environment consists of a standard (infix), Motorola-format assembler which supports local labels and forward references, a symbolic debugger/disassembler which allows multiple task debugging with single-stepping, breakpoints, and more. The Macintosh and Atari ST systems include a mouse-based, multi-window text editor and all systems support the use of text source files.

The **MACH 2** system is a professional development system designed to take the programmer through all phases of product development -- from initial design/prototyping to the creation of the final, stand-alone application.

### **MACH 2 FOR THE MACINTOSH™**

features full support of the Macintosh toolbox, support of the Macintosh speech drivers, printing, and floating point, easy I/O redirection and creates double-clickable, multi-segment Macintosh applications. Includes RMaker, and 500 pg manual.

**\$99.95**

### **MACH 2 FOR THE ATARI ST™**

features full GEM and TOS support, floating point, I/O redirection and creates double-clickable ST applications. Includes 300 page manual.

**\$99.95**

### **MACH 2 FOR THE OS-9 OPERATING SYSTEM™**

provides position-independent and re-entrant code execution, full support of all OS-9 system calls. Creates stand-alone OS-9 applications. Link FORTH to C and vice-versa. Includes 400 page manual.

**\$495.00**

### **MACH 2 FOR INDUSTRIAL BOARDS**

is 68020 compatible, provides 68881 Floating Point support, and produces position-independent, relocatable, ROM-able code with no meta-compilation or target compilation required. Includes system manual and porting manual.

**\$495.00**

## **PALO ALTO SHIPPING COMPANY**

P.O. Box 7430

Menlo Park, California 94026

Support: 415 / 854-7994 Sales: 800 / 44FORTH

VISA/MC accepted. CA residents include 6.5% sales tax.

Include shipping/handling with all orders: US \$5 S/H; Australia \$20 S/H; Canada \$7 S/H; Europe \$10 S/H.

RoundTable and GENie are registered trademarks of the General Electric Information Services Company.

## Fast SEARCH for F83

Dear FIG,

I am happy to finally contribute something to the Forth community. For all of the 8086/8088 F83 users out there, here is a **SEARCH** function completely written in low-level code that executes very quickly. Since the original F83 **SEARCH** function was threaded code it was tolerably slow, but a project I've been working on lately needed a quicker **SEARCH**, so I bit the bullet and did it. Here, the function is adapted to the Laxen & Perry system ... enjoy faster searching!

(In order to maintain the threaded code "purity" of the UTILITY.BLK file, this function should be placed in either the KERNEL.BLK or the CPU8086.BLK source files, and the existing **SEARCH** function in the UTILITY.BLK file should be commented out.)

I have been programming exclusively in Forth for the past three years and, having written both Z80- and 8086-based systems, I feel qualified to say that Forth offers the greatest man/machine interface yet devised in software. Although it is slightly more difficult to adapt to Forth's subtle programming philosophy, the rewards are quick in coming. I know of many things that *can* be done in Forth but which are *impossible* in other programming languages.

As a rather lazy person, I would like to commend all of the FIG community for their tireless efforts in promoting the very best programming language yet designed. And special thanks to Chuck Moore, Leo Brodie, Henry Laxen, Michael Perry, Marlin Ouverson, Bill Ragsdale and all of the other regular contributors to the progress of *Forth Dimensions*. I hope their example motivates more people to contribute.

Sincerely,

Bill Zimmerly  
St. Charles, Missouri

### Natural Word Usage

Dear Mr. Ouverson:

Ting's computation of static F83 word reference counts<sup>1</sup> is the first I have seen. The total number of words (11,063) is large enough to be interesting. I immediately plotted a graph with the words ordered by frequency of use. A log-log plot was the cleanest and had, for me, a surprising result: reference count was inversely proportional to frequency, i.e., the data closely fits a line of slope -1. I tried several other populations I had available<sup>2,3</sup>: one of spoken English and one of written English. The results were the same!

While browsing at the library one day, I came across a volume on Zipf's

law<sup>4</sup>. The explanation was at hand: this is a property of human behavior. Thus, Forth has some of the properties of natural languages.

I also investigated various coding techniques<sup>6</sup> to determine the amount of compaction that can be obtained taking advantage of the frequency-of-use statistics. The results are somewhat disappointing. For hardware implementations, a block encoding is probably all that can be justified.

Number of different tokens (words)	555
Total number of occurrences	11,063
Block code size	9.116 bits
Theoretical code size	7.051
Hoffman code size	7.084
4-8-12 repeated comma code	7.821
4-8-12 non-repeated comma code	7.735
8-16 repeated comma code	9.316
8-16 non-repeated comma code	9.316

In the repeated codes, the same token (word) can be coded in several sizes which, of course, lowers the coding efficiency. The relatively small number of words (compared to 2\*\*16) accounts for the poor performance for the 8-16 codes.

1. C.H. Ting. "F83 Word Usage Statistics." *Forth Dimensions* VII/4, pg14, November/December 1985.
2. H.F. Gaines. *Cryptanalysis*. Dover, 1956.

### Zimmerly's F83 SEARCH

```

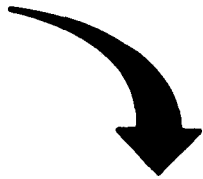
      86                                     223
0 \ String functions...                    WBZ 11-09-1985 \ String functions...      WBZ 11-09-1985
1
2 ASSEMBLER LABEL (FIND1)                  \ THE EXIT POINT CODE (FIND1) is the exit point for the SEARCH function that follows.
3     DX SI MOV BX DX MOV BP POP ZPUSH     At entry, the BX contains the offset address, and the
4                                           AX contains the TRUE or FALSE flag.
5 CODE SEARCH ( SADR SLEN BADR BLEN -- N F ) \ FIND SUBSTRING
6     CLD CX POP DI POP BX POP DX POP BP PUSH DX SI XCHG SEARCH is a very high speed (how could it be faster?) function
7     CS AX MOV AX ES MOV 0 [SI] AL MOV HERE BYTE REP SCAS 0= that scans a string trying to locate the given substring
8     IF CX PUSH SI PUSH DI PUSH DI DEC BX CX MOV ( get count) within it. The method used is to search for the first
9     BYTE REPZ CMPS 0= ( compare the strings for equality! ) character, and when found, compare the characters that
10    IF BX POP AX POP AX POP BX DEC -1 # AX MOV ( true flag!) follow it for a complete match. If both fail, the search
11    (FIND1) #) JMP THEN DI POP SI POP CX POP ELSE AX AX XOR for the first character continues from where it left off
12    (FIND1) #) JMP THEN #) JMP END-CODE until we've scanned the entire buffer.
13
14
15

```

# BRYTE FORTH

*for the*

# INTEL 8031 MICRO- CONTROLLER



## FEATURES

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

## COST

130 page manual — \$ 30.00  
8K EPROM with manual— \$100.00  
Postage paid in North America.  
Inquire for license or quantity pricing.

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

3. G.D.A. Brown. A frequency count of 190,000 words in the *London-Lund Corpus of English Conversation*. Behavior Research Methods, *Instruments & Computers*, 16 (6):502-532, 1984.
4. S.R. Ellis and R.J. Hitchcock. "The Emergence of Zipf's Law: Spontaneous Encoding Optimization by Users of a Command Language." *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-16(3):423, May 1986.
5. G.K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley Press, 1949.
6. R.W. Hamming. *Coding and Information Theory*. Prentice-Hall, 1980.

Sincerely,

James C. Brakefield, M.S.E.E.  
San Antonio, Texas

## Leaky Sieve

Mr. Ouverson:

In the process of optimizing the sieve benchmark, Terry Noyes has unwittingly rejected a superior algorithm and departed from the de facto benchmark standard. The sieves Mr. Noyes calls "corrupt" are not corrupt at all. They correctly count 1899 primes from 3 to 16383. The **FLAGS** array represents only odd integers, and only odd multiples of primes are "flicked." Fortunately, the Noyes version is easily modified to incorporate the better algorithm, and the resulting version finds the 1899 primes slightly faster than the unmodified version finds 1028.

Stephen Brault  
Chandler, Arizona

Mr. Ouverson:

I would like to retract my April letter to you (*Forth Dimensions* VIII/2) and live in shame for the rest of my life.

I had thought that the 0 - 8192 loop in all sieve benchmarks meant they were calculating the number of primes from zero to 8192. Not so. A few weeks after the letter was sent to you, someone pointed out that these sieves were actually finding primes in the range of zero to 16,000+ by looping through the 8192 *odd* numbers in that range.

Oh.

Fortunately, we use identical code to benchmark other Forth systems, so they also received the five percent speed improvement resulting from using the wrong sieve program. I've enclosed the proper Forth sieve with this letter.

Living and Learning,

Terry Noyes  
Palo Alto Shipping Company  
Menlo Park, California

## Seeing is Believing

Dear Marlin:

I enjoyed Michael Ham's "Making Numbers Pretty" (VII/5). I had just written a routine to calculate the necessary number for masking a given bit (or bits) and Michael's words **.BITS** and **16BITS** fit in perfectly, although I modified them slightly.

Referring to the enclosed listing, the words **BIT-MASK** and **2BIT-MASK** return to the console the number necessary to mask the desired bit, or bits, in the current base. The words **.BITS** and **16BITS** visually confirm the mask, making life a little easier for us doubters!

For example, if you want to mask bit five, then entering **5 BIT-MASK** (in base ten) will give:

```
32 In base 10  
HI: 0000 0000 0010 0000 :LO
```

while if bits four and six are to be masked, then **4 6 2BIT-MASK** (in base sixteen) will return:

```
50 In base 16  
HI: 0000 0000 0101 0000 :LO
```

### Noyes' Sieve

```
decimal
8192 constant size

variable flags size v allot

: primes ( - primes)      ( does the primes once)
  (flags size 01 fill)    ( initialize the array)
  0                         ( prime counter)
  size 0                    ( range/2 of numbers to do)
  DO
    flags i + c@ ( see if prime already)
    IF
      3 i + i + dup i + size <      ( don't go too far)
      IF
        size flags + over flags + i +      ( range of nums to tag)
        DO
          0 i c! dup      ( tag numbers as non-primes)
        +LOOP
      THEN
        drop 1+      ( drop the i used for +loop, increment prime count)
      THEN
    LOOP ;

: sieve
  COUNTER      ( start counting )
  10 0 DO primes LOOP      ( perform 10 iterations )
  TIMER        ( stop counting )
  CR ." primes"      ( print the number of primes )
  9 0 DO DROP LOOP ;      ( clean-up stack )

CR .( Type 'sieve' to execute this benchmark program ) CR
```

Forth Sieve. Uses pointer arithmetic to calculate the number of primes from zero through 16383. To save space and time, it only needs to work with the 8192 odd numbers.

### Thomas' Bit-Mask Locator

```
Listing 1
Screen #5
0. \ masking-number calculator      gtAug86
1. | SPC 32 HOLD |
2. | 16BITS ( FD 7/5, M.Ham, modified) CR ." HI"
3. | <# # # # ( hi nibble) SPC # # # # SPC SPC
4. | # # # # SPC # # # # > ( lo nibble) TYPE ." ILO" |
5. | .BITS ( FD 7/5, M.Ham, modified) BASE @ SWAP
6. | 2 BASE ! 8->D 16BITS BASE ! QUIT ;
7. | BAS= BASE @ DUP DECIMAL ." In base " . BASE ! ;
8.
9. | SEE-MASK DUP CR U. BAS= .BITS ;
10. | MASK DUP IF 1 SWAP SLA ( left shift) ELSE 1 OR THEN ;
11.
12. | BIT-MASK ( n -- |display number to mask bit n; n=0 thru 15)
13. | MASK SEE-MASK ;
14. | 2BIT-MASK ( n1 n2 -- |display number to mask bits n1 & n2)
15. | MASK SWAP MASK + SEE-MASK ;
```

### FORTHkit

5 Mips computer kit

**\$400**

#### Includes:

Novix NC4000 micro  
160x100mm Fk3 board  
Press-fit sockets  
2 4K PROMs

#### Instructions:

Easy assembly  
cmFORTH listing  
shadows  
Application Notes  
Brodie on NC4000

#### You provide:

6 Static RAMs  
4 or 5 MHz oscillator  
Misc. parts  
250mA @ 5V  
Serial line to host

#### Supports:

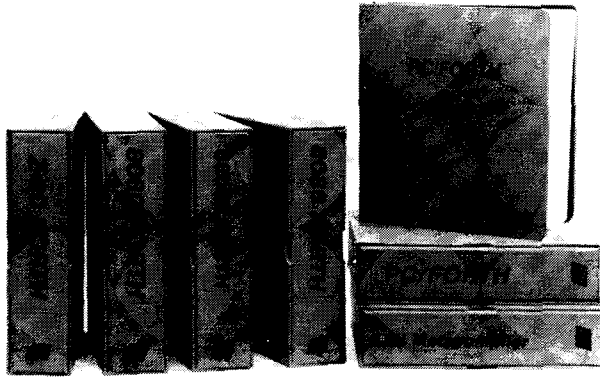
8 Pin/socket slots  
Eurocard connector  
Floppy, printer,  
video I/O  
272K on-board memory  
Maxim RS-232 chip

#### Inquire:

**Chuck Moore's  
Computer Cowboys**

410 Star Hill Road  
Woodside, CA 94062  
(415) 851-4362

# TOTAL CONTROL with LMI FORTH™



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

### For Development:

#### Interactive Forth-83 Interpreter/Compilers

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

### For Applications: Forth-83 Metacompiler

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

### For Speed: CForth Application Compiler

- Translates "high-level" Forth into in-line, optimized machine code
- Can generate ROMable code

### Support Services for registered users:

- Technical Assistance Hotline
- Periodic newsletters and low-cost updates
- Bulletin Board System

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

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

#### Overseas Distributors.

Germany: Forth-Systeme Angelika Flesch, Titisee-Neustadt. 7651-1665

UK: System Science Ltd., London, 01-248 0962

France: Micro-Sigma S.A.R.L., Paris, (1) 42.65.95.16

Japan: Southern Pacific Ltd., Yokohama, 045-314-9514

Australia: Wave-onic Associates, Wilson, W.A., (09) 451-2946

Conveniently, both the base of the mask and its binary representation are displayed. (Remember, the sixteen bits are numbered zero through fifteen.)

The word **SLA** in **MASK** is my system's **ML** shift-left arithmetic word (`n1 cnt -- n2`). Replace it with your appropriate instruction. The **1 OR** in **MASK** takes care of the zero bit position, as in **0 BIT-MASK**.

*Forth Dimensions* and its contributors often supply me with either some finishing touches or an idea to expand on. Thanks!

Sincerely,

Gene Thomas  
Little Rock, Arkansas

Student Roots

Dear Editor,

During this Summer Quarter of 1986, I have been providing the coursework for a student taking "Forth Programming" at Auburn University at Montgomery. As one of his assignments, this student (Hunter Moseley) was required to write a square root in Forth (F83) based upon a Newton's method-type algorithm. However, Hunter went beyond my thought and wrote code that put mine to shame. My code is shown in Figure One.

The **D\*/** used does the same thing as **\*/** but with double-precision numbers. In other words, (`d1 d2 d3 -- d4`). Also, the **2NIP** is a double-precision **NIP**. I hated to use the double-precision words, but for the accuracy needed, they were necessary.

Hunter's code was simply that shown in Figure Two.

In a time test on a Zenith-151 with 10,000 iterations, dropping the result each time, Hunter's code guaranteed 119 seconds with any input from zero to 32,766. Mine, however, with an equivalent range of inputs, does the square root of one in seventy-five seconds, the square root of two in 280 seconds, and gets even worse after that.

As can be seen, the two approaches are based on the same idea, but Hunter's does no bound checking. His



### Davies' Square Roots

```
: SQR ( d1 d2 -- d3 )
  RECURSIVE
    2OVER 2OVER 2OVER 10000 0 2ROT D*/ 2SWAP
    2OVER 10000 0 2SWAP D*/ 2OVER D- DABS
    5 0 D< IF 2NIP 2NIP EXIT
      ELSE D+ D2/
      THEN
    SQR ;

: SQR ( n1 -- n2 )
  10000 *D 10000 0 SQR 10000 UM/MOD NIP ;
```

Figure One

```
: SQR ( n1 -- n2 )
  1 10 0 DO 2DUP / + 2/ LOOP NIP ;
```

Figure Two

```
CODE SQR ( n1 -- n2 )
  DX POP SI PUSH DX SI MOV 1 # BX MOV
  10 DO
    DX DX XOR SI AX MOV BX DIV AX BX ADD BX SAR
  LOOP
  SI POP BX PUSH NEXT END-CODE
```

Figure Three

```
: DSQR ( d1 -- d2 )
  1. 19 0 DO 2OVER 2OVER D/ D+ D2/
    LOOP 2SWAP 2DROP ;
```

Figure Four

simpler application of the algorithm is much slicker — beauty in Forth.

Additionally, as an experiment with F83's assembler, I translated Hunter's algorithm into assembly. The code is listed in Figure Three. A time test on the Zenith-151 with 10,000 iterations, dropping the result each time, guaranteed five seconds! Yes, that's right — 2,000 iterations per second! Perhaps this amazes no one else, but I was somewhat shocked.

For those interested, Hunter also has the signed, double-precision version of the square root. The code is in Figure

Four. The **D/** is a double-precision divide. If anyone is interested in the code for these operators and their double-precision primitives, I will gladly share them.

In any case, I present these attempts as examples of how traditional mathematical thought sometimes must give way to the more efficient patterns used by our friends — the computers — and Forth.

Sincerely yours,

R.L. Davies  
Montgomery, Alabama

### Second Take:

### Multiple LEAVES by Relay

Dear Mr. Ouverson:

Please discard my previous letter to you (*Forth Dimensions* VIII/3), as it was completely erroneous. My intended verification test wound up with confusion between the fig-FORTH words in my system and the new words, due to my carelessness! Here is the new manuscript:

John Hayes' "Another Forth-83 LEAVE" (VII/1) stimulated me to try to find an even simpler way to handle multiple Forth-83 LEAVES. I decided that a straight-forward approach involved having each LEAVE simply branch to the next LEAVE, with the last one removing the index values from the return stack and branching to the word following LOOP.

I "grafted" such a construction onto fig-FORTH with the definitions below; words with a \* prefix are used to identify changes from fig-FORTH. Unstarred words such as (DO) and (LOOP) are unchanged. Whenever a \*LEAVE is compiled, the variable PLACE is used to hold the location of its branch value for later adjustment. This variable also serves as a flag to show that there is a leave branch to be resolved. \*LOOP calls a >RESOLVE to install the jump value of the preceding (if any) \*LEAVE; also, if there is a \*LEAVE in the word, a special OUTLEAVE is compiled immediately following the (LOOP) closure. OUTLEAVE removes the (two) loop parameters from the return stack and proceeds to the next word, i.e., the word that was entered after \*LOOP. If the \*LEAVE command is not invoked at run time, the normal loop operation removes these parameters from the return stack, so OUTLEAVE must be skipped over. \*LOOP compiles this bypass with a BRANCH 4 which is encountered in normal loop completion. Alternatively, (LOOP) could be modified to use OUTLEAVE in normal loop completion.

Note that OUTLEAVE can be a primitive which removes two words from the return stack by using PLA four times. If OUTLEAVE is defined as a

### Page's LEAVE by Relay

```
0 VARIABLE PLACE
: >RESOLVE HERE OVER - SWAP ! ;
: <RESOLVE HERE - ;
: OUTLEAVE R> R> DROP R> DROP R> ;
: *DO 0 PLACE ! COMPILER (DO) HERE 3 ; IMMEDIATE
\ Same as FIG DO with insertion of 0 PLACE !
: *LOOP 3 ?PAIRS COMPILER (LOOP) (RESOLVE \ usual, plus following
PLACE @ ?DUP IF COMPILER BRANCH 4 , \ for skipping OUTLEAVE
>RESOLVE COMPILER OUTLEAVE THEN 0 PLACE ! ; IMMEDIATE
: *LEAVE PLACE @ ?DUP IF >RESOLVE THEN \ resolve any preceding LEAVE
COMPILER BRANCH HERE PLACE ! 0 , ; IMMEDIATE

: TEST ( n1 n2---) *DO 1 5 = IF 1 . *LEAVE THEN
1 10 = IF 1 . *LEAVE THEN *LOOP ." END " ;
: TEST1 5 0 *DO 6 1 *DO 1 5 = IF 1 . *LEAVE THEN *LOOP ." END " *LOOP ;
: TEST2 5 0 *DO 6 1 *DO *LOOP 1 3 = IF 1 . *LEAVE THEN *LOOP ." END " ;
: TEST3 ( n1 n2---) *DO 7 0 *DO 1 3 =
IF 1 . *LEAVE THEN *LOOP ." INNER "
1 5 = IF 1 . *LEAVE THEN *LOOP ." END " ;
```

### Borenstein's Fixed-Point Trig

```
SCR #1
0 ( SIN SCALED BY 3784 )
1 ( 0 < X < 5944 <==> 0 < X < 90 DEG )
2 ( 0 < SIN X < 3784 <==> 0 < SIN X < 1 )
3 0 VARIABLE XS
4 : DEG 6604 100 */ ;
5 : KTIMES 17321 M* SWAP DROP ;
6 : TERM1 XS @ U* DROP U* SWAP DROP MINUS ;
7
8 : SIN1
9 DUP 256 > IF ( Check on small X )
10 DUP ( Leave one copy of X on stack )
11 DUP U* SWAP DROP DUP XS ! ( XS=Z*Z/2^16 )
12 4 U* DROP MINUS ( 2^16 - 4*XS )
13 7 TERM1 15 TERM1 50 TERM1
14 U* SWAP DROP
15 THEN ;

SCR #2
0 ( COS1 TAN1 SIN COS TAN )
1
2 : ?MIRROR DUP 5944 > IF 11888 SWAP - THEN ;
3
4 : REDUCE 23776 MOD DUP 0 < IF 23776 + THEN DUP 11888 < IF
5 ?MIRROR ELSE 11888 - ?MIRROR MINUS THEN ;
6
7 : SIN REDUCE DUP ABS SIN1 SWAP 0 < IF MINUS THEN ;
8
9 : COS1 5944 SWAP - SIN1 ;
10 : COS 5944 SWAP - SIN ;
11
12 : TAN1 DUP SIN1 3784 U* ROT COS1 461 MAX U/MOD SWAP DROP ;
13
14 : TAN 11888 MOD DUP 0 < IF 11888 + THEN DUP 5944 > IF
15 11888 SWAP - TAN1 MINUS ELSE TAN1 THEN ;
```

colon word, its first operation is to call the generic colon-word procedure which pushes the compilation address of the following word (the desired one at the end of the loop) onto the return stack, *above* the parameter values to be removed. In this case, the top value must be saved and restored by defining:

```
: OUTLEAVE
R> R> DROP
R> DROP > R ;
```

The test words use this **\*LEAVE** in multiple occurrence in a single loop; in single occurrence in both inner and outer of two nested loops; and in both inner and outer loops. In nested loops, any **\*LEAVE** in an outer loop must occur subsequent to the end of the inner loop. Starting a new loop before the forward resolution of the **\*LEAVE** jump would cancel the record of the **\*LEAVE** (in **PLACE**).

Chester H. Page  
Silver Spring, Maryland

### MMS Upgrade Offer

Dear Editor:

Any licensed MMSFORTH user who had not received a gold-colored v2.4 discount letter by the end of September 1986 — please notify MMS of your current address so we can send it along, or call us for further information.

Sincerely,

A. Richard Miller  
Miller Microcomputer Services  
61 Lake Shore Road  
Natick, Massachusetts 01760-2099

### Fixed-Point Trig

*In the May/June 1986 issue of Forth Dimensions, we carried an article titled "Fast Fixed-Point Trig" by Johann Borenstein. Due to space limitations, we were unable to print the companion screens to the article in that issue. You will find them herewith.*

—Editor

An invitation to attend the eighth annual

# FORML CONFERENCE

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

**Following Thanksgiving  
November 28 - 30, 1986**

## **Asilomar Conference Center**

**Monterey Peninsula overlooking the Pacific Ocean  
Pacific Grove, California**


### **Theme: Extending Forth towards the 87-Standard**

FORML isn't part of the Standards Team, but the conference is an opportunity to present your ideas for additions to the Forth standard. Papers are also welcome on other Forth topics. Meet other Forth professionals and learn about the state of the art in Forth applications, techniques, and directions.

**To get your registration packet call the FIG Business Office (408) 277-0668  
or write to: FORML Registration, Forth Interest Group, P. O. Box 8231,  
San Jose, CA 95155.**

**Registration:** \$275 Double Room  
\$325 Single Room (Limited availability)  
\$150 Non-conference guest (Share a double room)

Registration includes room, meals, conference materials, and social events.



**Space is limited,  
advance registration  
is required.**



## Conventions

We've talked about this before, but someone — one of our authors, yet — recently confused F83 with Forth-83. Big mistake! F83 is an ultra-superset of Forth-83, nearly an order of magnitude larger. Forth-83 and 83-Standard are common shorthand for the phrase, "Forth-83 Standard." The name F83 is not an even shorterhand, it is the name of an implementation of Forth. Does everyone understand the difference between a language implementation and a standard?

As we send this issue to press, last minute preparations are underway for the imminent 1986 Forth National Convention. A major component of this year's convention is a six-part seminar on Forth engines. Those sessions will focus on the new multiple-stack WISC (writeable instruction set computer) machine; applications of the Novix 4000 and the design of the Novix 6000 chip; Forth engines developed by Hartronix, Lockheed and Johns Hopkins; ROM-based Forth engines (i.e.,

the Super-8, R65F11 and F68HC11 microprocessors); Forth engine software; and the future of Forth engines.

Numerous additional events and concurrent sessions are planned to serve the particular interests of all attendees. Exhibitors will include major vendors of commercial Forth hardware and software. Special groups will gather to discuss F83, MacForth/MultiForth, MVP-FORTH, NC4000, polyFORTH and 68000 machines (e.g., Macintosh, Atari, Amiga). There will be tutorials about control structure extensions, files and string I/O, multi-tasking in polyFORTH, oblique flying wings, target compilation in F83 and vectored execution of I/O words.

A FIGGRAPH session will feature the latest in computer-generated graphics of significance to the Forth community. FIG chapters' representatives will convene, and there will be a national meeting of Forth Interest Group members. As in past years, the convention will also feature a banquet with

keynote speaker (separate registration required to reserve a seat), a report from FORML including this year's trip to important Forth sites in China, and a "fireside chat" with Mr. Charles Moore, original developer of Forth.

There you have it in a nutshell, or perhaps in a kernel. It seems that as Forth has matured, it has gathered a potency which can propel it into new areas. This integral vitality can lead Forth in unexpected, surprising directions. Keep abreast by joining us in California on November 21-22 at the Santa Clara Trade & Convention Center, near the new Doubletree Hotel. And for an intensive immersion in Forth methodology and experimental proposals, stay for the following weekend's FORML conference at the Asilomar conference grounds in Pacific Grove, adjacent to Monterey. Information for either event can be obtained by calling 408/277-0668, the FIG hotline.

—Marlin Ouverson  
Editor

### THE Journal OF Forth Application AND Research

The Journal contains applications and techniques papers, technical notes, review papers, book reviews, algorithms, and conference abstracts.

The aim of the **Journal of Forth Application and Research** is to provide a reliable source of state-of-the-art techniques and applications of Forth to scientific and industrial problems. The **Journal** is an international quarterly with frequent submissions from Europe and Canada. Past issues had in-depth coverage of such topics as Robotics, Data Structures, Forth Computers, Real Time Systems, and Extended Address Computing. The **Journal** is open to all new work in Forth as well as the entire range of threaded interpretive languages and Forth-like systems.

Subscriptions Volume 4, 1986 — Corporations and Institutions, \$100.; Individual subscribers, \$40. Please add \$20 for airmail delivery outside N. America, and make checks in US funds from a US bank or international money order, payable to the **Journal of Forth Application and Research**.

For further information on the Journal, subscriptions and back issues, or other Institute publications and activities, please contact:

The Institute for Applied Forth Research, Inc.  
PO Box 27686  
Rochester, NY 14627 USA

# Dual-CFA Definitions



Mike Elola  
San Jose, California

The introduction of a dual-CFA definition structure provides a new building block for Forth programming. It also provides a new method for decomposing larger functions into smaller functions.

Because of the new possibilities afforded by dual-CFA decomposition, worthwhile changes to many implementations of Forth can be formulated. By contrasting these new definitions with the alternatives currently available, the value of this new methodology can be demonstrated.

Changes are suggested within each of three different areas common to most implementations of Forth. The areas to be covered are (1) deferred definitions, (2) vectored definitions and (3) definitions that dispatch multiple functions. This roughly parallels the organization of topics in my original paper describing dual-CFA decomposition<sup>1</sup>. It showed that dual-CFA definitions help maintain a consistent strategy for decomposition and that this strategy can be the basis of a Forth programming philosophy aimed at memory compactness, brevity of expression and ease of programming.

## Deferred Definitions

Deferred definitions are used to allow a lower-level word to dispatch a function that is defined in terms of many high-level support words. When definitions that require the undefined function are compiled, a superficial, "stand-in" definition is compiled in lieu of the actual, desired function. Later, the stand-in definition's body is modified to reference the correct, high-level definition.

Dual-CFA definitions can be used to implement deferred definitions. The dual-CFA word functions as the stand-in definition that is modified later when the dictionary contains the support needed for compiling the "real" definition.

In this implementation, the dual-CFA definition performs a self-modification

function (aided by the dual-CFA definer). The child transforms itself into a parentless, single-CFA definition when executed. After execution, it contains a reference to a headerless definition at the top of the dictionary (see Figure One).

The definition for the parent definer is:

```
: DEFER: <word> ( -- )
CREATE DOCOL , COMPILE-DEF
DOES> ( cfa2 -- )
  DUP @ OVER 2- ! ( normalizing cfa1 )
  DUP @ , ( compiles docol )
  HERE 2- OVER ! ( overwriting cfa2 )
NFA ." COMPILING BODY OF " ID. CR
COMPILE-DEF ;
```

Dictionary entry before execution:

CHILD'S NAME	DOES> CFA	DOCOL CFA	CALL TO EXIT
↓			
CHILD'S NAME	DOCOL CFA	UPSTREAM CALL	CALL TO EXIT

Dictionary entry after execution:

Self-transformation of a dual-CFA word created with DEFER

Figure One

```
: FAILING-NUMBER: ( -- )
CREATE DOCOL , COMPILE-DEF
DOES> ( string-add cfa2 -- d )
  >R DUP 1+ C@ 45 = DUP
  >R +
  0 0 ROT BEGIN
  CONVERT DUP C@ 32 - WHILE
  DUP C@ DUP 58 =
  SWAP 44 48 WITHIN +
  NOT IF ROT ROT DDROP R> DROP R>
  ( add cfa2 -- ) EXECUTE THEN
  REPEAT DROP
  R> R> DROP IF DNEGATE THEN ;
```

Figure Two

The advantages of this technique are the unavailability of the definition-modifying function except to the uninitialized children of DEFER:, the use of only one name field (as compared usually to two) and the decreased chance of crashing (since the definition-modifying action can't be applied to just any word).

Note that the first advantage is actually a limitation that may not appear advantageous to some. In F83, defer-

red definitions are initialized with IS, which patches the specified word. IS can be used more than once and can be used with any word, not just those words that are created with DEFER.

A new disadvantage regarding the dual-CFA implementation is that the deferred function must be specified in a non-standard way: the name of the deferred definition replaces the colon and name string at the start of the definition. Because of this, it is more difficult to separately recompile the high-level definition again. For example, DEFER: may be used to create ?ERROR, which is defined at a later time in the following manner:

```
?ERROR ( flag -- )
IF ." ERROR " ABORT THEN ;
```

To separately recompile ?ERROR again, a colon must be provided. However, the first body of ?ERROR still points at the original, headerless definition.

One solution is to provide another support word:

```
: REDEFER: <old deferred word> ( -- )
[COMPILE] ' ( PFA )
DEFERRED-CFA SWAP CFA ! ;
```

Now you need not change the source code by adding a colon. Instead, you enter REDEFER: ?ERROR as a preparative step. Then you can load the source code as-is.

In the preceding definition, DEFERRED-CFA is a constant. It points to the DOES> phrase in the parent defining word. Its derivation was not shown. One way to derive it is to use:

```
DEFER: JUNK
LATEST PFA CFA @ ( cfa-value -- )
FORGET JUNK
CONSTANT DEFERRED-CFA
```

Note that REDEFER: increases the chance for crashes, since its definition-modifying function is not restricted to deferred words. To remedy this, extra code can be added to the definition to ensure that it contains an upstream reference:

## SOFTWARE for the **HARDCORE**

### MasterFORTH

#### FORTH-83 STANDARD

- 6809 Systems available for FLEX disk systems ..... \$150
- OS9/6809 ..... \$150
- 680x0 Systems available for MACINTOSH ..... \$125
- CP/M-68K ..... \$150
- tFORTH/20 for 68020 Single Board Computer
- Disk based development system under OS9/68K ... \$290
- EpROM set for complete stand-alone SBC ..... \$390
- Forth Model Library - List handler, spreadsheet, Automatic structure charts ... each . \$40
- Target compilers : 6809,6801, 6303, 680x0, 8088, 280, 6502

**Talbot Microsystems**  
1927 Curtis Ave  
Redondo Beach  
CA 90278  
(213) 376-9941

## HARDWARE for the **HARDCORE**

68020 SBC, 5 1/4" floppy size board with 2MB RAM, 4 x 64K EpROM sockets, 4 RS232 ports, Centronics parallel port, timer, battery backed date/time, interface to 2 5 1/4" floppies and a SASI interface to 2 winchester disks ..... \$2750

68881 flt pt option ..... \$500

OS9 multitask&user OS.. \$350

**FAST!** int. benchmarks speeds are

2 x a VAX780, 10 x an IBM PC

### Listing One

Processing of counted strings (i.e., already-parsed words)

```

Target Stack Effect( cadd -- )

: FAILING-LOOKUP: ( <error-processing-function> )
  CREATE DOCOL , COMPILE-DEF
  DOES> ( cfa2 ) >R ( cadd [ pfa len ] flag -- )
  DUP CONTEXT @ @ (FIND) ( cadd [ pfa len ] flag )
  0= IF R> ( cadd cfa2 -- ) EXECUTE ELSE R> DROP THEN ;

FAILING-LOOKUP: ?INTERPRET-NUMBER; ( cadd -- ? )
NUMBER-VALUE? ?STACK ( overflow? )
DPL @ 0< IF DROP THEN R> R> ZDROP ;

FAILING-LOOKUP: ?COMPILE-NUMBER; ( cadd -- ? )
NUMBER-VALUE? DPL @ 1+ IF
[COMPILE] DLITERAL ELSE [COMPILE] LITERAL THEN R> R> ZDROP ;

: INTERPRET-WORD ( cadd -- )
  ?INTERPRET-NUMBER;
  ( cadd pfa len -- ) ROT ZDROP
  ( pfa ) CFA EXECUTE ?STACK ( underflow? ) ;

: INTERPRET-WORD ( cadd -- )
  ?COMPILE-NUMBER;
  ( cadd pfa len -- ) ROT DROP
  ( pfa len ) 192 > IF CFA EXECUTE ELSE CFA , EXECUTE THEN ;

```

#### Word Parsers

```

Target Stack Effect( -- flag )

: (WORD) ( stream-add -- flag )
  ...TO.BE.SUPPLIED... HERE C@ ;

: TIB-WORD ( c -- flag )
  TIB @ (WORD) ;

: BLK-WORD ( c -- flag )
  BLK @ BLOCK (WORD) ;

```

#### Null-delimited input stream parsers/processors

```

Target Stack Effect( -- )

VARIABLE PROCESS-WORD'

STREAM-PROCESSOR:
( <parsing-function-leaving-counted-string-at-dp> )
  CREATE DOCOL , COMPILE-DEF
  DOES> ( cfa2 -- ) >R
  BEGIN R@ EXECUTE WHILE
  HERE PROCESS-WORD' @ EXECUTE REPEAT
  R> DROP ;

STREAM-PROCESSOR: TIB-PROCESS ( -- )
  32 TIB-WORD ;

STREAM-PROCESSOR: BLK-PROCESS ( -- )
  32 BLK-WORD ;

```

## Listing Two

```

STREAM-PROCESSOR: BLK-PROCESS.
32 BLK-WORD ( flag -- )
SPACE HERE COUNT TYPE ;

: PRINTING-LOAD ( blk# -- )
0 >IN !
BLK-PROCESS. ;
    
```

```

: REDEFER: <old deferred word> ( -- )
[COMPILE] ' ( PFA )
DUP @ U< 0= IF
CR ." - MUST BE A DEFERRED WORD"
ABORT THEN
DEFERRED-CFA SWAP CFA ! ;
    
```

### Vectored Definitions

Vectored definitions can often be replaced by fixed-behavior, dual-CFA words. Such words can directly invoke the desired function. For flexible processing, a variety of these words can be defined. Each one would be suited to use in a particular context.

Dual-CFA words offer flexibility in a fundamental Forth form: compile-time selection of the desired behavior by a reference to the correct word from the dictionary. This practice retains the ease-of-use that characterizes normal, fixed-behavior words<sup>1</sup>.

For example, **NUMBER** often employs a vector to provide a means for flexible processing. Because the behavior you desire is usually known at compile time, you do not really need run-time flexibility — just a wider selection of compilable behaviors. This makes **NUMBER** a good candidate for dual-CFA decomposition.

The function of **NUMBER** is to convert an input string into a number. When the conversion process fails, program execution is immediately aborted in many Forth implementations. Such an outcome is fine during the interpret or compile phase, but often is undesirable in a finished application.

Several versions of **NUMBER** are needed. Each would have a different failure outcome. This can be achieved by creating a definer word that incorporates **NUMBER** (see Figure Two).

To define a number-conversion routine suitable for internal use when interpreting or compiling, one child definition might be:

```

FAILING-NUMBER:
NUMBER-VALUE? ( string-addr -- d? )
CR COUNT TYPE
." - NOT RECOGNIZED" ABORT ;
    
```

To define a version of **NUMBER** more suitable to an application, another child definition could be:

```

FAILING-NUMBER:
INPUT-NUMBER ( string-addr -- d? )
C@ BACKSPACES
TIB 12 EXPECT 0 >IN !
32 WORD HERE RECURSE ;
    
```

The advantages of the dual-CFA definitions over vectored definitions are the memory compactness of the compiled words, the absence of intermediary variables, the absence of required initializations and greater immunity to crashes.


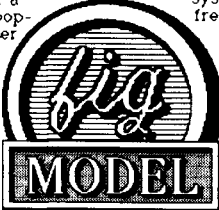


Normally, decomposition of the error-handling code within **NUMBER** would not be possible, unless such code is moved outside of the **BEGIN WHILE REPEAT** loop. This would allow the error instructions to be separately specified within any number of parent definitions.

This is similar to the approach taken in F83. This Forth implementation includes a primitive version of **NUMBER** that does not abort on error. Instead, it leaves a flag on the stack that can be used by parent words to trigger any kind of error processing desired. Since parent words must process the flag left on the stack, a conditional phrase is normally required in all the parent words where (**NUMBER**) is used.

# FORTH

## FREEDOM OF CHOICE

SOTA Computing Systems Limited lets you choose between either the versatile figFORTH model or the popular 79 Standard. Each version is available for a number of popular computer systems including the IBM PC, XT and AT (or compatibles), the TRS-80 Model 1, III and 4/4P, or any computer system running CP/M (version 2.x) or CP/M Plus (version 3.x). What's more, SOTA doesn't require you to enter into any awkward or expensive royalty or licensing arrangements. As long as your applications programs do not offer the end user access to the basic FORTH system, you are free to make as many copies of the compiled FORTH system as you please and distribute them as you wish. FORTH from SOTA is the FORTH of choice for both the novice and experienced programmer. Make it your choice now! Order your copy today.

When you order from SOTA, both the fig model and 79 standard come complete with the following extra features at no additional charge:

- full featured string handling • assembler • screen editor • floating point • double word extension set • relocating loader • beginner's tutorial • comprehensive programmer's guide • exhaustive reference manual • unparalleled technical support • source listings • unbeatable price •

### ORDER FORM

GENTLEMEN: Rush me my order!

Enclosed is my:  check  money-order

Please bill my:  VISA  MasterCard

for \$89.95

Please send me:  79 Standard FORTH  figFORTH model for the

IBM PC  XT  AT (and compatibles)

TRS-80 Model 1  Model III  Model 4  Model 4P

CP/M Version 2.x  CP/M Plus (Version 3.x)

For CP/M versions please note: 5 1/4" formats only and please specify computer type:

NAME: \_\_\_\_\_

STREET: \_\_\_\_\_

CITY/TOWN: \_\_\_\_\_

STATE: \_\_\_\_\_ ZIP: \_\_\_\_\_

CARD TYPE: \_\_\_\_\_ EXPIRY: \_\_\_\_\_

CARD NO: \_\_\_\_\_

SIGNATURE: \_\_\_\_\_

**ORDER TODAY** 213-1080 Broughton Street  
Vancouver, British Columbia  
Canada • V6G 2A8

Order by Mail or Phone  
• (604) 688-5009 •

State of the Art since 1981

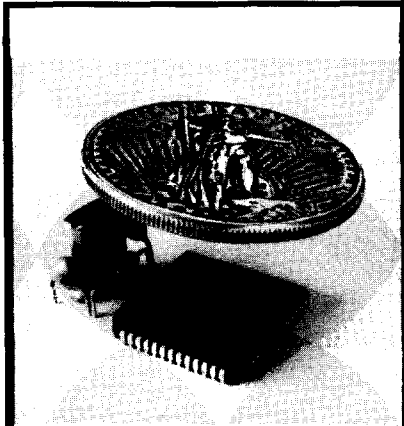
# SOTA

Computing Systems Limited

IBM, TRS-80 and CP/M are registered trademarks of International Business Machine Corporation, Radio Shack and Digital Research respectively.

All the parts needed to make the

# SMALLEST PROGRAMMABLE FORTH SYSTEM:



&  
+5V (9 mA, typical @ 2 MHz)  
TTL Serial In  
TTL Serial Out  
Ground

\$50 covers price of parts and manual in singles, \$20 covers cost of chip alone in 10,000 quantity. \$20 gold piece (not included) shown covering chip to illustrate actual size.

The F68HC11 features: 2 Serial Channels, 5 Ports, 8 Channel 8-bit A/D, major timer counter subsystem, Pulse Accumulator, Watchdog Timer, Computer Operating Properly (COP) Monitor, 512 bytes EEPROM, 256 bytes RAM, 8K byte ROM with FORTH-83 Standard implementation.

Availability: F68HC11 Production units with Max-FORTH™ in internal ROM available 4Q/86. Volume quantity available 1Q/87. X68HC11 emulator with Max-FORTH™ in external ROM available now. NMIX-0022 68HC11 Development System boards available now: \$290.00.

**New Micros, Inc.**  
808 Dalworth  
Grand Prairie, TX 75050  
(214) 642-5494



NEW MICROS INC.  
808 DALWORTH  
GRAND PRAIRIE, TEXAS 75050  
214/642-5494

To avoid having to repeat that failure processing with each use of the F83 **NUMBER** primitive, enlarged functions could be created. If desired, versions equivalent to **INPUT-NUMBER** and **NUMBER-VALUE** could be created. Such versions would exhibit the same ease-of-use as the dual-CFA versions.

The dual-CFA versions would retain a very slight advantage over their F83 equivalents: they should compile in less memory and should execute slightly faster due to a decreased number of conditionals.

### Definitions That Dispatch Multiple Functions

When implementing function-dispatching words, dual-CFA definitions can also be advantageous. Listing One includes several examples that help illustrate those advantages.

In most Forth implementations, the main function-dispatching routine is **INTERPRET**. Both the compiling and interpreting functions are often performed within **INTERPRET**. Since there is so much commonality between these two distinct functions, it is easy to think of them as children of the same parent process. But what exactly should this parent process be? The answer can be found by more clearly discerning what functional areas are to be combined.

The common ground between the compiler and the interpreter is the input parser. The input parsing function is the same, whether compiling or interpreting. It remains a static function even if there are mid-line transitions between the compiling and interpreting functions.

But because of input redirection, the input-parsing function is not always static. When loading a block, the input source must be the block buffer, not the text input buffer (TIB). Normally, this flexibility is achieved as a run-time function of **WORD**. So **WORD** normally has a variable behavior dispatched through a conditional phrase. A majority of the time, this conditionally-variable behavior can be eliminated. Input redirection is rarely exercised during run time. For those exceptions, a variable-behavior version of **WORD** can be defined by referencing the fixed-behavior versions. By defining **LOAD**

with a fixed-behavior version of **WORD** that only parses blocks, the input redirection required by **LOAD** is enabled using only fixed-behavior words at compile time.

The implementation shown in Listing One factors the function of input-stream parsing to a parent definer. Each of the two dual-CFA children dispatch a different version of **WORD**. The correct one can then be selected at compile time to suit a given context. Refer to **STREAM-PROCESS**, **TIB-PROCESS** and **BLK-PROCESS** in Listing One (as well as **PRINTING-LOAD** in Listing Two).

To provide additional, run-time flexibility, **WORD** can be defined in terms of the new primitives:

```
: WORD ( c -- )  
  BLK @ IF BLOCK-WORD  
  ELSE TIB-WORD THEN DROP ;
```

To make a nicer, error-detecting version, the flag returned by the primitive versions of **WORD** could be processed as follows:

```
: WORD ( c -- )  
  BLK @ IF BLOCK-WORD  
  ELSE TIB-WORD  
  THEN 0 = IF  
  CR ." UNEXPECTED END-OF-INPUT."  
  ABORT THEN ;
```

The variable-behavior version of **WORD** is needed for single-word parsers such as ' (tick) and **CREATE**. Having all three versions (**WORD**, **TIB-WORD** and **BLOCK-WORD**) provides the programmer with more choices. Why use the variable-behavior version of **WORD**, with its extra overhead, when input-redirection flexibility is not necessary at run time?

As an extensible programming language, Forth can exhibit a wide range of functionality that broadens with each new word added. For every programming problem confronted, Forth can be extended in ways that make the solution easy to program. Not only is the original problem more easily solved, but also many related problems become easier to solve.

See Listing Two for a printing version of **LOAD** that is defined very simply. It could be useful on those few occasions when a screen will not load

*(Continued on page 32.)*



# Simple File Query



Edward Petsche  
Greenport, New York

This article describes an implementation of a simple file query based on the data-base elements presented in *Forth Dimensions* (see volumes three and four). The parameter fields of words defined by **FILE** and **FIELD** have been extended to include some new parameters necessary for the query. The **DOER** and **MAKE** vectored execution words described in *Thinking Forth* are also used in this program. Implementations of these words for various versions of Forth are given in that book's appendix. If you don't have access to that book, the implementation in screen 8 should work for all versions of Forth-83. If you prefer the **DEFER** and **IS** vectored execution words, the necessary modifications, aside from replacing **DOER** with **DEFER** (screen 16), involve only screen 23.

This program allows the user to define and initialize a file, enter data, query a file on any combination of fields, delete records and change field values of records.

**FILE** is the defining word for files. The PFA of a word defined by **FILE** contains the following parameters:

- byte  
offset
- 0 starting block of file
  - 2 maximum number of records for file
  - 4 bytes/block
  - 6 record length in bytes
  - 8 current record number
  - 10 address of list of fields for this file

**FIELD** is the defining word for fields. The PFA of a word defined by **FIELD** contains the following parameters:

- byte  
offset
- 0 field width
  - 2 byte offset from start of record
  - 4 field type

The first record of each file (**0 RECORD**) is used for information regarding the length of the file (**LASTREC**) and the number of active records in the

file (**#ACTIVE**). These items occupy the first four bytes of this record.

Screen 24 shows the file and field definitions for a sample application. Three parameters must be specified when defining a file: the starting block, maximum number of records and the record length. In the **EMPLOYEES** file definition, sixty-four is the record length, 100 is the maximum and thirty is the starting block.

When a field is defined, three parameters must be specified: field type, offset and field width. A width is specified even for numeric types for display formatting.

**FIELDS** compiles a list of CFAs of field words. The address of the start of this list is stored in **FIELD-LIST**. The syntax for **FIELDS** is: <file name> n **FIELDS**. The file name executes and becomes the current file. The number of fields is then left on the stack to control the loop that compiles the list.

After the program has been loaded along with the sample file application (screen 24), type **NEWFILE EMPLOYEES**. We are now ready to enter data into the **EMPLOYEES** file. Figure One shows a sample data entry session. Actually, the field entry prompts appear one at a time on the screen. When a field entry is terminated with a carriage return, the next field entry prompt will appear on the next line. After all the fields in **FIELD-LIST** have been entered, the user is asked if there is more data to be entered. The word **NEXTREC** in the **ENTRY** routine reclaims space occupied by deleted records.

Before querying a file, a display mode should be chosen by entering either the **STEP** or **SELECT** commands. **STEP** is the default mode. Figure Two shows a display using the **STEP** mode. It displays all the fields of each record found by the query. Records are displayed one at a time and the user is presented with the following options with each displayed record: modify a record, continue the query or quit. **STEP** does not require any arguments.

The **SELECT** display mode allows the user to choose which fields will be displayed. This mode prints a heading with the names of the selected fields.

The field values for each record found are displayed under the corresponding field name in the heading. The syntax for **SELECT** is:

```
SELECT <file name> <field1>  
<field2> ... <fieldn>
```

The maximum number of selections (**EXCERPTS**) is five. This is arbitrary. More fields could be selected depending on the total number of characters of the selections. They should all fit on one display line. An example of a **SELECT** display is given in Figure Three.

**FIND** is the end-user query word. It will search any combination of fields for each record in a file. The conditions are **GR.THAN**, **LS.THAN**, **IS** and **ISNT**. The logical operators **AND** and **OR** are also used by the query. The maximum number of conditions (**Q#**) for the query is set for four. This could be increased, but since **TIB** will only accept eighty characters, I felt this was a reasonable maximum. A query requiring more than eighty characters could be input from a block using **LOAD**. If **#ARGS** is modified to use a command-line delimiter, a number of query commands could be included on a block and loaded.

After the query command line has been entered, the program executes the next word in **TIB** which is a file name. It is now the current file. Next, the number of words that follow the file name in **TIB** is counted (**#ARGS**). This number is incremented by one. If dividing this number by four leaves a zero remainder, the number of arguments is valid. The quotient is the number of conditions for this query. This value is left on the stack to be used by **FOUND?** and **Q-ARRAYS**. Now the query arguments are stored. The search arguments (the values that are to be compared with the specified fields) are stored in **TARGETS**. The maximum number of search arguments is thirty. Strings for numeric search arguments are converted by **NUMBER** before they are moved to **TARGETS** by the word **BRING**.

The file is searched, checking each record to see if it is active (not removed). If it is active, the query arguments are executed by **FOUND?**, which

```

ok
ENTRY EMPLOYEES

NAME : SMITH
HOURLY-RATE : 8.00
HOURS : 35
DEPT : ACCT

any more? Y/N Y

NAME : ALLEN
HOURLY-RATE : 5.00
HOURS : 45
DEPT : MAINT

any more? Y/N No

```

**Figure One**

processes the query arguments for each condition to see if the current record satisfies the conditions. After all the conditions have been tested, a flag is left on the stack. If it is true, then the query conditions have been matched by the current record and it will be displayed.

A word to list the entire file has not been included in this program. The entire file can be listed by entering a query with conditions that will be satisfied for all records (e.g., **FIND EMPLOYEES NAME ISNT XXX**). The program includes very little error checking. If the user enters field names or conditions that have not been defined, the program aborts displaying the usual Forth system error message.

### Query Glossary

**'OPEN** Contains parameter field address of current file.

**'FIELD** Contains parameter field address of current field.

**FIELD-LIST** Address within parameter field of current file that contains address of list of fields for that file.

```

ok
STEP ok
FIND EMPLOYEES HOURS GR.THAN 30

NAME : SWIFT
HOURLY-RATE : 7.00
HOURS : 35
DEPT : DP

RETURN to quit ESC to modify
any key to continue <space bar>

NAME : WILSON
HOURLY-RATE : 7.75
HOURS : 40
DEPT : DP

RETURN to quit ESC to modify
any key to continue <ESC>

Enter C to change or R to remove record
Enter name of field to be changed
HOURS
HOURS : 30
RETURN to quit ESC to modify
any key to continue <space bar>

NAME : FIRTH
HOURLY-RATE : 5.50
HOURS : 40
DEPT : ACCT

RETURN to quit ESC to modify
any key to continue <CR>

query aborted

```

**Figure Two**

**LASTREC** First byte of record number zero. It contains record number of last record in current file.

**#ACTIVE** Third byte of record number zero. It contains the number of active records (not removed) in current file.

**FILE** File-defining word. When a word defined by **FILE** is executed, it places its parameter field address in **'OPEN**.

**FIELD** Field-defining word. When a word defined by **FIELD** is executed, it places its parameter field address in **'FIELD** and leaves the address of the field on the stack.

**FLD-WIDTH** Contains the width of the current field. A field width is required for all field types. For numbers, the field width is required for display formatting.

**FLD-TYPE** Field types are 0, 2, 4 and 6, for text, single numbers, double numbers and dollar amounts.

**TABLE** Defining word for execution tables of type-dependent functions. When executing, words defined by **TABLE** use the current field width to select a function to be executed.

```
ok
SELECT EMPLOYEES NAME DEPT HOURLY-RATE ok
FIND EMPLOYEES HOURS GR.THAN 35
NAME DEPT HOURLY-RATE
```

```
FIRTH ACCT 5.50
```

```
SMITH ACCT 8.00
```

```
ALLEN MAINT 5.00
```

```
BENSON DP 7.50
```

```
ok
```

```
FIND EMPLOYEES DEPT IS ACCT AND HOURS LS.THAN 40
NAME DEPT HOURLY-RATE
```

```
ROGERS ACCT 6.50
```

```
SMITH ACCT 8.00
```

```
ok
```

Figure Three

**(ENTER)** An execution table containing entry words for all field types. The words in this table all expect a field address on the stack at execution time.

**DISPLAY** An execution table containing display words for all field types. A field address is expected on the stack at execution time.

**COMPARE** An execution table containing words for comparing fields to search arguments. Words in this table expect two addresses on the stack and return -1, 0 or 1, for less than, equal or greater than.

**DASHES** Used for prompting input for record entry.

**ENTER** Prompts the user for a field entry. Accepts the input and stores the entry in the file.

**REMOVED?** True if record has been marked as deleted.

**#ARGS** Counts the number of arguments remaining in **TIB**. Should be modified if block input is to be used for commands.

**Q#** Maximum number of conditions searched for by query.

**#HITS** Number of records found. In this application, **#HITS** is only used as a flag, but it is easy to imagine other uses for it.

**LOGICALS** Array of logical operations (**AND** and **OR**) to be performed by query.

**OPERANDS** Array of field operands to be compared by query.

**CONDITIONS** Array of query conditions (**GR.THAN**, **LS.THAN**, **IS** or **ISNT**).

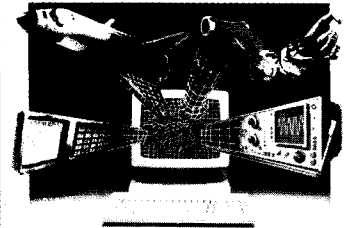
**TARGETS** Address of start of storage area for search arguments.

**+TARGET** Uses index on stack to offset into **TARGETS**.

**BRING** Execution table for words that bring the search arguments to **TARGETS**.

**GET-TARGET** Brings next word in **TIB** to **TARGETS** using index on stack to offset.

## polyFORTH GETS YOUR PROGRAM FROM CONCEPT TO REALITY 4 TO 10 TIMES FASTER



### THE ONLY INTEGRATED SOFTWARE DEVELOPMENT PACKAGE DESIGNED FOR REAL-TIME APPLICATIONS

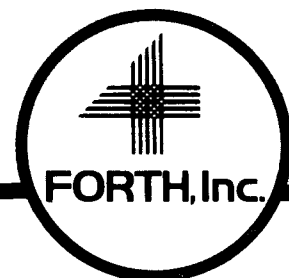
If you're a real-time software developer, polyFORTH can be your best ally in getting your program up and running on time. In fact, on the average, you will develop a program 4 to 10 times faster than with traditional programming languages.

polyFORTH shortens development time by making the best use of your time. There are no long waits while you load editors, compilers, assemblers, and other tools, no long waits while they run—because everything you need is in a single, easy-to-use, 100% resident system. Using polyFORTH, you take a raw idea to fast, compiled code in seconds—and then test it interactively.

polyFORTH has everything you need to develop real-time applications: fast multi-tasking, multi-user OS; FORTH compiler, interpreters, and assemblers; editor and utilities; and over 400 primitives and debugging aids. With its unique modular structure, polyFORTH even helps you test and debug custom hardware interactively, and it is available for most 8, 16, and 32-bit computers.

FORTH, Inc. also provides its customers with such professional support services as custom application programming, polyFORTH programming courses, and the FORTH, Inc. "Hotline."

For more information and a free brochure, contact FORTH, Inc. today. FORTH, Inc., 111 N. Sepulveda Blvd., Manhattan Beach, CA 90266. Phone (213) 372-8493.



**.HEADER** A quick and dirty formatting word for the **SELECT** display mode.

**SPREAD** A quick and dirty formatting word which attempts to keep the displayed field values lined up under the field names in the display header.

**SELECT** End-user word for choosing fields to be displayed. Maximum number is five in this application. Can be changed, but the total number of characters of the fields selected should be less than eighty.

**FIELDS** Includes fields in the field list for a file after they have been defined. Expects the number of fields on the stack (<file name> n **FIELDS**).

**STEP** An end-user word to control display. All fields for a record will be displayed, one record at a time.

**NEWFILE** Initializes file by setting **LASTREC** and **#ACTIVE** to zero.

**Q-ARRAYS** Uses the number on stack — which is the number of conditions

for a particular query — as a loop index to load query arrays with arguments from **TIB**. First entry in **LOGICALS** is a no-op word.

**FOUND?** Compares fields with search arguments to determine if query conditions are satisfied.

**FROM** Executes the next word in **TIB** which is a file name.

**(FIND)** Examines every record in the current file, checking first to see if the

#### Screen # 8

```
\ DOER/MAKE
: NOTHING ;
: DOER CREATE ['] NOTHING >BODY , DOES> @ >R ;
VARIABLE MARKER
: (MAKE) R> DUP 2+ DUP 2+ SWAP @ >BODY ! @ ?DUP
  IF >R THEN ;
: MAKE STATE @ IF COMPILE (MAKE) HERE MARKER ! 0 ,
  ELSE HERE [COMPILE] ' >BODY ! 1 STATE ! INTERPRET
  THEN ; IMMEDIATE
: ;AND COMPILE EXIT HERE MARKER @ ! ; IMMEDIATE
: UNDO ['] NOTHING >BODY [COMPILE] ' >BODY ! ;
```

#### Screen # 11

```
\ FILE words from FORTH Dimensions Vol. IV # 5
VARIABLE 'OPEN \ points to current file block
: REC# 'OPEN @ 0 + ; \ holds current record number
: LAYOUT \ leave bytes/record-2, bytes/block-1
  'OPEN @ 4 + 2@ ;
: MAXRECS ( -- n) 'OPEN @ 2+ @ ;
: READ ( n-th rec, on stack, is made current )
  0 MAX DUP MAXRECS < IF-NOT ." file error " QUIT THEN
  REC# ! ;
: RECORD ( n -- a) \ leave address of n-th record
  LAYOUT */MOD 'OPEN @ @ + BLOCK + ;
: ADDRESS ( -- a) \ leave address of current record
  REC# @ RECORD ;
: FIELD-LIST ( -- a) 'OPEN @ 10 + ;
: REC-LEN 'OPEN @ 6 + @ ;
```

#### Screen # 10

```
\ System extension words elp 03sep85
0 CONSTANT FALSE
-1 CONSTANT TRUE
: BLANK-PAD PAD 80 BL FILL ;
: TEXT ( c --) BLANK-PAD WORD COUNT PAD SWAP CMOVE> ;
: -TEXT ( adr n adr -- n) 2DUP + SWAP DO DROP 1+ DUP 1-
  C@ I C@ - DUP IF DUP ABS / LEAVE THEN LOOP SWAP DROP ;
: -ROT ROT ROT ;
: -DOUBLE ( a1 a2 -- n) \ works like -TEXT for double #s
  2@ ROT 2@ 2SWAP D- 2DUP D0=
  IF 0 ELSE 2DUP .0 D> IF 1 ELSE -1 THEN THEN
  >R 2DROP R> ;
: ARRAY CREATE 2* ALLOT DOES> SWAP 2* + ;
: IF-NOT COMPILE 0= [COMPILE] IF ; IMMEDIATE
: WHILE-NOT COMPILE 0= [COMPILE] WHILE ; IMMEDIATE
```

#### Screen # 12

```
\ FILE words elp 03sep85
: LASTREC 0 RECORD ; \ length of file
: #ACTIVE 0 RECORD 2+ ; \ # of records not marked by REMOVE
: FILE
  CREATE , \ starting block in file
  1+ , \ maximum number of records in file
  DUP B/BUF OVER / * , \ # bytes / block
  , 0 , 0 , \ bytes / record , current rec #, and
  \ adr of field-list
  DOES> 'OPEN ! ;
VARIABLE 'FIELD \ points to current field
: FIELD \ usage: ALPHA 0 20 FIELD NAME
  CREATE , ( length ) , ( offset ) , ( type )
  DOES> DUP 'FIELD ! 2+ @ ADDRESS + ;
: FLD-WIDTH ( -- n) 'FIELD @ @ ;
```

★ ★ SEE OUR HOLIDAY SPECIALS ★ ★

# FORTH INTEREST GROUP MAIL ORDER FORM

P.O. Box 8231 San Jose, CA 95155 (408) 277-0668

## MEMBERSHIP

### IN THE FORTH INTEREST GROUP

**108 - MEMBERSHIP** in the FORTH INTEREST GROUP & Volume 8 of FORTH DIMENSIONS. No sales tax, handling fee or discount on membership. See the back page of this order form.

The Forth Interest Group is a worldwide non-profit member-supported organization with over 4,000 members and 90 chapters. FIG membership includes a subscription to the bi-monthly publication, FORTH Dimensions. FIG also offers its members publication discounts, group health and life insurance, an on-line data base, a large selection of Forth literature, and many other services. Cost is \$30.00 per year for USA, Canada & Mexico; all

other countries may select surface (\$37.00) or air (\$43.00) delivery.

The annual membership dues are based on the membership year, which runs from May 1 to April 30.

When you join, you will receive issues that have already been circulated for the current volume of Forth Dimensions and subsequent issues will be mailed to you as they are published.

You will also receive a membership card and number which entitles you to a 10% discount on publications from FIG. Your member number will be required to receive the discount, so keep it handy.

### HOW TO USE THIS FORM

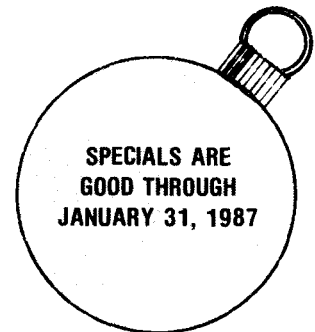
1. Each item you wish to order lists three different Price categories:

- Column 1 - USA, Canada, Mexico
- Column 2 - Foreign Surface Mail
- Column 3 - Foreign Air Mail

2. Select the item and note your price in the space provided.

3. After completing your selections enter your order on the fourth page of this form.

4. Detach the form and return it with your payment to the **Forth Interest Group**.



### FORTH DIMENSIONS BACK VOLUMES

The six issues of the volume year (May — April)

- 101** - Volume 1 FORTH Dimensions (1979/80) \$15/16/18 \_\_\_\_\_
- 102** - Volume 2 FORTH Dimensions (1980/81) \$15/16/18 \_\_\_\_\_
- 103** - Volume 3 FORTH Dimensions (1981/82) \$15/16/18 \_\_\_\_\_
- 104** - Volume 4 FORTH Dimensions (1982/83) \$15/16/18 \_\_\_\_\_
- 105** - Volume 5 FORTH Dimensions (1984/85) \$15/16/18 \_\_\_\_\_
- 106** - Volume 6 FORTH Dimensions (1983/84) \$15/16/18 \_\_\_\_\_
- 107** - Volume 7 FORTH Dimensions (1985/86) \$20/21/24 \_\_\_\_\_

**ALL 7 VOLUMES \$75.00**

**SAVE \$35.00**

### FORML CONFERENCE PROCEEDINGS

FORML PROCEEDINGS — FORML (the Forth Modification Laboratory) is an informal forum for sharing and discussing new or unproven proposals intended to benefit Forth. Proceedings are a compilation of papers and abstracts presented at the annual conference. FORML is part of the Forth Interest Group.

- 310** - FORML PROCEEDINGS 1980 . . . . \$30/33/40 \_\_\_\_\_  
Technical papers on the Forth language and extensions.

- 311** - FORML PROCEEDINGS 1981 . . . . \$45/48/55 \_\_\_\_\_  
Nucleus layer, interactive layer, extensible layer, metacompilation, system development, file systems, other languages, other operating systems, applications and abstracts without papers.
- 312** - FORML PROCEEDINGS 1982 . . . . \$30/33/40 \_\_\_\_\_  
Forth machine topics, implementation topics, vectored execution, system development, file systems and languages, applications.
- 313** - FORML PROCEEDINGS 1983 . . . . \$30/33/40 \_\_\_\_\_  
Forth in hardware, Forth implementations, future strategy, programming techniques, arithmetic & floating point, file systems, coding conventions, functional programming applications.
- 314** - FORML PROCEEDINGS 1984 . . . . \$30/33/40 \_\_\_\_\_  
Expert systems in Forth, using Forth, philosophy, implementing Forth systems, new directions for Forth, interfacing Forth to operating systems, Forth systems techniques, adding local variables to Forth.
- 315** - FORML PROCEEDINGS 1985 . . . . \$35/38/45 \_\_\_\_\_  
Also includes papers from the 1985 euroFORML Conference. Applications: expert systems, data collection, networks. Languages: LISP, LOGO, Prolog, BNF. Style: coding conventions, phrasing. Software Tools: compilers, structure charts. Forth internals: Forth computers, floating point, interrupts, multitasking, error handling.

★ FORML SPECIAL \$150 FOR ALL 6 . . . SAVE \$50.00 ★

## BOOKS ABOUT FORTH

- 200** - ALL ABOUT FORTH ..... \$25/26/35 \_\_\_\_\_  
Glen B. Haydon  
An annotated glossary for MVP Forth; a 79-Standard Forth.
- 216** - DESIGNING & PROGRAMMING  
**N** PERSONAL EXPERT SYSTEMS ... \$19/20/29 \_\_\_\_\_  
**E** Carl Townsend & Dennis Feucht  
**W** Introductory explanation of AI-Expert System Concepts. Create your own expert system in Forth. Written in 83-Standard.
- 217** - F83 SOURCE ..... \$25/26/35 \_\_\_\_\_  
**N** Henry Laxen & Michael Perry  
**E** A complete listing of F83 including source and shadow screens. Includes introduction on getting started.
- 218** - FOOTSTEPS IN AN EMPTY VALLEY  
**N** (NC4000 Single Chip Forth Engine) \$25/26/35 \_\_\_\_\_  
**E** Dr. C. H. Ting  
**W** A thorough examination and explanation of the NC4000 Forth chip including the complete source to cmForth from Charles Moore.
- 219** - FORTH: A TEXT AND REFERENCE \$22/23/33 \_\_\_\_\_  
**N** Mahlon G. Kelly & Nicholas Spies  
**E** A text book approach to Forth with comprehensive references to MMS Forth and the 79 and 83 Forth Standards.
- 220** - FORTH ENCYCLOPEDIA ..... \$25/26/35 \_\_\_\_\_  
Mitch Derick & Linda Baker  
A detailed look at each fig-Forth instruction.
- 225** - FORTH FUNDAMENTALS, V.1 ... \$16/17/20 \_\_\_\_\_  
Kevin McCabe  
A textbook approach to 79-Standard Forth
- 230** - FORTH FUNDAMENTALS, V.2 ... \$13/14/18 \_\_\_\_\_  
Kevin McCabe  
A glossary.
- 232** - FORTH NOTEBOOK ..... \$25/26/35 \_\_\_\_\_  
Dr. C. H. Ting  
Good examples and applications. Great learning aid. PolyFORTH is the dialect used. Some conversion advice is included. Code is well documented.
- 233** - FORTH TOOLS ..... \$22/23/32 \_\_\_\_\_  
Gary Feierbach & Paul Thomas  
The standard tools required to create and debug Forth-based applications.
- 235** - INSIDE F-83 ..... \$25/26/35 \_\_\_\_\_  
Dr. C. H. Ting  
Invaluable for those using F-83.
- 237** - LEARNING FORTH ..... \$17/18/27 \_\_\_\_\_  
Margaret A. Armstrong  
Interactive text, introduction to the basic concepts of Forth. Includes section on how to teach children Forth.
- 240** - MASTERING FORTH ..... \$18/19/22 \_\_\_\_\_  
Anita Anderson & Martin Tracy  
A step-by-step tutorial including each of the commands of the Forth-83 International Standard; with utilities, extensions and numerous examples.
- 245** - STARTING FORTH (soft cover) ... \$22/23/32 \_\_\_\_\_  
Leo Brodie  
A lively and highly readable introduction with exercises.
- 246** - STARTING FORTH (hard cover) .. \$20/21/30 \_\_\_\_\_  
Leo Brodie
- 255** - THINKING FORTH (soft cover) ... \$16/17/20 \_\_\_\_\_  
Leo Brodie  
The sequel to "Starting Forth". An intermediate text on style and form.
- 265** - THREADED INTERPRETIVE LANGUAGES ... \$25/26/35 \_\_\_\_\_  
R. G. Loelinger  
Step-by-step development of a non-standard Z-80 Forth.
- 270** - UNDERSTANDING FORTH ..... \$3.50/5/6 \_\_\_\_\_  
Joseph Reymann  
A brief introduction to Forth and overview of its structure.

## ROCHESTER PROCEEDINGS

The Institute for Applied Forth Research, Inc. is a non-profit organization which supports and promotes the application of Forth. It sponsors the annual Rochester Forth Conference.

- 321** - ROCHESTER 1981  
(Standards Conference) ..... \$25/28/35 \_\_\_\_\_  
79-Standard, implementing Forth, data structures, vocabularies, applications and working group reports.
- 322** - ROCHESTER 1982  
(Data bases & Process Control) ... \$25/28/35 \_\_\_\_\_  
Machine independence, project management, data structures, mathematics and working group reports.
- 323** - ROCHESTER 1983  
(Forth Applications) ..... \$25/28/35 \_\_\_\_\_  
Forth in robotics, graphics, high-speed data acquisition, real-time problems, file management, Forth-like languages, new techniques for implementing Forth and working group reports.
- 324** - ROCHESTER 1984  
(Forth Applications) ..... \$25/28/35 \_\_\_\_\_  
Forth in image analysis, operating systems, Forth chips, functional programming, real-time applications, cross-compilation, multi-tasking, new techniques and working group reports.
- 325** - ROCHESTER 1985  
(Software Management & Engineering) \$20/21/30 \_\_\_\_\_  
Improving software productivity, using Forth in a space shuttle experiment, automation of an airport, development of MAGIC/L, and a Forth-based business applications language; includes working group reports.

## THE JOURNAL OF FORTH APPLICATION & RESEARCH

A refereed technical journal published by the Institute for Applied Forth Research, Inc.

- 401** - JOURNAL OF FORTH RESEARCH V.1  
Robotics/Data Structures ..... \$30/33/38 \_\_\_\_\_
- 403** - JOURNAL OF FORTH RESEARCH V.2 #1  
Forth Machines. .... \$15/16/18 \_\_\_\_\_
- 404** - JOURNAL OF FORTH RESEARCH V.2 #2  
Real-Time Systems. .... \$15/16/18 \_\_\_\_\_
- 405** - JOURNAL OF FORTH RESEARCH V.2 #3  
Enhancing Forth. .... \$15/16/18 \_\_\_\_\_
- 406** - JOURNAL OF FORTH RESEARCH V.2 #4  
Extended Addressing. .... \$15/16/18 \_\_\_\_\_
- 407** - JOURNAL OF FORTH RESEARCH V.3 #1  
Forth-based laboratory systems and data structures.  
..... \$15/16/18 \_\_\_\_\_
- 409** - JOURNAL OF FORTH RESEARCH V.3 #3  
..... \$15/16/18 \_\_\_\_\_
- 410** - JOURNAL OF FORTH RESEARCH V.3 #4  
..... \$15/16/18 \_\_\_\_\_

## REPRINTS

- 420** - BYTE REPRINTS ..... \$5/6/7 \_\_\_\_\_  
Eleven Forth articles and letters to the editor that have appeared in *Byte Magazine*.

---

## DR. DOBB'S JOURNAL

This magazine produces an annual special Forth issue which includes source-code listing for various Forth applications.

422 -DR. DOBB'S 9/82	.....	\$5/6/7	_____
423 -DR. DOBB'S 9/83	.....	\$5/6/7	_____
424 -DR. DOBB'S 9/84	.....	\$5/6/7	_____
425 -DR. DOBB'S 10/85	.....	\$5/6/7	_____
426 -DR. DOBB'S 7/86	.....	\$5/6/7	_____

**ALL 5 VOLUMES \$15.00 ... SAVE \$10.00**

---

## HISTORICAL DOCUMENTS

- 501 -KITT PEAK PRIMER ..... \$25/27/35 \_\_\_\_\_  
One of the first institutional books on Forth. Of historical interest.
- 502 -Fig-FORTH INSTALLATION MANUAL \$15/16/18 \_\_\_\_\_  
Glossary model editor — We recommend you purchase this manual when purchasing the source-code listing.
- 503 -USING FORTH ..... \$20/21/22 \_\_\_\_\_  
FORTH, Inc.
- 

## REFERENCE

- 305 -FORTH 83-STANDARD ..... \$15/16/18 \_\_\_\_\_  
The authoritative description of 83-Standard Forth. For reference, not instruction.
- 300 -FORTH 79-STANDARD ..... \$15/16/18 \_\_\_\_\_  
The authoritative description of 79-Standard Forth. Of historical interest.

**BOTH FOR \$25.00**

---

## ASSEMBLY LANGUAGE SOURCE CODE LISTINGS

Assembly Language Source Listings of fig-Forth for Specific CPUs and machines with compiler security and variable length names.

514 -6502/SEPT 80	.....	\$15/16/18	_____
515 -6800/MAY 79	.....	\$15/16/18	_____
516 -6809/JUNE 80	.....	\$15/16/18	_____
517 -8080/SEPT 79	.....	\$15/16/18	_____
518 -8086/88/MARCH 81	.....	\$15/16/18	_____
519 -9900/MARCH 81	.....	\$15/16/18	_____
521 -APPLE II/AUG 81	.....	\$15/16/18	_____
523 -IBM-PC/MARCH 84	.....	\$15/16/18	_____
526 -PDP-11/JAN 80	.....	\$15/16/18	_____
527 -VAX/OCT 82	.....	\$15/16/18	_____
528 -Z80/SEPT 82	.....	\$15/16/18	_____

**\$10.00  
EACH**

---

## MISCELLANEOUS

- 601 -T-SHIRT SIZE \_\_\_\_\_  
Small, Medium, Large and Extra-Large.  
White design on a dark blue shirt. . . \$10/11/12 \_\_\_\_\_
- 602 -POSTER (BYTE Cover) ..... \$5/6/7 \_\_\_\_\_
- 616 -HANDY REFERENCE CARD ..... FREE \_\_\_\_\_
- 683 -FORTH-83 HANDY REFERENCE CARD ... FREE \_\_\_\_\_
- 

## FORTH MODEL LIBRARY

The model applications disks described below are new additions to the Forth Interest Group's library. These disks are the first releases of new professionally developed Forth applications disks. Prepared on 5 1/4" disks, they are IBM MSDOS 2.0 and up compatible. The disks are compatible with Forth-83 systems currently available from several Forth vendors. Macintosh 3 1/2" disks are available for MasterFORTH systems only.

### Forth-83 Compatibility IBM MSDOS

Laxen/Perry F83	LMI PC/FORTH 3.0
MasterFORTH 1.0	TaskFORTH 1.0
PolyFORTH® II	

### Forth-83 Compatibility Macintosh

MasterFORTH

---

## ORDERING INFORMATION

- 701 -A FORTH LIST HANDLER V.1 ..... \$40/43/45 \_\_\_\_\_  
by Martin J. Tracy  
Forth is extended with list primitives to provide a flexible high-speed environment for artificial intelligence. ELISA and Winston & Horn's micro-LISP are included as examples. Documentation is included on the disk.
- 702 -A FORTH SPREADSHEET V.2 ..... \$40/43/45 \_\_\_\_\_  
by Craig A. Lindley  
This model spreadsheet first appeared in Forth Dimensions Volume 7, Issue 1 and 2. These issues contain the documentation for this disk.
- 703 -AUTOMATIC STRUCTURE CHARTS V.3 \$40/43/45 \_\_\_\_\_  
by Kim R. Harris  
These tools for the analysis of large Forth programs were first presented at the 1985 FORML conference. Program documentation is contained in the 1985 FORML Proceedings.

**Please specify disk size when ordering** .....

---

- 701 -A FORTH LIST HANDLER V.1 ..... \$35.00 \_\_\_\_\_
- 702 -A FORTH SPREADSHEET V.2 ..... \$25.00 \_\_\_\_\_
- 703 -AUTOMATIC STRUCTURE CHARTS V.3 \$25.00 \_\_\_\_\_

# HOLIDAY SPECIALS !!





**Screen # 13**

```

\ Tables of type dependent functions      elp 03sep85
0 CONSTANT ALPHA      \ offset into tables of type dependent
2 CONSTANT SINGLE     \ functions
4 CONSTANT DOUBLE
6 CONSTANT $$
: FLD-TYPE ( --n) 'FIELD @ 4 + @ ;
: TABLE : DOES> FLD-TYPE + @ EXECUTE ;
: PAD>NUM ( -- d) PAD 1- FLD-WIDTH OVER C! NUMBER ;
: T-! ( adr --) PAD SWAP FLD-WIDTH CMOVE ;
: S-! ( adr --) PAD>NUM DROP SWAP ! ;
: D-! ( adr --) PAD>NUM ROT 2! ;
TABLE (ENTER) T-! S-! D-! D-! ; \ store field entry

```

**Screen # 14**

```

\ TABLES of type dependent functions    elp 03sep85
: $FORMAT ( d -- adr u) DUP >R DABS.<# # # ASCII . HOLD #S
R> SIGN #> ;
: T-TYPE ( adr --) FLD-WIDTH TYPE ;
: S-TYPE ( adr --) @ FLD-WIDTH .R ;
: D-TYPE ( adr --) 2@ FLD-WIDTH D.R ;
: $-TYPE ( adr --) 2@ $FORMAT FLD-WIDTH DUP ROT -
SPACES TYPE ;
TABLE DISPLAY T-TYPE S-TYPE D-TYPE $-TYPE ;

```

**Screen # 15**

```

\ TABLES cont'd                          elp 03sep85
: T-COMPARE ( a1 a2 -- n) FLD-WIDTH SWAP -TEXT ;
: S-COMPARE ( a1 a2 -- n) SWAP @ SWAP @ - ;
: D-COMPARE ( a1 a2 -- n) -DOUBLE ;
TABLE COMPARE T-COMPARE S-COMPARE D-COMPARE D-COMPARE ;

: GR.THAN COMPARE 0> ;
: LS.THAN COMPARE 0< ;
: IS COMPARE 0= ;
: ISNT COMPARE ;
\ Record display words
: .FIELD-NAME 'FIELD @ BODY> >NAME .NAME ASCII : EMIT ;
: .FIELD DISPLAY ;
: .LINE CR .FIELD-NAME SPACE DISPLAY ;
: .RECORD FIELD-LIST @ BEGIN DUP @ ?DUP WHILE EXECUTE
.LINE 2+ REPEAT DROP ;

```

**Screen # 16**

```

\ FILE input words                          elp 03sep85
: DASHES ( n --) SPACE DUP 0 DO 95 EMIT LOOP 0 DO 8 EMIT
LOOP ; \ use for input prompt
: INPUT QUERY BL TEXT ;
: .PROMPT CR .FIELD-NAME SPACE FLD-WIDTH DASHES ;
: ENTER \ prompts, accepts and stores field entries
.PROMPT INPUT (ENTER) UPDATE ;
\ Query words
: REMOVED? ( rec# -- ?) RECCRD C@ ASCII * = ;
: #ARGS ( --n) \ count arguments in command line
>IN @ 0 BEGIN BL WORD C@ WHILE 1+ REPEAT SWAP
>IN ! ;
DOER .DISPLAY
DOER DELAY
DOER HEADING

```

**Screen # 17**

```

\ File modification words
: SIGNAL 7 EMIT CR COUNT TYPE ." is not a valid field" ;
: CHANGE
BEGIN CR ." Enter name of field to be changed" CR QUERY
BL WORD FIND WHILE-NOT SIGNAL REPEAT EXECUTE ENTER ;
: REMOVE
ASCII * ADDRESS C! UPDATE -1 #ACTIVE +! UPDATE ;

: MODIFY
CR ." Enter C to change or R to remove record " KEY
DUP ASCII C = IF DROP CHANGE ELSE ASCII R =
IF REMOVE THEN THEN DELAY ;

```

**Screen # 18**

```

\ Query set-up words
4 CONSTANT Q# \ max# query conditions
VARIABLE #HITS \ # of records found by query
Q# ARRAY LOGICALS
Q# ARRAY OPERANDS
Q# ARRAY CONDITIONS
: TARGETS HERE 200 + ;
: +TARGET ( i --) 30 * TARGETS + ;
: T-BRING ( a --) TEXT PAD SWAP FLD-WIDTH CMOVE ;
: 1-BRING ( a --) WORD NUMBER DROP SWAP ! ;
: 2-BRING ( a --) WORD NUMBER ROT 2! ;
TABLE BRING T-BRING 1-BRING 2-BRING 2-BRING ;
: GET-TARGET ( i --) +TARGET BL BRING ;

```

**Screen # 19**

```

\ QUERY words                               elp 03sep85
: Q-ARRAYS ( n --) ['] NOTHING 0 LOGICALS !
  0 DO I IF ' I LOGICALS ! THEN
  ' DUP I OPERANDS ! >BODY 'FIELD !
  ' I CONDITIONS ! I GET-TARGET LOOP ;
: LOGIC ( i --) LOGICALS @ EXECUTE ;
: OPERAND ( i --) OPERANDS @ EXECUTE ;
: CONDITION ( i --) CONDITIONS @ EXECUTE ;
: TARGET +TARGET ;
: FOUND? ( n -- f) 0 DO I OPERAND I TARGET I CONDITION
  I LOGIC LOOP ;
: FROM ' ( filename) EXECUTE ;
: (FIND) ( n --) 0 #HITS ! LASTREC @ 1+ 1
  DO I REMOVED? IF-NOT I REC# ! ( n) DUP FOUND?
  IF 1 #HITS +! CR .DISPLAY CR DELAY THEN THEN
  LOOP DROP #HITS @ IF-NOT CR ." search failed " THEN ;

```

**Screen # 21**

```

\ FILE entry words                           elp 03sep85
: NEWFILE FROM 0 #ACTIVE ! 0 LASTREC ! ;
: FREE ( -- rec#) LASTREC @ 1+ 1 DO I REMOVED?
  IF I LEAVE THEN LOOP ;
: NEXTREC ( -- rec#) LASTREC @ #ACTIVE @ >
  IF FREE REC# ! ADDRESS REC-LEN BL FILL UPDATE
  ELSE LASTREC DUP @ 1+ DUP READ SWAP ! UPDATE THEN ;
: WRITE
  FIELD-LIST @ BEGIN DUP @ ?DUP WHILE EXECUTE ENTER 2+
  REPEAT DROP ;
: 3DOWN CR CR CR ;
: ENTRY #ARGS 1 (>) ABORT" needs filename " FROM
  BEGIN CLEARSCREEN 3DOWN NEXTREC WRITE 1 #ACTIVE +!
  UPDATE 3DOWN DONE? UNTIL
  SAVE-BUFFERS ;

```

**Screen # 23**

```

\ file display words                         elp 12sep85
: .EXCERPTS EXCERPTS BEGIN DUP @ ?DUP WHILE EXECUTE
  .FIELD SPREAD 2+ REPEAT DROP ;
: SELECT \ usage: SELECT <filename> <field1> ...<fieldn>
  FROM EXCERPTS #ARGS DUP 5 > ABORT" too many " 0
  DO ' OVER ! 2+ LOOP 0 SWAP ! MAKE DELAY NOOP ;AND
  MAKE .DISPLAY .EXCERPTS ;AND MAKE HEADING .HEADER ;
: FIELDS \ usage: filename n FIELDS field1 field2 field3 ....
  HERE SWAP 0 DO ' , LOOP 0 , FIELD-LIST ! ;
: .MSSG CR ." RETURN to quit ESC to modify" CR
  ." any key to continue" CR ;
: STEP MAKE DELAY .MSSG KEY DUP 27 = IF DROP MODIFY
  ELSE 13 = IF CR ." query aborted " ABORT THEN THEN
  ;AND MAKE .DISPLAY .RECORD ;AND MAKE HEADING NOTHING ;
STEP \ default display mode

```

**Screen # 20**

```

\ QUERY words                               elp 03sep85
: FIND \ end user query word
  FROM
  #ARGS 1+ 4 /MOD SWAP ABORT" incorrect # of arguments"
  DUP 0# > ABORT" incorrect # of arguments"
  CR HEADING DUP Q-ARRAYS (FIND) ;
  \ usage: FIND EMPLOYEE DEPT IS PARTS AND HOURS GR.THAN 40
\ Other words
: DONE? ( -- t=no-more=entries)
  CR ." any more? Y/N " KEY DUP EMIT ASCII N = ;

```

**Screen # 22**

```

\ Display header
VARIABLE EXCERPTS 12 ALLOT \ points to field to be displayed
: DASH-LINE CR 72 0 DO ASCII - EMIT LOOP CR ;
: .HEADER EXCERPTS BEGIN DUP @ ?DUP WHILE DUP >BODY
  'FIELD ! BODY> >NAME DUP .NAME C@ 31 AND FLD-WIDTH
  SWAP - ABS 1+ SPACES 2+ REPEAT DROP DASH-LINE CR ;
: SPREAD FLD-TYPE IF 'FIELD @ BODY> >NAME C@ 31 AND
  FLD-WIDTH SWAP - ABS FLD-WIDTH + SPACES ELSE 2 SPACES
  THEN ;

```

**Screen # 24**

```

\ Application file and field definitions
64 100 30 FILE EMPLOYEES
ALPHA 1 20 FIELD NAME
$$ 21 6 FIELD HOURLY-RATE
SINGLE 25 2 FIELD HOURS
ALPHA 27 6 FIELD DEPT
EMPLOYEES 4 FIELDS NAME HOURLY-RATE HOURS DEPT

```



# NGS FORTH

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

## STANDARD FEATURES INCLUDE:

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

A COMPLETE FORTH  
DEVELOPMENT SYSTEM.

PRICES START AT \$70

NEW ◀ HP-150 & HP-110  
VERSIONS AVAILABLE



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

record has been deleted. If it hasn't, it is checked to see if it matches the conditions specified in the query command line.

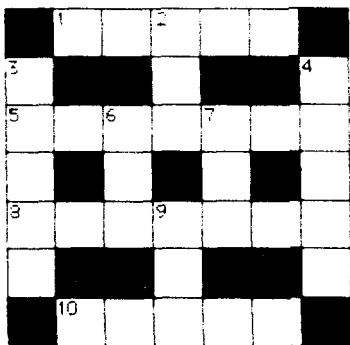
**FIND** End-user query word. Checks to see if an incorrect number of arguments has been entered in the query command line.

**NEXTREC** If the number of active records is less than **LASTREC**, the first deleted record (found by **FREE**) is used for the next entry. If there are no deleted records, the file is extended one more record.

**WRITE** Goes through the list of fields for the current file, prompting and accepting entries.

**ENTRY** A generic entry word for all files defined by **FILE**. The fields must be included in the field list (**FIELDS**).

**EXCERPTS** Address of start of the list of fields chosen by **SELECT** to be displayed.

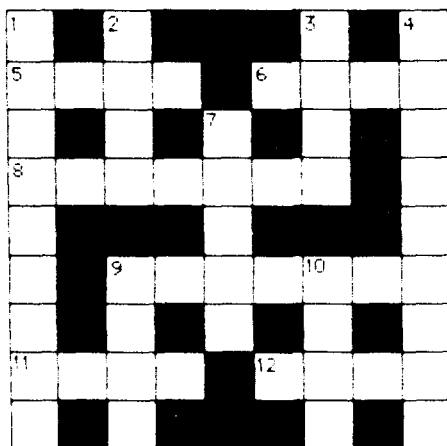


### Across

1. A process control language
5. What a computer does
8. Pertaining to metal men
10. Changes in the flow of a program

### Down

2. What a bad program should do: abbr.
3. Liked by squirrels
4. Remove solid H<sub>2</sub>O
6. Programmers in a frenzy
7. Type of transistor
9. Measure of resistance



### Across

5. Exchange
6. Average
8. Make bigger
9. Indicates an address
11. Not odd
12. A computer noise

### Down

1. Creates machine code
2. What a programmer never wants to do
3. Give out
4. What stops the processor
7. Character \_\_\_\_\_
9. Look at memory location
10. Layer

—Rick Watson

(Answers on page 31.)

---

# A Forth Standard?

Glen B. Haydon  
La Honda, California

What is a standard language? Natural languages evolve. Only after a word is used with a specific meaning for a period of time do dictionary editors consider including it. Many words have multiple meanings. Many definitions include examples of their use. Some words become obsolete or archaic. Languages are dynamic. They cannot be set in concrete. There is no such thing as "standard language." Dictionaries only record current usage.

Forth does not differ from any other language. It is evolving. That is the way Charles Moore designed it. He changed his kernel and application utilities almost daily. Many of you are aware that he includes a meta-compiler with most of his applications so he can easily recompile his kernel. It will be interesting to see what direction he takes now that he has cast his kernel in the Novix 4000 chip.

Before going any further, I would like to make a distinction between a kernel and a functional language. The Forth kernel is, in essence, the emulation of a hardware processor. The Novix 4000 is the implementation of a kernel in hardware. On the other hand, Forth as a functional language is built upon a kernel. It utilizes its extensibility to develop an operating system, compiler directives and utilities to solve problems. The functional language is a bridge between application requirements and the kernel. The beauty of Forth is the ease with which the necessary and sufficient functions can be added to a kernel.

The kernel usually includes between sixty and seventy hardware-related functions. There is little problem identifying these, but in actual hardware it has become obvious that some of the emulated functions are not optimal. Some of the problems were not anticipated by anyone.

The best example of a problem is the **DO LOOP** structure. The original fig-FORTH implementation requires a range in reverse order. What did the emulation do when a range crossed the

boundary of a signed number? Considerable error checking was added to the **LOOP** function in the 79-Standard definition. This proved to be a real boat anchor for speed nuts. This problem was addressed again in the 83-Standard and was improved. In the Novix 4000 the function was replaced by **FOR NEXT**. This function takes a count and decrements it to zero. The hardware requirements for speed dictated that a count-down register would work better and faster. Now the higher-level **DO LOOP** function becomes a part of the functional language, if it is going to be used. So the language changes.

With any Forth kernel, in hardware or emulated, it is an easy job to implement any desired dialect of functional Forth. Each vendor has his own idea of what should be included and what should be excluded. Each vendor provides a slightly different dialect of Forth. Most vendors make their kernel and the basic part of their functional Forth proprietary.

Let us review the public-domain versions of the primitive Forth functions. I started with the first public-domain version readily available — the fig-FORTH Model. The installation manual provided a verbal definition, and the several implementations clarified any possible misunderstandings. The system worked well. I did a moderate amount of programming with it.

Then came the 79-Standard. This was the result of about twenty Forth programmers who addressed some of the "problems" of the fig-FORTH Model. They did several things.

First, they changed the functional definitions for forty words previously defined in the fig-FORTH Model. Some of the changes were simply the use of an alias for the same function. Other changes were of a minor nature. The improvement to the compiler directive **CREATE DOES>** was perhaps the most significant. The ability to write special compiler directives as part of an application program is unique to Forth among computer languages.

Second, the 79-Standard went beyond these functional changes. It in-

cluded a list of additional "Requirements" for any program adhering to the 79-Standard. In the Standard publication under Section 8, "Use":

*"A Forth Standard program may reference only the definitions of the Required Word Set, and definitions which are subsequently defined in terms of these words . . ."*

This is patently ridiculous. At the November 1981 FORML Conference, I had an implementation of Forth which contained only the 148 words in the required word set. None of the members of the Standards Team who were there could do anything with the program. No vendor I know of has built a product in complete conformity with the restrictions imposed by the 79-Standard document.

About this same time, Robert L. Smith released and copyrighted a Forth-79 Standard Conversion. This publication consisted of a series of screens which could be loaded on a fig-FORTH Model. They would redefine the necessary forty words in the required word set. He admonishes the user to meet the other requirements of the 79-Standard.

Instead of conversion screens, I modified the compiler source code for the fig-FORTH Model to conform with the 79-Standard Required Word Set and made the additional functions required for a headerless operating system. This was a simple matter of changing a flag for the cross-compiler. I must acknowledge the efforts of Jerry Boutelle, who adapted his cross-compiler for the job and added many of the features. In a period of months two revisions were made. The resulting MVP-FORTH has remained stable for four years! The glossary *All About Forth* provides a reference to the common functions in public-domain implementations of FORTH up to that time.

Added to the MVP-FORTH kernel are a number of utilities and some supplemental definitions that will make this functional Forth almost completely compatible with Leo Brodie's *Starting Forth*. The differences are related to his use of a proprietary product (poly-

FORTH) which was supposed to be 79-Standard. Alan Winfield's *The Complete Forth* provides an excellent alternative tutorial.

Copyright protection of software is a continuing problem. The spirit of fig-FORTH was to put all of the source code and documentation in the public domain, asking only for appropriate acknowledgment. MVP-FORTH adopted the same spirit and placed all of the basic source code and documentation in the public domain. The contents of Volume 1 in the MVP-FORTH Series, *All About Forth*, are released without restrictions. Each entry includes a functional definition, indicates the source, an implementation, the usage in the MVP-FORTH kernel, an example with a note and a general comment. The general comment includes known differences in function among dialects.

As an interesting aside concerning the significance of copyrights, we had some correspondence with the publisher of *Starting Forth*. They claimed they had a copyright on all of the functional definitions included in their book. They claimed we could not include any of their functional definitions in *All About Forth*. I made an exhaustive study of prior functional definitions of the same words and was able to cite at least one prior definition for each word. Some of those prior definitions were also copyrighted and the publisher had failed to secure a proper release. So much for copyrights.

Other vendors approached the 79-Standard in various ways. Generally, their documentation has been excellent. I have always felt that the more implementations of Forth there are available, the more Forth will be used. By the time these products were on the market, the Standards Team was at it again and came out with the 83-Standard. In my opinion, this was a great disservice to the advancement of Forth.

When the 83-Standard was first available, I made a very careful comparison of the new functional definitions of the Required Word Set with those in the 79-Standard. The number of required words was reduced from 148 to 132. All but five had some

change in the functional definitions. No implementations were included as in the original fig-FORTH Model. In fact, some of the adopted functions had never been tested by the team.

In fairness to the members of the Standards Team, they are a dedicated group whose sole objective has been to improve and advance Forth. Many of the changes I found were simply attempts to clarify the wording of the previous standard.

However, they saw fit to change the functional definition of some words without changing the names. **PICK** and **ROLL** are examples. They required that the value on the stack be decreased by one from the value according to the 79-Standard. Thus:

: ROT 3 ROLL ; ( 79-Standard )  
: ROT 2 ROLL ; ( 83-Standard )

When you know of this incompatibility, it is easy to go through your code and change all the values to make it function. But I can see no improvement. Once a convention is adopted, stay with it.

I have no inclination to go through such a careful comparison again. Most of the changes made little difference. However, as has been observed by members of the Standards Team, most people don't do floored division. Forth has enough problems as it is. Why add to them with obscure changes? Forth needs stability.

In addition to the changes in the Required Word Set, similar requirements to those cited above in the 79-Standard are included in the 1983 document. There is no way to verify the compliance of the many systems purporting now to be 83-Standard.

In the best spirit of Forth, Laxen and Perry have done an implementation of Forth which has become known as F83. It is unfortunate that this has been assumed to be the 83-Standard. It goes far beyond the 83-Standard. It includes nearly 1200 words, and contains many excellent examples of problem solving with Forth. They provide full source code and shadow screens to assist the user. Unfortunately, there is

## **DASH, FIND & ASSOCIATES**

Our company, DASH, FIND & ASSOCIATES, is in the business of placing FORTH Programmers in positions suited to their capabilities.

We deal only with FORTH Programmers and companies using FORTH. If you would like to have your resumé included in our data base, or if you are looking for a FORTH Programmer, contact us or send your resumé to:

DASH, FIND & ASSOCIATES  
808 Dalworth, Suite B  
Grand Prairie TX 75050  
(214) 642-5495



**Committed to Excellence**



### **FIG-FORTH for the Compaq,**

**IBM-PC, and compatibles. \$35**  
Operates under DOS 2.0 or later,  
uses standard DOS files.

Full-screen editor uses 16 x 64  
format.

Editor Help screen can be called  
up using a single keystroke.

Source included for the editor  
and other utilities.

Save capability allows storing  
Forth with all currently defined  
words onto disk as a .COM file.

Definitions are provided to allow  
beginners to use *Starting Forth*  
as an introductory text.

Source code is available as an  
option, add \$20.

#### **Async Line Monitor**

Use Compaq to capture,  
display, search, print, and  
save async data at 75-19.2k  
baud. Menu driven with  
extensive Help. Requires two  
async ports. **\$300**

**A Metacompiler on a**  
host PC, produces a PROM  
for a target 6303/6803  
Includes source for 6303  
FIG-Forth. Application code  
can be Metacompiled with  
Forth to produce a target  
application PROM **\$280**

**FIG-Forth in a 2764 PROM**  
for the 6303 as produced by  
the above Metacompiler.  
Includes a 6 screen RAM-Disk  
for stand-alone operation. **\$45**

**An all CMOS processor**  
board utilizing the 6303.  
Size: 3.93 x 6.75 inches.  
Uses 11-25 volts at 12ma,  
plus current required for  
options. **\$210 - \$280**

Up to 24kb memory: 2 kb to  
16kb RAM, 8k PROM contains  
Forth. Battery backup of RAM  
with off board battery.

Serial port and up to 40 pins of  
parallel I/O.

Processor buss available at  
optional header to allow expanded  
capability via user provided  
interface board.

### **Micro Computer Applications Ltd**

8 Newfield Lane  
Newtown, CT 06470  
203-426-6164

Foreign orders add \$6 shipping and handling.  
Connecticut residents add sales tax.

no tutorial such as *Starting Forth* to go along with it. Every Forth programmer should be familiar with the many techniques these master Forth programmers have used.

Among the vendors, Laboratory Microsystems, Inc. has a version which is supposed to comply with the 83-Standard. After finishing his implementation, Ray Duncan wrote a most interesting commentary on the 83-Standard which was published in *Dr. Dobb's Journal*. Other vendors have also implemented what they call 83-Standard Forth. Each of the vendors has excellent documentation for its particular implementation. A variety of other books on Forth are gradually appearing. Each is based on a specific Forth dialect, many of which are proprietary and copyrighted. However, many of the examples and ideas are portable to other Forth dialects with minimal effort. These books are a great help to the intermediate Forth programmer.

Already, some members of the Standards Team are soliciting suggestions for an 87-Standard. It is hoped that the FORML Conference this year will be able to address some of these recommendations.

I would humbly urge those interested in promoting the careful evolution of Forth to take a lesson from the pharmaceutical industry. Only after years in the chemical laboratory and more years of animal testing, are new drugs released for clinical trials. Only after all of the testing and trials have proven satisfactory are drugs finally released for general clinical use.

The Forth Modification Laboratory, FORML, is a fitting place for the laboratory development of modifications. The modifications should first be tried in the laboratory. Favorable results from such work should be submitted to clinical trial in the hands of vendors. Only by acceptance on the part of vendors should changes to a standard be adopted. But then it will not really be necessary: the modifications will have evolved into the common base of the functional Forth language. The standard will be established by common usage.

There is a recurring question of standard libraries. If people would publish their techniques, they could be adapted into most Forth dialects. But there is a reservation on the part of many authors. They want to have some return from all of their efforts. It is only reasonable that they be rewarded for their efforts.

Mountain View Press has found a partial answer to the problem. Namely, though some of their nine volumes are copyrighted, the contents are released for non-commercial use. At least the user can learn from the examples. It is highly likely that he will want to redo any algorithm in his dialect for his own application. Certainly it is not reasonable to let others reprint a book for profit as has been done with Volume 1 of the MVP-FORTH Series.

The current edition of Volume 3 in the MVP-FORTH Series is an example of the evolution of such thinking. The original text was written more than four years ago, and has been actively used since then. In 1985, author Phil Koopman agreed to a restricted copyright releasing it for non-commercial use. Each entry is modeled on *All About Forth* and includes a functional definition, a high-level Forth implementation, an example with a note and a comment.

The local fig-FORTH community still objected: they could not use it because of the copyright, as open as it was. Some in the community have copyrighted their work and made no concessions to non-commercial use. This year, Phil Koopman released his work from copyright, with no restrictions. I hope more Forth authors will see fit to follow his example.

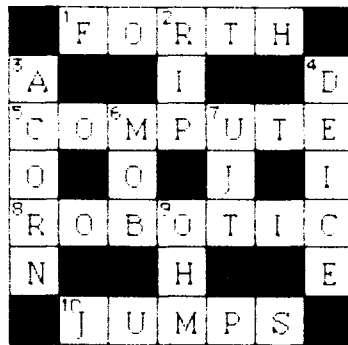
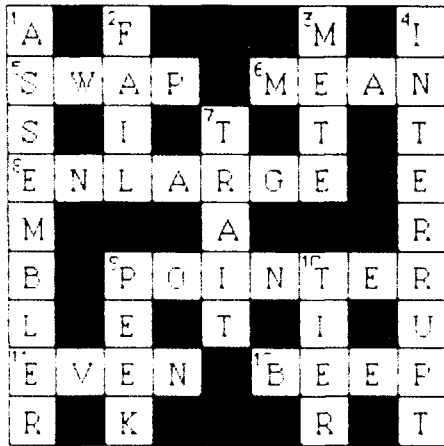
To argue about Forth standards is for those who have nothing better to do. Let Forth evolve like any natural language. Unlike other programming languages, it is easy to start over and meta-compile a new kernel. It is easy to build a new functional system.

Keep the FORML work active in the background. Encourage regional FORML workshops. As modern micro-

computers are becoming more powerful, something more than sixteen-bit address space is needed. How to incorporate this new hardware into the language presents several alternatives. None of the existing public-domain implementations address this problem. The existing standards are simply not compatible with thirty-two-bit stacks.

Don't let the existing standards be an albatross to the language.

We have an urgent need for a stable language for beginners, for the management team entering new projects and for administrators new to the language. Let common usage provide a dynamic standard to meet the evolving needs. Let everyone participate.



### Index to Advertisers

- |   |                                    |
|---|------------------------------------|
| Bryte - 6                                 | MicroMotion - 36                   |
| Computer Cowboys - 7                      | Miller Microcomputer Services - 33 |
| Dash, Find & Associates - 29              | Mountain View Press - 31           |
| Forth, Inc. - 19                          | New Micros - 16                    |
| Forth Interest Group - 11, 21-24, 44      | Next Generation Systems - 27       |
| Harvard Softworks - 35                    | Palo Alto Shipping Company - 4     |
| Insite Computing - 38                     | Software Composers - 2             |
| Institute for Applied Forth Research - 12 | SOTA - 15                          |
| Laboratory Microsystems - 8               | Talbot Microsystems - 14           |
| MCA - 30                                  | UBZ Software - 32                  |

## FORTH

The computer language for

*increased...*

**EFFICIENCY**

*reduced.....*

**MEMORY**

*higher.....*

**SPEED**

### MVP-FORTH SOFTWARE

Stable...Transportable...  
Public Domain...Tools

### MVP-FORTH PROGRAMMER'S KIT

for IBM, Apple, CP/M, MS/DOS, Amiga, Macintosh and others. Specify computer.  
**\$175**

### MVP-FORTH PADS,

a Professional Application Development System. Specify computer.  
**\$500**

### MVP-FORTH EXPERT-2 SYSTEM

for learning and developing knowledge based programs.  
**\$100**

### Word/Kalc,

a word processor and calculator system for IBM.  
**\$150**

Largest selection of FORTH books: manuals, source listings, software, development systems and expert systems.

Credit Card Order Number:  
800-321-4103  
(In California 800-468-4103)

Send for your  
FREE  
FORTH  
CATALOG

**MOUNTAIN VIEW PRESS**

PO BOX 4656  
Mountain View, CA 94040

# UBZ FORTH™

for the Amiga™

- \* FORTH-83 compatible
- \* 32 bit stack
- \* Multi-tasking
- \* Separate headers
- \* Full screen editor
- \* Assembler
- \* Amiga DOS support
- \* Intuition support
- \* ROM kernel support
- \* Graphics and sound support
- \* Complete documentation
- \* Assembler source code included
- \* Monthly newsletter

**\$85**

Shipping included  
in continental U.S.  
(Ga. residents add sales tax)

**UBZ Software**  
**(404)-948-4654**

(call anytime)  
or send check or money order to:

**UBZ Software**  
395 St. Albans Court  
Mableton, Ga. 30059

\*Amiga is a trademark for  
Commodore Computer. UBZ FORTH  
is a trademark for UBZ Software.

(Continued from page 16.)

properly. **STREAM-PROCESSOR:** could also be used to implement the function of character translation by defining a character-parsing child. Other possibilities include a string search function for source screens. Implementing all these functions is made simpler and clearer through the added functionality afforded by a well-decomposed Forth kernel.

The dictionary look-up words shown in Listing One also make effective use of dual-CFA decomposition: the failure-mode processing is factored into a child definition, which inherits a dictionary look-up function from the parent. So one word, the child definition, integrates and binds two related behaviors. While the child represents efficient factoring, the parent suggests a related family of words.

As shown in Listing One, the children of **FAILING-LOOKUP:** are **?COMPILE-NUMBER;** and **?INTERPRET-NUMBER;**. Both of these words represent incremental progress toward their parent functions, **COMPILE-WORD** and **INTERPRET-WORD**. Note also that these string-handling functions need not be expanded any further to produce a workable system (as will be shown). To expand them any further would produce undesirable crossover into the domains of other families of words.

As defined in Listing One, **STREAM-PROCESSOR:** actually combines three behaviors into each of its children. The parsing loop is inherited by the children, but it also contains a vectored execution that specifies the processing after each word is parsed. The child merely specifies the version of **WORD** to be used within the shared word-parsing loop. (See **TIB-PROCESS**, **BLK-PROCESS**, **TIB-WORD** and **BLOCK-WORD**.)

The flexibility needed to switch from compiling a word to interpreting a word at run time (and vice versa) requires the use of a vector. The left and right bracket definitions must reinitialize the vector. Since the brackets may occur amidst an input stream, the action of the children of **STREAM-PROCESSOR:** is also variable midstream. To expand Listing One to include bracket definitions, you could use:

```
: [ ( -- )
  192 STATE !
  ' COMPILE-WORD
  CFA PROCESS-WORD' 1 ;
: [ ( -- )
  0 STATE !
  ' INTERPRET-WORD
  CFA PROCESS-WORD' 1 ; IMMEDIATE
  Finally, the Forth functions normally performed by QUIT and INTERPRET can be easily constructed as a single definition:
  : INTERPRET
  [COMPILE] [
  RPI BEGIN
  CR QUERY TIB-PROCESS
  STATE @ 0= IF
  ." OK" THEN
  AGAIN ;
```

### Early Impressions

The relative newness of dual-CFA decomposition has not prevented me from forming opinions regarding its most suitable use.

I have some reservations about the implementation of deferred definitions (**DEFER:**). I prefer to see a closer relationship between the two functions bound together through dual-CFA decomposition. In **DEFER:**, the parent definition provides a compiler-extending behavior and the child definition forward references to an arbitrary function.

I favor **FAILING-NUMBER:** and **FAILING-LOOKUP:** as examples of how dual-CFA decomposition techniques should be applied. I appreciate how closely united the parent and child definitions are: the parent look-up function is made more specific by the failure mode processing provided by the child. In actual use, the child refers to both functions as if they were a single, undecomposed function. Yet because they are decomposed, you are free to define new children without restating the parent function.

The demystification of Forth would be a welcome by-product of a more clearly and more fully decomposed kernel, if one should ever find its way into widespread use. Some evidence of this can already be seen in Listing One: (1) The end-of-input-stream detection function is within the parent stream-



processing function, not hidden in a definition of **NULL**. (2) The **STATE** variable is less central to one's comprehension of Forth — the interpret and compile functions are explicitly separate, even though they still share a common word-parsing loop. (3) Words that manipulate input streams are more easily distinguished from words that perform interpreting or compiling actions.

On the other hand, programming became more difficult than before. The program code in Listing One required subtle but definite changes in my programming style. Many times, I had to abandon a particular approach in search of something more intuitively obvious. However, the development process did fine tune my perception of the problem along functional lines.

The functional areas of concern required clearer identification at the outset. Next, each of these functional areas had to be well decomposed. Finally, refinements were made so that the stack effects of all functionally related subsets of words belonging to a particular family remained consistent. The comment header shown in Listing One also helped.

Throughout development, a continual effort was necessary to prevent subsets of words from wandering into the domain of another family of words. I cannot overemphasize the point that this kind of programming demands a clearer delineation of definitions along functional lines. Hybrid words must be acknowledged before useful dual-CFA decompositions can be found (such as the effort surrounding **WORD**).

## Conclusions

The examples shown of dual-CFA decomposition have helped illustrate some of the advantages possible with this methodology (see **ITERATOR**:<sup>1</sup> as well). A summary of the advantages includes:

(1) Better organized definitions, particularly along functional lines, in-

creasing the ease with which Forth source code can be read and understood.

(2) Increased emphasis on more complete decomposition, resulting in a richer programming environment and increased productivity.

(3) Decreased likelihood of programming error and system crashes, through elimination of many environmentally dependent behaviors.

(4) Decreased need for passing flag parameters on the stack, as well as a corresponding decrease in the number of conditional-behavior words (control-flow constructs such as **IF THEN** now are factorable and need appear only once per function — even if the function is decomposed).

(5) Increased memory compactness for compiled applications.

Also, modern innovations associated with new programming languages or operating systems may be more easily implemented. Examples might include object-oriented modules, relocatable modules and "piping" capabilities for stream-processing modules. These areas are generating more and more interest lately. Dual-CFA decompositions can bring each of these areas of programming interest within closer reach.

Someday, perhaps, the Forth dictionary will be mostly a library of forms<sup>2</sup> or general algorithms, from which a programmer compiles more specific instances of each algorithm to accomplish a particular task. If this happens, each issue of *Forth Dimensions* may include many practical applications. Each would be derived easily using provisions already included in the Forth dictionary.

## References

1. Elola, Mike. "Dual-CFA Definitions," part one, *Forth Dimensions* VIII/2.
2. Luoto, Kurt. "Procedural Arguments," *Forth Dimensions* VI/2.

FOR TRS-80 MODELS 1, 3, 4, 4P  
IBM PC/XT, AT&T 6300, ETC.

## COMMERCIAL SOFTWARE DEVELOPERS and INDIVIDUAL PROGRAMMERS

appreciate MMSFORTH for its:

- Power
- Flexibility
- Compactness
- Development speed
- Execution speed
- Maintainability.

When you want to create the ultimate:

- Computer Language
- Application
- Operating System
- Utility,

## BUILD IT in

# MMSFORTH

(Unless we have it ready for you now!)

Bulk Distribution Licensing @ \$500  
for 50 units, or as little as pennies  
each in large quantities.  
(Corporate Site License required.)

The total software environment for  
IBM PC/XT, TRS-80 Model 1, 3, 4  
and close friends.

- Personal License (required):  
MMSFORTH V2.4 System Disk . . . . . \$179.95  
(TRS-80 Model 1 requires lowercase, DDEN, 1 40-track drive.)
  - Personal License (additional modules):  
FORTHCOM communications module . . . . . \$ 49.95  
UTILITIES . . . . . 49.95  
GAMES . . . . . 39.95  
EXPERT-2 expert system . . . . . 69.95  
DATAHANDLER . . . . . 59.95  
DATAHANDLER-PLUS (PC only, 128K req.) . . . . . 99.95  
FORTHWRITE word processor . . . . . 99.95
  - Corporate Site License  
Extensions . . . . . from \$1,000
  - Bulk Distribution . . . . . from \$500/50 units.
  - Some recommended Forth books:  
FORTH: A TEXT & REF. (best text) . . . . . \$ 19.95  
THINKING FORTH (best on technique) . . . . . 16.95  
STARTING FORTH (popular text) . . . . . 19.95
- Shipping/handling & tax extra. No returns on software.  
Ask your dealer to show you the world of  
MMSFORTH, or request our free brochure.

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

# Windows for the TI 99/4A



Blair MacDermid  
Fort Wayne, Indiana

The Forth routines described here provide plotting of algebraic functions in a choice of five windows located in different positions of the display screen. These windows can be displayed simultaneously. Four of the windows can be located in the four quadrants of the display screen. The fifth window occupies most of the full screen. The program computes the coordinates of the plotted function, appropriately scaled to fit within the selected window.

These routines are a by-product of the group project undertaken as a learning exercise by members of the Fort Wayne FIG Chapter. The primary objective of the exercise was to allow the group members to participate in development of a useful Forth program, with efforts partitioned among members according to their skills. The simpler elements of the program were assigned to beginners. We also hoped to learn how well Forth would serve in a multiple-programmer task.

Ed Harmon, the chapter's guru, selected the ACM SIGGRAPH CORE Standard as a useful tool. He provided us with a model written in the UCSD p-System Pascal (see the *Journal of Pascal, Ada, Modula-2*, May/June 1984, page 19). Ron Bishop, president of the local TI 99/4A Users Group, completed the program using thirty screens and integer arithmetic. That implementation provides the freedom to locate and define the size of a number of viewports (i.e., windows) to be displayed simultaneously. The size and location are continuously adjustable.

Here I have defined a simpler version of the program, using only six screens. It does not provide the degree of freedom intended to be part of the ACM standard. However, it provides a useful choice of window locations and sizes. The program exploits the excellent graphics capabilities of the TI 99/4A using the SPLIT2 mode and the TI-FORTH words **DOT** and **LINE**, as well as the TI 99/4A's floating-point routines (which I used to plot functions

that contain the transcendental functions, e.g., sine, tangent, logarithm).

It will be useful to refer to the Forth screens 30-35 in the discussion that follows. These screens provide a useful utility but can readily be modified and expanded to include different elements of the Pascal model of the full ACM SIGGRAPH CORE Standard.

## Screen 30

Lines 1-9 define the required variables. The variables associated with the horizontal axis use X, as is common practice. **XMIN** and **XMAX** represent the minimum and maximum values of the real-world function to be plotted. **VL** and **VR** represent the left- and right-hand viewport coordinates in pixels, referenced to the TI screen display. Similarly, **VBOT** and **VTOP** represent the bottom and top coordinates of the viewport.

The function  $Y=f(X)$  is computed using X as the independent variable in the world coordinates. **XD** is the corresponding variable referenced to the display screen coordinates. Similarly, **YDB** represents the display screen coordinate corresponding to Y of the world coordinates.

The variable **YDB** warrants further explanation. The TI 99/4A screen display uses coordinates that reference the upper left-hand corner of the screen as the 0,0 point. I found this confusing, since it is normal to use the lower left-hand corner as the origin when plotting functions. So I invented **YDB** to allow me to readily handle the necessary mathematics. Subsequently, when specifying the coordinates to be plotted on the screen, I use the constant **YT10** equal to 191 (see screen 31, line 2 and screen 35, line 5) to make the necessary corrections for the TI 99/4A screen coordinates. This is justified by the following relation:

$$(TI's Y) + YDB = YT10 = 191$$

Therefore,

$$(TI's Y) = YT10 - YDB$$

The variables **KX** and **KY** are scaling factors modifying the world coordinate

variables to fit the selected viewport dimensions.

The variable **YDBARA** is an array to store 200 computed values of **YDB**. The TI 99/4A shares some of its display facilities with the floating-point routines; to avoid any difficulty in this regard, I chose to compute the values of the plotted function (see screen 34) before using the screen to display the function (see screen 35).

All of the variables discussed above represent integer values. This assumes that the selected minimum and maximum world coordinate variables will be integer values, the normal thing to do. Certain of these variables will require floating-point representatives in the computation routine of screen 34. These variables have been prefixed with an F as in **FX**, **FKX** and **FKY**.

The words **KXCALC** and **KYCALC** specify the computation of the scaling factors **KX** and **KY**, as well as the floating-point equivalents **FKX** and **FKY**. It may be helpful to display the mathematical definitions of these variables:

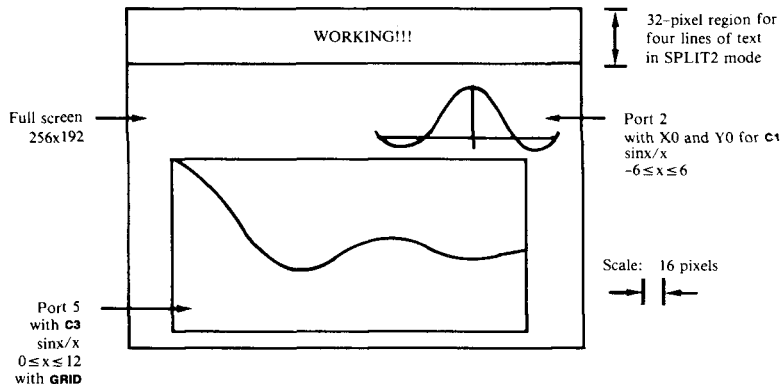
$$KX = (VR-VL)/(XMAX-XMIN)$$

$$KY = (VTOP-VBOT)/(YMAX-YMIN)$$

TI-FORTH uses the word **S->F** to convert integer values to floating point.

## Screen 31

This screen contains the definition of the viewport size and location. The word **PICKPORT** selects the viewport corresponding to the number (one through five) placed on the stack before executing the word. The numbers one through four select a viewport, size 100x50 pixels high, at locations in any of the four quadrants of the display screen. A value of five selects the largest viewport, 200x100, occupying most of the available screen display. There is sufficient space to allow a height of 190 pixels; however, it is easier to interpolate values of Y with the height of 100 pixels. Of course, other viewport dimensions and locations can readily be specified by substituting different numbers. (I find



the word **PP** useful in exercising the program, since it relieves me of the need to type **PICKPORT**, whose length is dictated by the desire to write readable code.)

### Screen 32

The words **Y LINES** and **X LINES** use TI-FORTH's **LINE** to draw vertical and horizontal lines at useful increments. The word **GRID** uses these words to superimpose on the viewport a grid to expedite interpolation of values of the displayed function. The words **X0** and **Y0** are abbreviated versions of the words locating 0 axes for both X and Y.

### Screen 33

The words **C1**, **C2** and **C3** specify different parameters for the world coordinates of the function to be plotted. The word **FUNCTION** specifies the function to be plotted, in this case  $\sin x/x$ , defining the Fourier spectrum of the rectangular pulse waveform. Sufficient space is available on this screen to substitute another definition of the word **FUNCTION**. Notice, however, that the definition must use floating-point representation.

### Screen 34

The word **ARAYDB** specifies the computation of the values stored by **YDB** in the array **YDBARA**. It also causes the word **WORKING!!!** to be displayed on the screen so that the user will not assume his computer has contracted amnesia while executing the calculations. Note that lines 2-5 perform calculations in integer arithmetic, and the results are converted to floating point by line 6. Line 8 contains the word **FUNCTION**, and the resulting computation is con-

verted to integer by the word **F->S** (a single-precision integer value in two bytes). The word **CY** is my convenient macro for **ARAYDB**.

The **DO LOOP** increments the current value of **XD** by one pixel from **VL** to **VR**. Lines 3-4 compute the corresponding value of the world coordinates. This value is converted to floating point and is divided by the floating-point representative of the scaling factor **FKX**. The result is placed on the stack, and a copy is stored in **FX** where it can be used in more complicated functions requiring different powers and functions of X.

The computed value of **FUNCTION** is multiplied by the scaling factor **FKY** to define the corresponding value **YDB** for the display screen. The resulting computation for each increment in **XD** produced by the **DO LOOP** is stored in the array **YDBARA**.

### Screen 35

The word **PLOTY** uses another loop to increment **XD** in one-pixel increments and selects the appropriate element of the array **YDBARA** to plot the function on the screen using the TI-FORTH word **DOT**. (**PY** is my macro for initiating the plotting routine.)

### Final Notes

The definitions used assume the SPLIT2 graphics mode if the TI 99/4A is used. TI-FORTH is a fig-FORTH extension, but the words **DOT** and **LINE** are probably machine dependent. However, it is reasonable to assume the screens could be modified to work on a different Forth implementation. Both the Apple II and the IBM-PC have graphics capabilities providing pixel resolution.

COMBINE THE  
RAW POWER OF FORTH  
WITH THE CONVENIENCE  
OF CONVENTIONAL LANGUAGES

# HS / FORTH

Why HS/FORTH? Not for speed alone, although it is twice as fast as other full memory Forths, with near assembly language performance when optimized. Not even because it gives MANY more functions per byte than any other Forth. Not because you can run all DOS commands plus COM and EXE programs from within HS/FORTH. Not because you can single step, trace, decompile & disassemble. Not for the complete syntax checking 8086/8087/80186 assembler & optimizer. Nor for the fast 9 digit software floating point or lightning 18 digit 8087 math pack. Not for the half megabyte LINEAR address space for quick access arrays. Not for complete music, sound effects & graphics support. Nor the efficient string functions. Not for unrivaled disk flexibility — including traditional Forth screens (sectored or in files) or free format files, all with full screen editors. Not even because I/O is as easy, but far more powerful, than even Basic. Just redirect the character input and/or output stream anywhere — display, keyboard, printer or com port, file, or even a memory buffer. You could even transfer control of your entire computer to a terminal thousands of miles away with a simple >COM <COM pair. Even though a few of these reasons might be sufficient, the real reason is that we don't avoid the objections to Forth — WE ELIMINATE THEM!

Public domain products may be cheap; but your time isn't. Don't shortchange yourself. Use the best. Use it now!

HS/FORTH, complete system: \$395. with "FORTH: A Text & Reference" by Kelly and Spies, Prentice-Hall and "The HS/FORTH Supplement" by Kelly and Callahan



## HARVARD SOFTWARES

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

# PORTABLE POWER WITH MasterFORTH



Whether you program on the **Macintosh**, the **IBM PC**, an **Apple II** series, a **CP/M** system, or the **Commodore 64**, your program will run unchanged on all the rest.



If you write for yourself, MasterFORTH will protect your investment. If you write for others, it will expand your marketplace.

Forth is interactive - you have immediate feedback as you program, every step of the way. Forth is



**CP/M** use its built-in assembler to make it even faster. MasterFORTH's relocatable utilities and headerless code let you pack a lot more program into your memory. The resident debugger lets you decompile, breakpoint and trace your way through most programming problems. A string package, file interface and full screen editor are all standard features. And the optional target compiler lets you optimize your application for virtually any programming environment.

The package exactly matches *Mastering Forth* (Brady, 1984) and meets all provisions of the Forth-83 Standard.

MasterFORTH standard package..... \$125  
(Commodore 64 with graphics)..... \$100

#### Extensions

Floating Point..... \$60  
Graphics (selected systems)..... \$60  
Module relocater (with utility sources)..... \$60  
TAGS (Target Applic. Generation System) -  
MasterFORTH, target compiler and  
relocater..... \$495

#### Publications & Application Models

Printed source listings (each)..... \$35  
Forth-83 International Standard..... \$15  
Model Library, Volumes 1-3 (each)..... \$40

(213) 821-4340



**MICROMOTION**

8726 S. Sepulveda Bl., #A171  
Los Angeles, CA 90045

```
SCR #30
0 CR ." SCR#30" ( TI-FORTH PLOTTING ROUTINES 7/7/85 )
1 0 VARIABLE XMIN 0 VARIABLE YMIN ( WORLD COORDINATES )
2 0 VARIABLE XMAX 0 VARIABLE YMAX
3 0 VARIABLE VL 0 VARIABLE VBOT ( VIEWPORT COORDINATES )
4 0 VARIABLE VR 0 VARIABLE VTOP
5 0 VARIABLE X 0 VARIABLE FX 6 ALLOT
6 0 VARIABLE XD 0 VARIABLE YDB
7 0 VARIABLE KX 0 VARIABLE KY ( X & Y SCALE FACTORS )
8 0 VARIABLE FKX 6 ALLOT 0 VARIABLE FKY 6 ALLOT
9 0 VARIABLE YDBARA 400 ALLOT
10
11 : KXCALC VR @ VL @ - XMAX @ XMIN @ - / DUP KX ! S->F FKX F ! ;
12 : KYCALC VBOT @ VTOP @ - YMAX @ YMIN @ - / DUP
13 KY ! S->F FKY F ! ;
14
15 : 2DUP DUP DUP ; -->

SCR #31
0 CR ." SCR#31" ( PICKPORT BWM 7/7/85 )
1 0 VARIABLE VBOTB : SVBOTB 191 VBOT @ - VBOTB ! ;
2 0 VARIABLE YINC 20 CONSTANT XINC 191 CONSTANT YTI0
3 : PICKPORT ( n --- ) 2DUP 2DUP
4 1 = IF 96 VBOT ! 46 VTOP ! 12 VL ! 112 VR ! 10 YINC !
5 SVBOTB ENDIF ( TOP LEFT VIEWPORT )
6 2 = IF 96 VBOT ! 46 VTOP ! 144 VL ! 244 VR ! 10 YINC !
7 SVBOTB ENDIF ( TOP RIGHT )
8 3 = IF 178 VBOT ! 128 VTOP ! 144 VL ! 244 VR ! 10 YINC !
9 SVBOTB ENDIF ( BOTTOM RIGHT VIEWPORT )
10 4 = IF 178 VBOT ! 128 VTOP ! 12 VL ! 112 VR ! 10 YINC !
11 SVBOTB ENDIF ( BOTTOM LEFT )
12 5 = IF 182 VBOT ! 82 VTOP ! 25 VL ! 225 VR ! 10 YINC !
13 SVBOTB ENDIF SP ! ; ( FULL SCREEN VIEWPORT )
14
15 : PP PICKPORT ; ( n --- ) -->

SCR #32
0 CR ." SCR#32 " ( BWM PLOTTING UTILITES 7/7/85 )
1 0 VARIABLE YD 0 VARIABLE XNN
2 : YLINES VBOT @ 5 + VTOP @ DO I YD !
3 VL @ YD @ VR @ YD @ LINE YINC @ +LOOP ;
4 : XLINES VR @ 5 + VL @ DO I XNN !
5 XNN @ VBOT @ XNN @ VTOP @ LINE
6 XINC +LOOP ;
7 : GRID YLINES XLINES ;
8 : YTI0 YTI0 VBOTB @ - YMIN @ S->F FKY F @ F* F->S +
9 VL @ SWAP VR @ OVER LINE ; ( DRAWS LINE Y = 0 )
10 : Y0 YTI0 ;
11 : XTI0 VL @ XMIN @ S->F FKX F @ F* F->S - VTOP @
12 OVER VBOT @ LINE ; ( DRAWS LINE X = 0 )
13 : X0 XTI0 ;
14
15 -->

SCR #33
0 CR ." SCR#33 " ( CANNED EXAMPLES WORLD COORDINATES BWM 7/8/85 )
1
2 : C1 -6 XMIN ! 6 XMAX ! -1 YMIN ! 1 YMAX ! KXCALC KYCALC ;
3
4 : C2 0 XMIN ! 24 XMAX ! -1 YMIN ! 1 YMAX ! KXCALC KYCALC ;
5
6 : C3 0 XMIN ! 12 XMAX ! -1 YMIN ! 1 YMAX ! KXCALC KYCALC ;
7
8 : FUNCTION SIN FX F @ F / ; ( x --- sinx/x in fltg pt )
9
10
11
12
13
14
15 -->
```

(Screens continued on page 40.)

# Getting Started with F83



Greg McCall  
Werrington, NSW, Australia

The documentation with F83 is in F83.COM and in the shadow screens that are part of the source files that come with F83. At first glance, the thought of sifting through hundreds of kilobytes of shadow screens is bewildering, to say the least. Just to get you started, I have put together a summary of how to use the file words and how to edit these files. This relates to the CP/M-80 version of F83, but as far as I know it should be similar to other versions of Laxen and Perry's F83.

This Forth can have two files open at once. One file is called the **CURRENT** file. This is the file used by all normal reads and writes. You would normally edit or load from the **CURRENT** file. The other file is called the **FROM** file. This is a second file you may have open for reading only. For example, if you currently are working on a file (i.e., loading and editing), and you wish to load some screens from another file, then you may open a **FROM** file and load screens from it without changing the **CURRENT** file. Following is a description of some useful file words:

## **CREATE-FILE** (S n --)

Creates a new file containing n blocks.

### **10 CREATE-FILE TEST.BLK**

opens a file called test.blk and writes ten blank screens to this file. The file is then closed.

**FILE?** Prints the name of the **CURRENT** file.

**DIR** Prints the directory of the current drive.

**OPEN** Open the following file name and make it the current file, e.g., **OPEN TEST.BLK**

**FROM** Make the next word in the input stream the **FROM** file and **OPEN** it. It then sets the current vocabulary to **FILES**.

## **LOAD**

In the **FORTH** vocabulary, **LOAD** will load screens from the **CURRENT** file. In the **FILES** vocabulary, **LOAD** will load screens from the **FROM** file. So while we have a file as the **CURRENT** file, we can still open another file by making this second file the **FROM** file and loading from it, e.g., **FROM TEST.BLK**  
**10 LOAD**

## **CA**

Copy a screen to its shadow.

## **COPY**

(S from to --)

In the **FORTH** vocabulary, copies a screen in the **CURRENT** file. In the **FILES** vocabulary, copies a screen from the **FROM** file to the **CURRENT** file. In the **SHADOW** vocabulary, copies a screen and its shadow in the **CURRENT** file.

## **CONVEY**

(S from to --)

In the **FORTH** vocabulary, copies a set of screens in the **CURRENT** file. In the **FILES** vocabulary, copies a set of screens from the **FROM** file to the **CURRENT** file. In the **SHADOW** vocabulary, copies a set of screens and their shadows in the **CURRENT** file.

## **HOPPED**

A variable containing the number of screens to skip when copying with **CONVEY**.

## **U/D**

A variable containing the direction of the screen move using **CONVEY**. +1 is a forward screen move and -1 is a backward screen move.

## **TO**

Sets up the variables **HOPPED** and **U/D**. Used as *first-source last-source TO first-destination CONVEY*

The F83 editor uses the same words as the editor in *Starting Forth* by Leo Brodie, with some additions such as the word **NEW** which allows replacement of

multiple lines. To get the editor going correctly, you should look at screens 28 - 30 and 88 of **UTILITY.BLK** which hold the terminal-dependent routines. You can select your terminal — or see if any of the routines are the same as those of your terminal — or write your own routines. The terminal words patch the words **AT**, **DARK**, **BLOT** and **-LINE** to suit your terminal. While we are looking at patching the editor, you could remove the backslash in line 14 of screen 24 so that **(WHERE)** is patched into **WHERE** and, if you have a real-time clock, then you could change **GET-ID** in screen 23 so as to have the **ID** supplied when the editor is first invoked. These screen numbers refer to the CP/M-80 version of F83. To find where the source screens are for the editor in your Forth, type **VIEW AT** which should give you the second source screen of your editor. Now just look through the editor's screens for the required words.

A summary of the editor commands follows:

**TOP** Go to the top of the screen.

**C** (S n --)  
Move n characters, right or left.

**T** (S n --)  
Go to beginning of line n.

**.BUFS** Displays the contents of the insert and find buffers.

**KEEP** Places the current line in the insert buffer.

**K** Exchanges the contents of the insert and find buffers.

**W** Write all changes to disk.

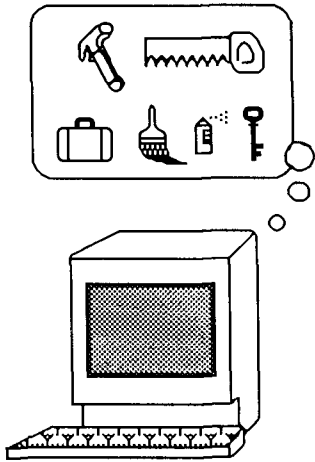
**N** Move to next screen.

**B** Move back a screen.

**A** Alternate between a screen and its shadow.

# ForthTalk!

Fast object based programming!  
Improves on **SmallTalk** concepts!



## FEATURES:

Builds on MacFORTH level 1  
Multiple Inheritance  
Unshadowed Mixins  
Method Combination  
Flavor Variables  
Instance Variables  
SELF Pseudo-Variable  
Debug Tools:  
Formatted Traceback  
Message Tracing  
Formatted Object Dumps  
and Descriptions

\$55

Available soon on  
Atari ST and Amiga

Created by:

**InSite Computing**

P.O. Box 2949, Ann Arbor, MI 48106  
313/994-3660

Also available from:

**MacForth Users Group**

3081 Westville Station  
New Haven, CT 06515  
203/777-5618

MacFORTH is a registered trademark  
of Creative Solutions, Inc.

<b>&lt; text &gt;</b>	Represents the text following the command. If <b>&lt; text &gt;</b> is just a carriage return, the contents of the insert buffer is used in place of the missing <b>&lt; text &gt;</b> .	inserts it in front of the current line. In the <b>SHADOW</b> vocabulary, <b>G</b> gets a line and its shadow. In the <b>FILES</b> vocabulary, <b>G</b> gets the line from the <b>FROM</b> file.
<b>I &lt; text &gt;</b>	Inserts <b>&lt; text &gt;</b> on the current line at the cursor.	<b>BRING</b> (S screen first last -- ) Brings several lines from another screen and inserts them in front of the current line. In the <b>SHADOW</b> vocabulary, <b>BRING</b> gets a range of lines and their shadows. In the <b>FILES</b> vocabulary, <b>BRING</b> gets the lines from the <b>FROM</b> file.
<b>O &lt; text &gt;</b>	Overwrites <b>&lt; text &gt;</b> onto the current line.	<b>NEW</b> (S n -- ) Moves the terminal's cursor to the start of line n and overwrites until the line has a null input, i.e., just a carriage return.
<b>P &lt; text &gt;</b>	Replaces the current line with <b>&lt; text &gt;</b> and blanks.	<b>QUIT</b> Exits the editor without updating or flushing.
<b>U &lt; text &gt;</b>	Inserts a line under the current line.	<b>DONE</b> Exits the editor, updates the ID stamp, tells you if the screen was modified, flushes it to disk and removes automatic redisplay.
<b>F &lt; text &gt;</b>	Finds the <b>&lt; text &gt;</b> and leaves the cursor just past it.	<b>ED</b> Re-enters the editor. It clears and reinitializes the display, and begins automatic redisplay of the screen.
<b>S &lt; text &gt;</b> (S n -- )	Searches for <b>&lt; text &gt;</b> through all screens from the current one up to screen n. Each time a match is found, n remains on the stack until screen n is reached. To continue the search, just type S until screen n is reached.	<b>EDIT</b> (S n -- ) Sets <b>SCR</b> to n, then uses <b>ED</b> to start editing.
<b>R &lt; text &gt;</b>	Replaces the text just found with <b>&lt; text &gt;</b> .	This should enable you to copy and edit screens with Laxen and Perry's F83. The best way to get the entire documentation on this Forth is by printing out all the source files. If your printer can print at least 132 characters per line, then look in your printer's manual for the characters needed to put your printer in this mode. My FAX-80 needs a control-O to set the condensed mode. I define a word <b>FAX-80</b> to send this code and then patch it into the <b>DEFERED</b> word <b>INIT-PR</b> , i.e.:
<b>D &lt; text &gt;</b>	Finds and deletes the text.	: <b>FAX-80 CONTROL O EMIT</b> ; ' <b>FAX-80 IS INIT-PR</b>
<b>TILL &lt; text &gt;</b>	Deletes all text on the line from the cursor up to and including <b>&lt; text &gt;</b> .	To print the entire file, you use the word <b>LISTING</b> . For example, to list <b>META80.BLK</b> , I would type: <b>OPEN META80.BLK LISTING</b>
<b>JUST &lt; text &gt;</b>	Deletes up to, but not including, <b>&lt; text &gt;</b> .	
<b>KT &lt; text &gt;</b>	Puts all text between the cursor and <b>&lt; text &gt;</b> inclusive into the insert buffer ("keep-till").	
<b>E</b>	Erases the text just found by <b>F</b> or <b>S</b> .	
<b>X</b>	Deletes the current line.	
<b>SPLIT</b>	Breaks the current line in two at the cursor.	
<b>JOIN</b>	Puts a copy of the next line after the cursor.	
<b>WIPE</b>	Clears the screen to blanks.	
<b>G</b> (S screen line -- )	Gets a line from another screen and	

# Batcher's Sort



John Konopka  
Mitaka Shi, Japan

Quicksort is often suggested as a sorting algorithm because of its speed. The reputation for speed is well deserved but Quicksort has other features which may make it difficult to use. An alternative sorting method discovered by K.E. Batcher in 1964<sup>1,2</sup> is a little slower than Quicksort but is more robust and avoids most of Quicksort's pitfalls.

One problem with Quicksort is its variable performance. It is usually stated that Quicksort requires about  $N \log N$  operations to sort  $N$  items (base 2 logarithm). This is an average result which depends on the input data being random. In other cases where the data is already ordered in some way, then Quicksort may require as many as  $N^2$  operations to sort  $N$  items. This is as slow as a Bubble sort. Thus you don't know from one execution to the next just how long a sort will take. Extra code can be added — complicating the algorithm — to handle some, but not all, of the time-consuming cases. Quicksort also varies in its use of space. Every branch in Quicksort creates one stack entry (the number of words per stack entry is implementation specific) on the return stack (if, as usual, recursion is used). Normally, a maximum of about  $\log N$  stack entries are created. However, in degenerate cases this number may approach  $N$ . When sorting ill-ordered data you may find your program running out of room with unanticipated consequences.

A second source of trouble with Quicksort is that it is difficult to implement. Quicksort is generally presented in a recursive form. If recursion is not available you must implement this yourself. You can, at the expense of more complicated code, implement a non-recursive version<sup>3</sup>. To limit, but not eliminate, the number of cases requiring much time or much stack space more code can be added, again increasing the complexity of the algorithm. The final implementation problem is how to test it. Because the operation of the algorithm is data

## Sample portions of link map data.

```

OVLY SEG  SIZE
1 1 1833 *****
1 2 1130 *****
1 3 1972 *****
1 4 2245 *****
1 5 1696 *****
1 6 2495 *****
1 7 2402 *****
1 10 1499 *****

```

## Before sorting

```

OVLY SEG  SIZE
1 21 2621 *****
1 27 2618 *****
1 25 2509 *****
1 6 2495 *****
1 13 2485 *****
1 17 2469 *****
1 14 2443 *****
1 7 2402 *****

```

## After sorting

Figure One

## 100 random numbers before and after sorting.

```

14820 -10904 29081 -30212 3226 -16975 -6865 -31694 28585 29040
-22503 25399 24896 -27251 21804 29720 -10403 11702 -3684 -13959
1293 -17882 11160 16792 -28685 21788 364 3362 -14444 -32176
-24843 27148 -1267 -17090 28362 -16741 249 12214 -32405 1678
31832 -7663 9461 30700 -7458 -12676 -7101 9277 -6936 -8360
-15913 -13499 -27433 14612 -8610 -26152 -9637 -19365 6962 6143
-31048 -19079 711 13083 -16616 -14840 15938 -19628 -19793 20656
22997 32032 18638 -9148 1954 968 9551 -17276 11578 -16357
17601 5905 -3600 -20587 -14952 -5764 26437 -28174 -474 -22334
-32679 6053 32007 -1349 30393 -14024 -26301 4785 28746 -22250

32032 32007 31832 30700 30393 29720 29081 29040 28746 28585
28362 27148 26437 25399 24896 22997 21804 21788 20656 18638
17601 16792 15938 14820 14612 13083 12214 11702 11578 11160
9551 9461 9277 6962 6143 6053 5905 4785 3362 3226
1954 1678 1293 968 711 364 249 -474 -1267 -1349
-3600 -3684 -5764 -6865 -6936 -7101 -7458 -7663 -8360 -8610
-9148 -9637 -10403 -10904 -12676 -13499 -13959 -14024 -14444 -14840
-14952 -15913 -16357 -16616 -16741 -16975 -17090 -17276 -17882 -19079
-19365 -19628 -19793 -20587 -22250 -22334 -22503 -24843 -26152 -26301
-27251 -27433 -28174 -28685 -30212 -31048 -31694 -32176 -32405 -32679

```

Figure Two

dependent you may have sleeping bugs which only awaken when presented with rightly ordered data. See the Sedgewick and Knuth references for more information about Quicksort.

Batcher's sort suffers none of these problems. It iterates the same way every time, calculating the same pairs of indices regardless of the data presented for sorting. It sorts in place, requiring no buffer space, and it places no unusual demands on either the return or data stacks. Furthermore, it is easy to implement, requiring only one screen of Forth code. Recursion is

not required. Finally, because it is simpler there are fewer things to go wrong. It is thus easier to test and easier to trust. Once you have it working for one set of data it is likely to work well afterwards.

The cost for this robustness is time. Quicksort requires, on average, about  $N \log N$  operations. Batcher's sort requires less than  $(N/4) \log N [(\log N) + 1]$  iterations. The difference is less than  $(\log N + 1)/4$ . As an example of what this means in terms of normal array sizes Quicksort should be, on average, about two times faster when sorting 1024 items.

This does not take into account any time difference for one iteration between Quicksort and Batcher's sort. The clincher is the phrase "on average." Depending on the input data, in some cases Batcher's sort may in fact be quicker than Quicksort. In any event the absolute difference in time will probably not be large. For example, using no code words I can sort 512 names on a DEC LSI 11/23 in twelve seconds. In this case the cost for using Batcher's sort is certainly tolerable.

Batcher's sort has one more interesting feature which someday may let it far outpace Quicksort or any other sorting method, in terms of speed. Looking at the code you can see three nested loops. At every iteration of the innermost loop **INNER-LOOP** the pairs of keys which are compared are completely independent. Thus a parallel computer could implement the inner loop in one step for really fast sorting. The number of iterations in this case is just  $(1/2)\log N[(\log N)+1]$ . This is just fifty-five iterations when processing an array of 1024 items.

## Implementation

The Forth code for the sort is displayed in screen 2. While the code is not particularly complex, the operation of the algorithm is not obvious. See Knuth for further details. The program uses seven constants: **TT**, **PP**, **DD**, **NN**, **RR**, **QQ** and **KC**. These names were chosen to be consistent with the description of the algorithm given by Knuth. **QQ** can easily be carried only on the stack but I made it explicit for easier reading. Constants are used rather than variables, as the data is accessed much more often than it is set. **TT** stores a parameter which determines the sizes of the outer loops. It is calculated in **SELECT-T**. **PP** drives the outermost loop, **QQ** drives the next nested loop. These loops are driven by dividing the loop counter by two rather than by incrementation as in **DO LOOP**. **RR**, **NN** and **DD** are used to calculate indices to keys. When sorting  $N$  items, this routine generates indices in the range from zero to  $N-1$ . The actual output of the program is this sequence of number pairs. Implementation-

```

Screen #2
1 \ BSORT K. E. Batcher's sort. From Knuth, vol 3.
2 0 CONSTANT TT 0 CONSTANT RR 0 CONSTANT DD 0 CONSTANT PP
3 0 CONSTANT NN 0 CONSTANT QQ 0 CONSTANT KC
4 : KEY_COMPARE KC EXECUTE ;
5 : SELECT-T NN 15 0 DO DUP I 2**N <= IF DROP I LEAVE THEN LOOP
6   1- 14 MIN ' TT ! ;
7 : INNER-LOOP NN DD - 0 DO I PP AND RR =
8   IF I DUP DD + KEY_COMPARE THEN LOOP ;
9 : Q-TEST QQ PP <> IF QQ PP - ' DD ! QQ 2/ ' QQ !
10  PP ' RR ! 0 THEN ;
11 : QRD-SET TT 2**N ' QQ ! 0 ' RR ! PP ' DD ! ;
12 \ n --- n is number of items to sort. n must be positive.
13 : BSORT ' NN ! SELECT-T TT 2**N ' PP !
14   BEGIN QRD-SET QQ
15     BEGIN INNER-LOOP Q-TEST UNTIL
16     PP 2/ DUP ' PP ! 0= UNTIL ;

Screen #6
1 \ BSORT example. Sort array of integers.
2 0 CONSTANT X1 0 CONSTANT X2 CREATE DATA 200 ALLOT
3
4   --- Load array DATA with random numbers.
5 : INIT-DATA 100 0 DO RANDOM DROP I 2* DATA + ! LOOP ;
6
7 \ --- Exchange entries pointed to by X1 and X2.
8 : SWAP-DATA X1 DATA + @ X2 DATA + @ X1 DATA + ! X2 DATA + ! ;
9
10 \ N M --- Compare and maybe exchange Nth and Mth entries.
11 : COMPARE-AND-SWAP 2* ' X1 ! 2* ' X2 ! \ Save pointers
12   X1 DATA + @ X2 DATA + @ > \ Compare values
13   IF SWAP-DATA THEN ; \ Exchange if misordered
14 \ --- .R is defined in 79-Standard Reference Word Set.
15 : LIST-DATA 100 0 DO I 2* DATA + @ 7 .R I 1+ 10 MOD 0=
16   IF CR THEN LOOP ;

```

(Screens continued from page 36.)

```

SCR #34
@ CR ." SCR#34 " ( ARRAY YDB CALC FLTG POINT BWM 7/7/85 )
1 : ARAYDB CLS ." WORKING !!! " ( --- YDB[i] )
2   VR @ VL @ DO I XD !
3     I VL @ -
4     XMIN @ KX @ *
5     + ( diff prod --- kx*x )
6     S->F FKX F@ F/ ( f[kx*x] --- fx )
7     FDUP FX F! ( stores current fx )
8 FUNCTION FKY F@ F* ( fx --- fy*fky )
9   F->S ( fky*fy --- ky*y )
10  YMIN @ KY @ * - VBOTB @ +
11  YDBARA I VL @ - 2 * + ! ( store ydb in array ydbara )
12  LOOP ;
13 ( COMPUTES VIEWPORT REPRESENTATIVE OF WORLD Y )
14
15 : CY ARAYDB ; -->

SCR #35
@ CR ." SCR#35 " ( BWM PLOTTING UTILITES 7/7/85 )
1 : PLOTY
2   VR @ VL @ DO I XD !
3     I VL @ - 2 *
4     YDBARA + @
5     YTI@ SWAP - XD @ SWAP DOT ( PLOTS NEXT PT )
6     LOOP ;
7
8 : PY CLS PLOTY ;
9
10
11
12
13
14
15

```



specific code uses these pairs of numbers to point to the items to be sorted, then does the compare and possible exchange. If, for example, you were sorting a list of names and the output was 1 and 5, then your implementation-specific word would compare the first and fifth names in the list and exchange their positions if they were misordered. The execution address of this code is stored in constant **KC**. The word **KEY\_COMPARE** accesses that constant and executes the word whose address is stored there. By this vectoring, the sort routine is separated from the data being sorted so you can use the same sort routine for all applications. To use the routine, put the execution address of your compare code in constant **KC**, put **N** (the number of items to be sorted) on the stack, then invoke **BSORT**.

#### Application Examples

Screen 6 shows an application which sorts an array of random data. The array is initialized with a random number generator<sup>5</sup> by invoking **INIT-DATA**. If a random number generator of some kind is not available you can load the array with an editor, using , (comma) to enter integers picked from your imagination. **LIST-DATA** will type the data on a terminal. To sort the data put the execution address of **COMPARE-AND-SWAP** in constant **KC**, then put 100 on the stack and invoke **BSORT**:

```
FIND COMPARE-AND-SWAP ' KC !
100 BSORT
```

Now you can use **LIST-DATA** to see the effect of sorting. This simple example is useful for verifying the operation of **BSORT**.

As another example application, I use this routine to sort the vocabulary names in the Forth dictionary. Code specific to my system first scans the dictionary and builds an array of addresses. Each entry points to the name field of a Forth word. The length of the array is the number of words in the

dictionary. The comparison word deposited in **KC** takes two indices from **BSORT** and using these pointers compares two names in the dictionary, then exchanges the addresses stored in the array if the names are not in alphabetical order. This comparison word must not only know how to compare strings alphabetically but it must be able to strip out special bits such as the **IMMEDIATE** flag, and it must be able to determine the length of the name. After sorting, I write the names to a text file and then use an editor to make glossaries for documenting applications. See the paper by Baden<sup>4</sup> for another example of sorting vocabulary names in the Forth dictionary.

In another case I use this routine to sort information about a large Fortran program. When the program is compiled and linked, a map is generated giving, among other information, the size of each of the program overlays. The size of the program in memory is determined by the largest segments; thus, to reduce the memory requirements one needs to know which are the largest segments and how they differ from the second or third largest segments. I wrote one routine to scan the map and extract the size information. For sorting, the word deposited in **KC** compares these sizes numerically and exchanges them if they were out of order. Figure One graphically shows the results before and after sorting.

In the near future, I have two more sorting applications in mind. One is in an application I wrote called "Card File." This is a software version of a box of 3x5 cards. In this case I will first create an array of pointers in memory indicating which cards I want to list on the printer. Then I will sort this list using **BSORT**. The most natural order would be to alphabetize the cards according to the first word on a given row of the card. The application-specific word which would be deposited in **KC** would have to know how to extract this information from the cards, then do the compare and swap pointers if needed. The second application I have in mind is in x-ray spectroscopy. I now have Forth words which create directories of file names of stored x-ray

spectra. It would be helpful to sort these directories in various ways. Just by changing the compare word deposited in constant **KC** I will be able to sort the directory according to file name, date, number of elements in the spectra or even according to the atomic numbers of the elements which generated the spectra.

From these few examples you can see that almost anything can be sorted. All you need is a word which knows how to compare two items in a list and exchange them if they are misordered. If the items are small and easy to move, then you can exchange the positions of the items themselves. If it is costly to move the items, as in the case of disk-based data, it is better to keep a list of pointers and just exchange the pointers.

#### References

1. Batcher, K.E., *Proceedings AFIPS Spring Joint Computer Conference*, 32(1968), 307-314.
2. Knuth, Donald E., *Art of Computer Programming*, Vol. 3, pp 111-122, Addison-Wesley, 1973.
3. Sedgewick, Robert, *Algorithms*, pp 107-111, Addison-Wesley, 1983.
4. Baden, Wil, "Quicksort and Swords", *Forth Dimensions* VI/5, 1985.
5. Doyle, William T., "A Portable Forth Random Number Generator", *Journal of Forth Applications and Research*, vol. 1, no. 2, 1983.

## U.S.

### • ALABAMA

**Huntsville FIG Chapter**  
Call Tom Konantz  
205/881-6483

### • ALASKA

**Kodiak Area Chapter**  
Call Horace Simmons  
907/486-5049

### • ARIZONA

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

**Tucson Chapter**  
Twice Monthly,  
2nd & 4th Sun., 2 p.m.  
Flexible Hybrid Systems  
2030 E. Broadway #206  
Call John C. Mead  
602/323-9763

### • ARKANSAS

**Central Arkansas Chapter**  
Twice Monthly, 2nd Sat., 2p.m. &  
4th Wed., 7 p.m.  
Call Gary Smith  
501/227-7817

### • CALIFORNIA

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

**Monterey/Salinas Chapter**  
Call Bud Devins  
408/633-3253

**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

**Sacramento Chapter**  
Monthly, 4th Wed., 7 p.m.  
1798-59th St., Room A  
Call Tom Ghormley  
916/444-7775

**Bay Area Chapter**  
Silicon Valley Chapter  
Monthly, 4th Sat.  
FORML 10 a.m., Fig 1 p.m.  
H-P Auditorium  
Wolfe Rd. & Pruneridge,  
Cupertino  
Call John Hall 415/532-1115  
or call the FIG Hotline:  
408/277-0668

**Stockton Chapter**  
Call Doug Dillon  
209/931-2448

### • COLORADO

**Denver Chapter**  
Monthly, 1st Mon., 7 p.m.  
Cliff King  
303/693-3413

### • CONNECTICUT

**Central Connecticut Chapter**  
Call Charles Krajewski  
203/344-9996

### • FLORIDA

**Orlando Chapter**  
Every two weeks, Wed., 8 p.m.  
Call Herman B. Gibson  
305/855-4790

**Southeast Florida Chapter**  
Monthly, Thurs., p.m.  
Coconut Grove area  
Call John Forsberg  
305/252-0108

**Tampa Bay Chapter**  
Monthly, 1st. Wed., p.m.  
Call Terry McNay  
813/725-1245

### • GEORGIA

**Atlanta Chapter**  
Monthly, 3rd Tues., 6:30 p.m.  
Computone Cotilion Road  
Call Nick Hennenfent  
404/393-3010

### • ILLINOIS

**Cache Forth Chapter**  
Call Clyde W. Phillips, Jr.  
Oak Park  
312/386-3147

**Central Illinois Chapter**  
Urbana  
Call Sidney Bowhill  
217/333-4150

**Fox Valley Chapter**  
Call Samuel J. Cook  
312/879-3242

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

### • INDIANA

**Central Indiana Chapter**  
Monthly, 3rd Sat., 10 a.m.  
Call John Oglesby  
317/353-3929

**Fort Wayne Chapter**  
Monthly, 2nd Tues., 7 p.m.  
IPFW Campus  
Rm. 138, Neff Hall  
Call Blair MacDermid  
219/749-2042

### • IOWA

**Iowa City Chapter**  
Monthly, 4th Tues.  
Engineering Bldg., Rm. 2128  
University of Iowa  
Call Robert Benedict  
319/337-7853

**Central Iowa FIG Chapter**  
Call Rodrick A. Eldridge  
515/294-5659

**Fairfield FIG Chapter**  
Monthly, 4th day, 8:15 p.m.  
Call Gurdy Leete  
515/472-7077

### • KANSAS

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

### • LOUISIANA

**New Orleans Chapter**  
Call Darryl C. Olivier  
504/899-8922

### • MASSACHUSETTS

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

### • MICHIGAN

**Detroit/Ann Arbor area**  
Monthly, 4th Thurs.  
Call Tom Chrapkiewicz  
313/322-7862 or 313/562-8506

### • MINNESOTA

**MNFIG Chapter**  
Even Month, 1st Mon., 7:30 p.m.  
Odd Month, 1st Sat., 9:30 a.m.  
Vincent Hall Univ. of MN  
Minneapolis, MN  
Call Fred Olson  
612/588-9532

### • MISSOURI

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

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

### • NEVADA

**Southern Nevada Chapter**  
Call Gerald Hasty  
702/452-3368

### • NEW HAMPSHIRE

**New Hampshire Chapter**  
Monthly, 1st Mon., 6 p.m.  
Armtec Industries  
Shepard Dr., Grenier Field  
Manchester  
Call M. Peschke  
603/774-7762

### • NEW MEXICO

**Albuquerque Chapter**  
Monthly, 1st Thurs., 7:30 p.m.  
Physics & Astronomy Bldg.  
Univ. of New Mexico  
Jon Bryan  
Call 505/298-3292

### • NEW YORK

**FIG, New York**  
Monthly, 2nd Wed., 7:45 p.m.  
Manhattan  
Call Ron Martinez  
212-749-9468

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

**Syracuse Chapter**  
Monthly, 3rd Wed., 7 p.m.  
Call Henry J. Fay  
315/446-4600

### • OHIO

**Akron Chapter**  
Call Thomas Franks  
216/336-3167

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

**Cleveland Chapter**  
Call Gary Bergstrom  
216/247-2492

**Cincinnati Chapter**  
Call Douglas Bennett  
513/831-0142

**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**

**Central Oklahoma Chapter**  
Monthly, 3rd Wed., 7:30 p.m.  
Health Tech. Bldg., OSU Tech.  
Call Larry Somers  
2410 N.W. 49th  
Oklahoma City, OK 73112

• **OREGON**

**Greater Oregon Chapter**  
Monthly, 2nd Sat., 1 p.m.  
Tektronix Industrial Park  
Bldg. 50, Beaverton  
Call Tom Almy  
503/692-2811

• **PENNSYLVANIA**

**Philadelphia Chapter**  
Monthly, 4th Sat., 10 a.m.  
Drexel University, Stratton Hall  
Call Melanie Hoag or Simon Edkins  
215/895-2628

• **TENNESSEE**

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

• **TEXAS**

**Austin Chapter**  
Contact Matt Lawrence  
P.O. Box 180409  
Austin, TX 78718

**Houston Chapter**  
Call Dr. Joseph Baldwin  
713/749-2120

**Periman Basin Chapter**  
Call Carl Bryson  
Odessa  
915/337-8994

• **UTAH**

**North Orem FIG Chapter**  
Contact Ron Tanner  
748 N. 1340 W.  
Orem, UT 84057

• **VERMONT**

**Vermont Chapter**  
Monthly, 3rd Mon., 7:30 p.m.  
Vergennes Union High School  
Rm. 210, Monkton Rd.  
Vergennes, VT  
Call Don VanSyckel  
802/388-6698

• **VIRGINIA**

**First Forth of Hampton Roads**  
Call William Edmonds  
804/898-4099

**Potomac Chapter**  
Monthly, 2nd Tues., 7 p.m.  
Lee Center  
Lee Highway at Lexington St.  
Arlington, VA  
Call Joel Shprentz  
703/860-9260

**Richmond Forth Group**  
Monthly, 2nd Wed., 7 p.m.  
154 Business School  
Univ. of Richmond  
Call Donald A. Full  
804/739-3623

• **WISCONSIN**

**Lake Superior FIG Chapter**  
Monthly, 2nd Fri., 7:30 p.m.  
University of Wisconsin  
Superior  
Call Allen Anway  
715/394-8360

**Milwaukee Area Chapter**  
Call Donald H. Kimes  
414/377-0708

**MAD Apple Chapter**  
Contact Bill Horzon  
129 S. Yellowstone  
Madison, WI 53705

**FOREIGN**

• **AUSTRALIA**

**Melbourne Chapter**  
Monthly, 1st Fri., 8 p.m.  
Contact Lance Collins  
65 Martin Road  
Glen Iris, Victoria 3146  
03/29-2600

**Sydney Chapter**  
Monthly, 2nd Fri., 7 p.m.  
John Goodsell Bldg.  
Rm. LG19  
Univ. of New South Wales  
Sydney  
Contact Peter Tregear  
10 Binda Rd., Yowie Bay  
02/524-7490

• **BELGIUM**

**Belgium Chapter**  
Monthly, 4th Wed., 20:00h  
Contact Luk Van Loock  
Lariksdruff 20  
2120 Schoten  
03/658-6343

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

• **CANADA**

**Alberta Chapter**  
Call Tony Van Muyden  
403/962-2203

**Nova Scotia Chapter**  
Contact Howard Harawitz  
227 Ridge Valley Rd.  
Halifax, Nova Scotia B3P2E5  
902/477-3665

**Southern Ontario Chapter**  
Quarterly, 1st Sat., 2 p.m.  
General Sciences Bldg., Rm. 312  
McMaster University  
Contact Dr. N. Solntseff  
Unit for Computer Science  
McMaster University  
Hamilton, Ontario L8S4K1  
416/525-9140 ext. 3443

**Toronto FIG Chapter**  
Contact John Clark Smith  
P.O. Box 230, Station H  
Toronto, ON M4C5J2

• **COLOMBIA**

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

• **ENGLAND**

**Forth Interest Group — U.K.**  
Monthly, 1st Thurs.,  
7p.m., Rm. 408  
Polytechnic of South Bank  
Borough Rd., London  
D.J. Neale  
58 Woodland Way  
Morden, Surrey SM4 4DS

• **FRANCE**

**French Language Chapter**  
Contact Jean-Daniel Dodin  
77 Rue du Cagire  
31100 Toulouse  
(16-61)44.03.06

• **GERMANY**

**Hamburg FIG Chapter**  
Monthly, 4th Sat., 1500h  
Contact Horst-Gunter Lynsche  
Common Interface Alpha  
Schanzenstrasse 27  
2000 Hamburg 6

• **HOLLAND**

**Holland Chapter**  
Contact: Adriaan van Roosmalen  
Heusden Houtsestraat 134  
4817 We Breda  
31 76 713104

**FIG des Alpes Chapter**  
Contact: Georges Seibel  
19 Rue des Hirondelles  
74000 Annecy  
50 57 0280

• **IRELAND**

**Irish Chapter**  
Contact Hugh Doggs  
Newton School  
Waterford  
051/75757 or 051/74124

• **ITALY**

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

• **JAPAN**

**Japan Chapter**  
Contact Toshi Inoue  
Dept. of Mineral Dev. Eng.  
University of Tokyo  
7-3-1 Hongo, Bunkyo 113  
812-2111 ext. 7073

• **NORWAY**

**Bergen Chapter**  
Kjell Birger Faeraas  
Hallskalet 28  
Ulset  
+47-5-187784

• **REPUBLIC OF CHINA**  
**R.O.C.**

Contact Ching-Tang Tzeng  
P.O. Box 28  
Lung-Tan, Taiwan 325

• **SWEDEN**

**Swedish Chapter**  
Hans Lindstrom  
Gothenburg  
+46-31-166794

• **SWITZERLAND**

**Swiss Chapter**  
Contact Max Hugelshofer  
ERNI & Co., Elektro-Industrie  
Stationsstrasse  
8306 Bruttisellen  
01/833-3333

**SPECIAL GROUPS**

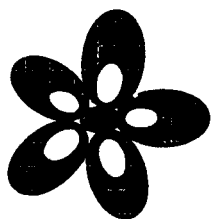
**Apple Corps Forth Users Chapter**  
Twice Monthly, 1st &  
3rd Tues., 7:30 p.m.  
1515 Sloat Boulevard, #2  
San Francisco, CA  
Call Robert Dudley Ackerman  
415/626-6295

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

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

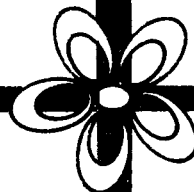
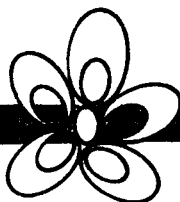
# HOLIDAY SPECIALS !!

See Our Order Form Inside for Details

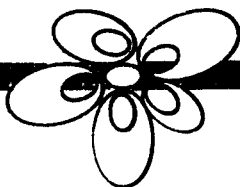


FORTH DIMENSIONS  
BACK VOLUMES

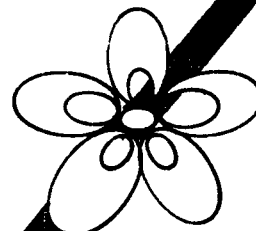
FORTH MODEL  
LIBRARY



FORML  
CONFERENCE  
PROCEEDINGS



FORTH & FORTH  
83 79  
STANDARDS



DR. DOBB'S  
JOURNAL

## FROM THE FORTH INTEREST GROUP

### FORTH INTEREST GROUP

P. O. Box 8231  
San Jose, CA 95155

BULK RATE  
U.S. POSTAGE  
PAID  
Permit No. 3107  
San Jose, CA