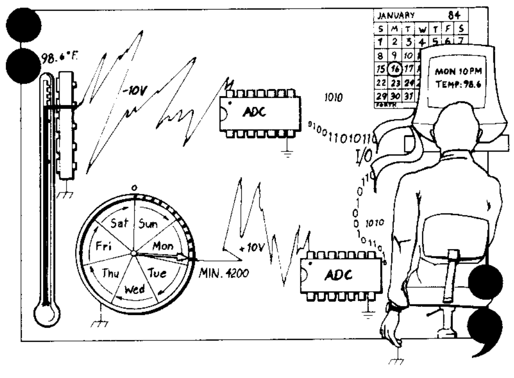
FORTH Dimensions

Volume 5, Number 5
January/February 1984
$2.50

Data Acquisition



Is There A Consultant In Your Future?

"If you require an expert in a given area you have three choices: you can hire one, you can cultivate one, or you can find a consultant who is already an expert. The first two choices are reasonable only if this expertise is required on a very long term basis. Cultivating an expert can be a particularly frustrating experience since it usually takes one to make one. A consultant, on the other hand, can supply his expertise only when and in the amount needed. It will cost quite a bit more per hour to use a consultant expert but the long run savings, in this case, are dramatic ..."

"Guide to Using Consultants"
Inner Access Corporation 1984

We Wrote The Book!

For your free copy of *"Guide to Using Consultants"* call or write:

517-K Marine View, Belmont, CA 94002
OR
P.O. Box 888, Belmont, CA 94002

(415) 591-8295

Inner Access Corporation

A computer software and hardware consulting firm for business and industry

• PROCESS CONTROL • DATABASE MANAGEMENT • DATA ACQUISITION
• GRAPHICS • HARDWARE DESIGN • SELECT/SORT/MERGE • SOFTWARE SUPPORT
• TURN-KEY SYSTEMS • FORM DRIVEN SOFTWARE • 68000 • Z8000 • 8086/88 • NCS 800
• 8048/49 • Z80/8085 • FORTH • SYSTEMS SOFTWARE • SPECIALIZED EDITORS
• MULTI PROCESSOR SOFTWARE • AUTOMATED DESIGN • MODELLING
• SPECIFICATION • DOCUMENTATION • TRAINING • MICROPROCESSOR APPLICATIONS

FORTH Dimensions
Published by Forth Interest Group
Volume V, No. 5
January/February 1984
Editor
Marlin Ouverson
Publisher
Roy C. Martens
Typesetting/Production
LARC Computing, Inc.
Cover Art
Al McCahon

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

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

Letters to the Editor

On Curses and Communication

Dear FIG:

As you responded to one reader, we can't impose a standard language on everyone, and anyone should be free to deviate from the standard as he desires. On the other hand, since we do have a standard, can't we urge authors to at least make reference to the current standard in their program listings? How about a special Forth word to call down a curse of disk errors on authors who fail to indicate the instances where their listing deviates from the standard? I presume such a list would be small compared to the main program.

How many Forth articles have we all seen wherein the author simply mentions that his program is written in *xyz*-Forth, with never a mention of those strange words which have never been seen in any standard and, frequently, are not found in any of the six Forth references I use. In an ideal world, it should be possible for one at least to derive the intent of any Forth program presented without having to buy that specific version. As it stands, the comment that "...this program is written in PC-Forth..." or FIG-Forth, MVP-Forth, *etc.*, is simply an

(Continued on next page)

Editorial
Cover Ups

In the last issue, I promised that, this time, we would have a review of this year's FORML conference. Here it is, as expected! The amount of available space did not permit us to cover all the sessions, or even each talk in any one session. The selection process was very nearly arbitrary, because so many papers of outstanding quality were presented. But read for yourself, then make plans to attend next year. It will be the Friday, Saturday and Sunday after Thanksgiving. Look for details, as they become available, in these pages.

Have you been paying attention to our cover art recently? The last one arrived with a handwritten apology to Renaissance painter, engraver and designer Albrecht Durer (and whoever finds both the missing umlauts in this issue gets Proofreader-of-the-Month award — our typesetter doesn't like 'em). Artist Al McCahon has been synthesizing such designs for some time, taking apparently disparate technical elements and finding a unified artistic concept. Some time back, I had the opportunity to listen to a group of Forth people trying to figure out the hidden meaning of one of our covers. (My favorite line of the evening: "But why is the dog looking in that direction? It must *mean* something!") The most obscure visual references were objects of extended conjecture — could they really be just design elements? We aren't giving away any secrets, but are willing to print interesting letters to the editor which attempt to decipher what was in the artist's mind. Mainly, this is an indirect way to thank Al for the nice touch he has added to our covers.

You may recall that, earlier this year, *Forth Dimensions* had the opportunity to publish a lengthy interview with Charles Moore. It has since been pointed out that there are a number of other people who also have figured prominently in the development of Forth. We are laying plans now for a series of interviews with key Forth people around the world. May we have your nominations for interviewees? The next one is slated to be Bill Ragsdale, President and one of the founders of the Forth Interest Group. Look for it!

Bill's welcome contribution to this issue is his article on "Timekeeping in Forth." A glance at his code will reveal what we believe is the first published example of shadow screens. Note that each shadow is linked to the source screen, and each comment line corresponds visually to the line it documents. Let us know how you like the concept and its execution.

Finally, don't panic when you notice the absence of a list of Forth vendors in this issue. It has been retired for extensive updating, with the assistance of the vendors themselves, who will be receiving correspondence from us about the project. Over time, some inaccuracies had slipped in, and comments we received also asked for a more readable format. While the list is out for repair, copies of it will continue to be mailed to any who write or call for a copy.

See you next issue!

—Marlin Ouverson
Editor

invitation to skip to the next article if that is not the version you use. Is this communication? Doesn't Forth suffer enough from lack of communication with the outside world without creating isolated islands within itself?

And wherein has the Forth community benefitted from a standard if only those articles written in your own version are readable? I believe that unless better communication is established among the diverse users of Forth, it will remain a very interesting language of minor importance. I recognize that Forth has grown in popularity, but I observe much less growth in accomplishment as compared to some other languages, and I believe that this is in part due to the difficulty of the exchange of ideas.

Yours truly,

James H. Ramsay
Box 1015
Mathews, VA 23109

You are, of course, correct that we must be vitally concerned with communication. And while we cannot dictate which Forth dialect one uses (nor do we wish to), Forth Dimensions *can indeed contribute to the legibility of non-standard code. For example, participants at FORML were interrogated in detail about coding conventions which, when used, will greatly enhance the readability of anyone's Forth. We believe that where function is adequately apparent, most problems with legibility will disappear. These and other measures not strictly tied to any specific dialect will be introduced in these pages as they become available to us. In the absence of absolute transportability, we are aiming for 100% comprehensibility. With the help of readers like yourself, we are learning how best to accomplish this. Thanks to all who have written on this subject, and keep the letters coming! —Ed.*

The Minimal Nucleus

Dear Editor:

After you have Forth up and running, you may wish to experiment with ways to either speed up the nucleus or minimize its size. It is particularly easy to do this if you have a cross compiler and the source to the Forth nucleus. Even though saving a few bytes here and there may not let you write another thousand lines of code, it is satisfying (and Forth-like) to know things are as efficient as possible.

Assuming you don't remove nucleus words, there are general techniques to save code space in the nucleus:

Reuse Forth code for duplicated definitions. In the current version, **R@** and **I** execute identically, as do **0=** and **NOT**, and **CFA** and **2-**. It is possible to save code by allowing the second definition to reuse the first definition's code. The technique is:

```
CODE R@ ' I HERE CFA ! END-CODE
```

This makes the length of **R@** only seven bytes instead of seventeen. This technique will not work if there is a separate loop stack or if **NOT** is redefined to mean the one's complement.

0> can use part of **0<**'s code. **TRAVERSE** can be coded in **CODE**, not only running more quickly, but taking less code as well.

(Continued on page 17)

1 proFORTH COMPILER

8080/8085, Z80 VERSIONS

- SUPPORTS DEVELOPMENT FOR DEDICATED APPLICATIONS
- INTERACTIVELY TEST HEADERLESS CODE
- IN-PLACE COMPILATION OF ROMABLE TARGET CODE
- MULTIPLE, PURGABLE DICTIONARIES
- FORTH-79 SUPERSET
- AVAILABLE NOW FOR TEKTRONIX DEVELOPMENT SYSTEMS — $2250

2 MICROPROCESSOR-BASED PRODUCT DESIGN

- SOFTWARE ENGINEERING
- DESIGN STUDIES — COST ANALYSIS
- ELECTRONICS AND PRINTED CIRCUIT DESIGN
- PROTOTYPE FABRICATION AND TEST
- REAL-TIME ASSEMBLY LANGUAGE/proFORTH
- MULTITASKING
- DIVERSIFIED STAFF

MICROSYSTEMS, INC.

(818) 577-1471

2500 E. FOOTHILL BLVD., SUITE 102, PASADENA, CALIFORNIA 91107

Introduction to Data Acquisition

Michael A. Perry
Berkeley, California

Computers are often used to analyze, display or control physical processes. Acquiring data means taking measurements and turning them into numbers, which a computer can manipulate. There are three steps in this process. First, the physical value is converted into a voltage by a transducer, then the voltage is turned into a number by an analog-to-digital converter (ADC), then the number is saved by the computer.

For an example, take monitoring a temperature. First the signal, the temperature, is converted into an electrical signal by a transducer. A transducer is anything which converts energy into voltages by a circuit connected to a thermistor. A thermistor is a resistor whose value changes sharply with its temperature. A typical circuit might produce a signal of ten milli-volts per degree centigrade, so that zero degrees would give 0.00 volts and 100 degrees would give 1.00 volts.

This voltage is then converted to a number by an ADC. Analog-to-digital converters produce a number which represents the input signal as a fraction of the full scale value. ADCs are available in different precisions, and many allow the user to select the full-scale input voltage range. For example, one fairly typical ADC is the Analog Devices 574, which allows full-scale ranges of -5 to +5, or -10 to +10 volts bipolar; or 0 to +10, or 0 to +20 volts unipolar; and converts the input into a twelve-bit binary number, signed for bipolar inputs or unsigned for unipolar. Assume we have amplified the temperature signal by ten, so that 100 degrees is 10.0 volts, and we set the ADC to a full-scale input range of 0 to +10 volts. The twelve-bit output will range from zero (for zero volts) to 4095 (for 4095/4096 * 10 volts). Converters with eight- or twelve-bit resolution are commonly available now, and sixteen-bit converters are becoming more available. High resolution, high accuracy and high speed are all more expensive.

Finally, the number is read and saved by the computer. The ADC will usually be controlled and read by writing and reading I/O ports or memory addresses. In the simplest case, all of the data you intend to collect will fit in memory, and all the computer need do is to read the converter at regular intervals and store the numbers at successive addresses in memory.

So far, this should sound pretty easy. It is. The first complication that arises is that it is generally desirable to take readings at very regular intervals. The only way to be sure of the interval is to set a clock to cause an interrupt, and take the readings in the interrupt routine. The second is that large amounts of data will not fit in memory, and so must be written to the disk. While writing to the disk, the processor is busy and cannot process interrupts. A DMA disk controller is required if you must respond to interrupts quickly and you will be writing to the disk.

A system which can respond to external events promptly is called a real-time system. Actual systems are often called upon to provide fancy graphic displays of the data as it is collected, and even to interpret the data and use it to control the process being measured.

C64-FORTH/79
New and Improved for the Commodore 64

C64-FORTH/79™ for the Commodore 64-$99.95

- New and improved FORTH-79 implementation with extensions.
- Extension package including lines, circles, scaling, windowing, mixed high res-character graphics and sprite graphics.
- Fully compatible floating point package including arithmetic, relational, logical and transcendental functions.
- String extensions including LEFT$, RIGHT$, and MID$.
- Full feature screen editor and macro assembler.
- Compatible with VIC peripherals including disks, data set, modem, printer and cartridge.
- Expanded 167 page manual with examples and application screens.
- "SAVE TURNKEY" normally allows application program distribution without licensing or royalties.

(Commodore 64 is a trademark of Commodore)

TO ORDER

-Disk only.
-Check, money order, bank card, COD's add $1.65
-Add $4.00 postage and handling in USA and Canada
-Mass. orders add 5% sales tax
-Foreign orders add 20% shipping and handling
-Dealer inquiries welcome

PERFORMANCE MICRO PRODUCTS

770 Dedham Street,
Canton, MA 02021
(617) 828-1209

Timekeeping in Forth

William F. Ragsdale
Hayward, California

I often hear the lament, "While learning Forth, I'd like to see some real-life applications. I'm tired of rewriting editors!" Well, here's what the doctor ordered. In seven screens, you will see how to:

- input text
- convert time bases
- handle interactive error prompting
- maintain a real-time clock
- control time-dependent execution
- perform string compilation and lookup
- use custom output formatting

These concepts are detailed in the source screens and text. Be aware that their brevity may conceal their utility. As you understand and expand any of them, your Forth toolkit becomes larger; more ambitious applications are easier.

What

Recently, I was developing an application for data collection and machine control. One of the data structures was a weekly schedule of event times. Each schedule time represented one minute and was followed by a fixed-length record.

Since these records were held in time sequence, I needed a compact representation that was easy to scan and sort. One possibility was minutes, hours and day number held in three fields (words or bytes). This form needed a multifield comparison for sorting.

I then considered other formats, starting with the smallest unit. Sixty minutes in an hour; then 60*24 hours or 1440 minutes in a day; then 1440*7 or 10080 minutes in a week. How convenient! A sixteen-bit positive integer representing the minute of the week — MOW.

Why

Like most programmers, after selecting the concept, I searched for justification of the use of MOW: 1. Compact storage — an integer held in two bytes, using fifteen of sixteen bits. 2. Same representation on the stack or in memory. 3. Easy to sort and insert into tables. 4. Can be operated upon by numerical comparison operators >, < and =. 5. Simple conversion operators to and from ASCII text.

Conventions

Time input and output of this application are designed for the conventions of twenty-four hour timekeeping. Please see table one.

Day	English	MOW	24-Hr.
Sunday	12:00M	0	2400
	12:01AM	1	0001
	12:59AM	59	0059
	1:00AM	60	0100
	11:59AM	719	1159
	12:00N	720	1200
Sunday	11:59PM	1439	2359
Monday	12:00M	1440	2400
Saturday	11:59PM	10079	2359

Some contemporary applications of keeping time use 0000 hours for midnight. If this format is desired, you may remove the flag generation and subsequent adjustment in **M/H/D** and **>MOW**. You might also wish to extend the **DAY-NAMES** table to include another day name of **HOL** for holidays. In this case, **MOW** would run from 0 thru 11519.

Commentary

The Forth words of the application are grouped as follows:

Conversion, scaling	42
Text output	43
Time-keeping primitives	44
High-level timekeeping	45
Text primitives	46
Input text high level	47
Sample use	48

Block 42 presents the conversion word **M/H/D** which breaks a **MOW** value into minutes, hours and days. The hour is adjusted for **0 MOW** being 2400 hours. **>MOW** is the complementary word which returns the **MOW** integer.

The output operators are next most important as you will need them for testing. Day numbers are converted from the text in **DAY-NAMES**. **<#HHMM#>** uses the usual Forth numerical output formatting primitives to convert hours and minute values to ASCII text. **.DHM** displays the day and time. Could you suggest an appropriate English word name?

Shadow 162

```
   The conversion words convert to and from the MOW,
minute of the week.  M/H/D  returns the minute, hour, and week
from the minute of the week.  DAY-NUMBER just gives the day
of the week, with Sunday being zero.

  >MOW combines the minute, hour, and day into a single
integer minute of the week, in the range 0 to 10079. The
convention used is that 12:00 Midnight on Sunday morning is
minute zero. 11:59 PM on Saturday night is MOW 10079.

  The current minute of the week is maintained in the variable
MOW. The RUN word rolls over the day and week properly.
```

Blk 42

```
( Timekeeping in Forth                          83 dec 27 WFR )
( The following code follows Forth-83. The only non-standard
  words are THRU which is a controlled reference word and
  ?KEY which returns a false flag until any terminal key is
  pressed. )                                          DECIMAL

( conversion words )
: M/H/D        ( mow --- min\hr\day  in range 0100 to 2459 hrs )
    DUP 1440 MOD 0= >R   60 /MOD  24 /MOD
    R> IF >R 24 + R> THEN ;
: DAY-NUMBER       60 / 24 / ;      ( mow —— 0..6 day number )

: >MOW                                     ( min\hr\day --- mow )
     OVER 24 = IF >R 0= R> THEN  24 * +  60 * + ;

BLK @ 1+ DUP 5 + THRU
```

Shadow 163

```
  DAY-NAMES is an array in the dictionary which holds the names
of the seven days of the week as three letter entries, with
count. The array is first named with 27 bytes for the actual
names. Then the text is inserted into the array and the word
INSERT forgotten.

  DAY$ leaves the address and count from the day number on the
stack.  .DAY displays the day from its number.

  <#HHMM#> converts the hour and minute number into text in
memory. It uses the usual numerical output formatting words
<# # #> which are combined forming the name of this new word.
.HM displays the hour and minute after the D/H/M conversion.

  Finally, .DHM displays the day hour and minute according to
the conventions for 24 hour time.
```

Blk 43

```
( Output words                                  83 dec 17 WFR )
CREATE DAY-NAMES   28 ALLOT  ( storage for 7 three leter names)
: INSERT                          ( into address, compile strings )
     0 DO   BL WORD   OVER I 4 * + 4 CMOVE LOOP DROP ;
DAY-NAMES 7 INSERT SUN MON TUE WED THU FRI SAT   FORGET INSERT

: DAY$      4 * DAY-NAMES + COUNT ;     ( day# —— addr\count )

: .DAY     DAY$ TYPE ;              ( day# —— display the name )

: <#HHMM#>      ( min\hr —— addr count  numerical conversion )
    SWAP  <# 0 # # DROP DROP   0 # # #> ;

: .HM    <#HHMM#>   TYPE ;          ( min\hr —— display value )

: .DHM   M/H/D  .DAY SPACE .HM ;        ( display day and time )
```

If you aren't actually keeping time with this application, blocks 44 and 45 may be ignored. Block 44 assumes an interrupt and machine code which will increment **JIFFY** at a 60 Hz rate. Your clock interface will probably differ. **TICK** and **TOCK** continue the time-keeping process over seconds and minutes. The style here is typical Forth. The interrupt routine should be short. (How about **JIFFY INC, RTI,**?) Limits and carries should be at high level, as with **RUN, TICK** and **TOCK.**

I've broken out the words **EACH-SECOND** and **EACH-MINUTE** so you may include your code for specific data collection or control actions. These actions would generally be conditional on **MOW** and a table of specific times.

GET-TIME and **GET-DAY** on block 47 give the operator a simple prompt. If he fails to respond with the appropriate input, an additional prompt is given to clarify the expected format. The usual action occurs between **BEGIN** through to **WHILE**. The error prompt occurs from **WHILE** to **REPEAT**.

Block 46 gives text input and comparison, so we may look up the user's input in **DAY-NAMES**. After completing this application, I wanted to compile times from disk (block 48). I found a design problem, as the user cannot respond to error re-prompting. Thus **?BLK** was inserted to alert the operator. If the input stream is being taken from the disk (**BLK** contains a non-zero val-

(Continued)

Shadow 164

```
    The variables  JIFFY, SECONDS,  MOW,  and DOY allow the
system to watch the passage of time.   The contents of
of MOW are minutes of the week, in the range 0..10079.

    Here is typical 6502 assembly code for the interrupt routine
which has a memory mapped port at CLOCK-PORT which interrupts
60 times a second (.0166 sec.).  These interrupts are counted
into JIFFY at the interrupt level, which is later examined by
the high level code in TICK.  This is typical Forth technique
to keep the interrupt code short and non-conditional, with
testing and range limits done in high level.

   TICK is called more often than once a second and leaves a
true flag at the new second.  TOCK is called each second and
leaves a true flag at the new minute.
```

Blk 44

```
( Timekeeping                                     83 dec 27 WFR )
CREATE JIFFY     0 ,         ( holds increments of 60 Hz time )
CREATE SECONDS   0 ,    ( accumulates seconds till one minute )
CREATE MOW       0 ,                ( holds minute of the week )
CREATE DOY       0 ,                       ( day of the year )
65280 CONSTANT CLOCK-PORT               ( interrupt flag in memory )

                                                     ASSEMBLER
( the interrupt vector for the 60 Hz interrupt points here    )
CLOCK-PORT LDA,  01 # AND,  0= NOT IF, JIFFY INC, THEN,  RTI,

: TICK                        ( detectes new second with true flag )
    JIFFY @ 59 > DUP      IF -60 JIFFY +!  1 SECONDS +!  THEN ;

: TOCK   ( once a second routine, leaves true for new minute )
    SECONDS @ 59 > DUP   IF -60 SECONDS +!   1 MOW  +!  THEN ;
```

Shadow 165

```
    You may insert your code into the routines EACH-SECOND
and EACH-MINUTE.  Each will be executed at the appropriate
time by RUN.

    RUN is the prototype for the outer control loop which can
run a process, collect data or wait until a specified time.
It assumes that an interrupt is incrementing JIFFY at a 60 Hz
rate. RUN keeps using TICK to watch for seconds change and
TOCK to watch for minutes changes. If you keep time over the
period of years, then RUN must determine leap years and maintain
DOY and YEAR correctly.  The modification would be done at the
point commented as 'day'.
    If RUN is never to stop, just replace ?KEY with the
value zero "0".
```

Blk 45

```
( Project time control                            83 dec 17 WFR )
: EACH-SECOND   ( insert your action here )   ;

: EACH-MINUTE   ( insert your action here )   ;



: RUN                      ( outer loop of time dependent process )
BEGIN  TICK  IF  ( second ticked over ) EACH-SECOND
       TOCK  IF  MOW @ 10080     =  IF 0 MOW ! THEN
                 MOW @ 1440 MOD 0=  IF 1 DOY +! ( day  ) THEN
                 EACH-MINUTE  ( since minute tocked over )
              THEN THEN
     ?KEY UNTIL   ;
```

Shadow 166

```
    ?QUERY is similar to the Forth 83 reserved word QUERY
which accepts text from the operators terminal. In this
case, ?QUERY only does so if input is already coming from the
terminal. If input is being taken from the disk (blk non zero)
then text continues from the disk.

    =TEXT does a simple letter by letter comparison over the
length of the second string. It leaves a true flag on a good
match.

    N.TH-TEXT is used to scan the table of day names.
Input parameters are the input text addr, the table of possible
choices, and the number of entries that may be scanned in the
table.  N.TH-TEXT scans for the table of choices and returns
the position number in the table if found or else -1 (false).
Note that N.TH-TEXT and =TEXT are quite general in use.
```

Blk 46

```
( Operator input                                  83 dec 17 WFR )

: ?QUERY                 ( addr count ——  obtain terminal input )
    BLK @ 0= IF   0 >IN !    TIB 20 EXPECT THEN  ;

: =TEXT       ( unknown text, table address —— true for match )
    >R >R -1 R> R>     COUNT OVER + SWAP
  DO 1+ I C@ OVER C@ - IF SWAP 0= SWAP LEAVE THEN   LOOP DROP ;

: N.TH-TEXT                  ( addr,  addr, entry count —— flag )
   >R >R >R -1 R> R> R> 0 DO   OVER OVER =TEXT
   IF DROP DROP DROP I 0 0 LEAVE THEN  COUNT + LOOP DROP DROP ;

: ?BLK               ( give error message and abort if from disk )
     BLK @ ABORT" Time or date error from disk "  ;
```

Shadow 167

```
    GET-TIME prompts the operator for the time of day in the
24 hour time system. Input is error checked for decimal
digits and a range of 0001 thru 2400.  The two WHILE clauses
give specific suggestion for correct input form.

    GET-DAY prompts the operator for the day name. Input must
be a three letter abbreviation. In case of error, the operator
is given an example of correct input.

    SETUP-TIME combines GET-TIME and GET-DAY to initialize
the variable MOW (minute of the week). At this point the
RUN routine must be operating to keep the time current.
```

Blk 47

```
( Operator input                                  83 dec 27 WFR )
: GET-TIME                               ( prompt —— min, hour )
  BEGIN   BEGIN  CR ." What is the time?  "  ?QUERY
           0 0  BL WORD CONVERT C@ BL - ( digits till blank? )
          WHILE ?BLK  DROP DROP  ." use 4 digits" REPEAT
      DROP   0001 OVER >  OVER 2400 > OR  ( 0001..2400 range )
   WHILE ?BLK DROP ." in range 0001 to 2400 " REPEAT  100 /MOD ;

: GET-DAY                      ( prompt operator —— day number )
   BEGIN  CR ." What is the day of the week?   "  ?QUERY
          BL WORD   DAY-NAMES 7  N.TH-TEXT  DUP -1 =
          WHILE ?BLK DROP 3 SPACES ." input in the form: MON"
     REPEAT ;

: SETUP-TIME                     ( loads mow from operator input )
   GET-TIME  GET-DAY  >MOW  MOW !   0 SECONDS !  ;
```

FOR TRS-80 MODELS 1, 3 & 4
IBM PC, XT, AND COMPAQ

The MMSFORTH System. Compare.

- The speed, compactness and extensibility of the MMSFORTH total software environment, optimized for the popular IBM PC and TRS-80 Models 1, 3 and 4.
- An integrated system of sophisticated application programs: word processing, database management, communications, general ledger and more, all with powerful capabilities, surprising speed and ease of use.
- With source code, for custom modifications by you or MMS.
- The famous MMS support, including detailed manuals and examples, telephone tips, additional programs and inexpensive program updates, User Groups worldwide, the MMSFORTH Newsletter, Forth-related books, workshops and professional consulting.

mmsFORTH

A World of Difference!

- Personal licensing for TRS-80: $129.95 for MMSFORTH, or "3+4TH" User System with FORTHWRITE, DATAHANDLER and FORTHCOM for $399.95.
- Personal licensing for IBM PC: $249.95 for MMSFORTH, or enhanced "3+4TH" User System with FORTHWRITE, DATAHANDLER-PLUS and FORTHCOM for $549.95.
- Corporate Site License Extensions from $1,000.

If you recognize the difference and want to profit from it, ask us or your dealer about the world of MMSFORTH.

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

ue), then execution aborts with an error message.

Block 48 illustrates how you may build a table of **MOW** event times. **EVENTS** is given the number of times to request input, in this case ten. Then the **EVENT-TABLE** is filled in. The command **10 EVENTS** should work either from disk or the terminal. In either case ten prompts are displayed. You might want to make conditional all prompting in **GET-TIME** and **GET-DAY**. If data or commands are to be included in **EVENT-TABLE** then make appropriate additions just before **LOOP** in **EVENTS**.

Standards Consideration

This application is written in Forth-83. This dialect was chosen because its control of the input stream is most precisely specified, and this Standard should dominate over the next several years.

The only non-standard words are **THRU** (which compiles a sequence of blocks) and **?KEY** (which returns a terminal key value or zero, lacking an operator input). If running on an earlier dialect of Forth, the definitions of **?QUERY**, **?BLK** and the use of **WORD**, **TIB**, **LEAVE** and **CREATE** will have to be altered.

A possible problem exists in **TICK** in the phrase **-60 JIFFY +!**. If the interrupt incrementing **JIFFY** occurs just before the **+!** writes into **JIFFY**, the increment may be lost. You should consider interrupt protecting the machine code for **+!**.

Conclusion

We have seen in this application how Forth may be used to build a set of application-specific words for time-keeping. Their utility is obvious, yet complexity is low. Presently, over 500 stores have operating schedules controlled by **MOW** with **JIFFY**s **TICK**ing and **TOCK**ing.

About the Author

Bill Ragsdale is the President of the Forth Interest Group. Bill has previously authored articles on the Forth Standards development, disk storage allocation, higher level defining words and the **ONLY** concept for vocabularies, as well as the book "fig-FORTH Model and Installation Guide". Memberships include the Forth Standards Team (Chairman 1980-1982), IEEE and ACM. Bill is the President of an electronics manufacturer and a graduate of the University of California at Berkeley in Electrical Engineering.

```
                         Shadow 168

   EVENTS builds a given number of time events into a table.
In this example, the EVENT-TABLE holds ten times.

   In an application, you might follow each event time with
several bytes holding data or commands to be executed at the
specific time.

   DISPLAY-EVENTS just plays back the event times of this
example for testing.
```

```
                     Blk 48

( Time in data tables                        83 dec 17 WFR )
: EVENTS            ( compile n times into memory, addr\n -- )
    0 DO GET-TIME GET-DAY  >MOW , LOOP ;

CREATE EVENT-TABLE
   10 EVENTS      0100 SUN    1400 SUN    0800 MON
                  1245 MON    2400 TUE    1500 TUE
                  0945 WED    0945 THU    1400 FRI
                  2300 SAT

: DISPLAY-EVENTS          ( output the ten times, for testing )
   EVENT-TABLE 10 0 DO  CR   DUP @ .DHM  2+  LOOP  DROP ;


```

TOTAL CONTROL:

***FORTH:* FOR Z-80®, 8086, 68000, and IBM® PC**

Complies with the New 83-Standard

GRAPHICS • GAMES • COMMUNICATIONS • ROBOTICS
DATA ACQUISITION • PROCESS CONTROL

- ***FORTH*** programs are instantly portable across the four most popular microprocessors.
- ***FORTH*** is interactive and conversational, but 20 times faster than BASIC.
- ***FORTH*** programs are highly structured, modular, easy to maintain.
- ***FORTH*** affords direct control over all interrupts, memory locations, and i/o ports.
- ***FORTH*** allows full access to DOS files and functions.
- ***FORTH*** application programs can be compiled into turnkey COM files and distributed with no license fee.
- ***FORTH*** Cross Compilers are available for ROM'ed or disk based applications on most microprocessors.

Trademarks: IBM, International Business Machines Corp.; CP/M, Digital Research Inc.; PC/Forth+ and PC/GEN, Laboratory Microsystems, Inc.

FORTH Application Development Systems include interpreter/compiler with virtual memory management and multi-tasking, assembler, full screen editor, decompiler, utilities, and 130 + page manual. Standard random access files used for screen storage, extensions provided for access to all operating system functions.

Z-80 FORTH for CP/M® 2.2 or MP/M II, $50.00; **8080 FORTH** for CP/M 2.2 or MP/M II, $50.00; **8086 FORTH** for CP/M-86 or MS-DOS, $100.00; **PC/FORTH** for PC-DOS, CP/M-86, or CCPM, $100.00; **68000 FORTH** for CP/M-68K, $250.00.

FORTH + Systems are 32 bit implementations that allow creation of programs as large as 1 megabyte. The entire memory address space of the 68000 or 8086/88 is supported directly.

PC FORTH +	$250.00
8086 FORTH + for CP/M-86 or MS-DOS	$250.00
68000 FORTH + for CP/M-68K	$400.00

Extension Packages available include: software floating point, cross compilers, INTEL 8087 support, AMD 9511 support, advanced color graphics, custom character sets, symbolic debugger, telecommunications, cross reference utility, B-tree file manager. *Write for brochure.*

VISA

Laboratory Microsystems Incorporated
4147 Beethoven Street, Los Angeles, CA 90066
Phone credit card orders to (213) 306-7412

MasterCard

Macro Expansion in Forth

Jeffrey Soreff
Santa Clara, California

By the proper use of **COMPILE**, **[COMPILE]** and **IMMEDIATE**, one can write macros in Forth that insert immediate and normal words into colon definitions. A string of non-immediate Forth words is easily put in a macro, since this is just the ordinary behavior of Forth function definitions. Suppose, however, that we wish to put immediate words, such as **DO** or **IF**, into a macro. This case occurred in an attempt to code altered versions of **DO** and **LOOP**. A version of **DO** and **LOOP** was desired which would not execute the loop body even once if the lower index was greater than or equal to the upper index. This would give cleaner behavior for some situations, since the number of times that the loop would execute would be the value of the upper index minus the value of the lower index for all cases where this is possible (non-negative integers), instead of having an exception for zero. This can be done if one could expand the code in figure one into that shown in figure two.

The initial **IF** test jumps around the loop if the index difference is less than or equal to zero. The problem with simply coding this as in figure three is that the compiler will complain about the unterminated **DO** and **IF**. The essential problem is that **IF**, **DO**, **LOOP** and **ENDIF** are all immediate words, with effects at compile time. We need to **DEFER** their execution from the compile time of **DO'** and **LOOP'** to the compile time of the word that **DO'** and **LOOP'** are to be used within. This can be done by using **[COMPILE]** to delay execution. If, however, we simply wrote **DO'** as in figure four and used it as in figure five, then the *compile time* actions of **IF** and **DO** would occur at the *run time* of **TST**, rather than during its compilation, as we want. This can be avoided by writing **DO'** as an immediate function, shown in figure six.

```
:  NEW-FUNCTION
   A  DO'  B  LOOP'  C  ;
```

Figure One

```
:  NEW-FUNCTION
   A  2DUP  -  0>  IF  DO  B  LOOP  ELSE  2DROP  ENDIF  C  ;
```

Figure Two

```
:  DO'  2DUP  -  0>  IF  DO  ;
:  LOOP'  LOOP  ELSE  2DROP  ENDIF  ;
```

Figure Three

```
:  DO'  2DUP  -  0>
   [COMPILE]  IF  [COMPILE]  DO  ;
```

Figure Four

```
:  TST  DO'  I  .  LOOP'  ;
```

Figure Five

```
:  DO'  2DUP  -  0>
   [COMPILE]  IF  [COMPILE]  DO  ;  IMMEDIATE
```

Figure Six

```
:  DO'  COMPILE  2DUP  COMPILE  -  COMPILE  0>
   [COMPILE]  IF  [COMPILE]  DO  ;  IMMEDIATE
```

Figure Seven

```
:  A-MACRO  COMPILE  NON-IMMEDIATE-WORD
   [COMPILE]  IMMEDIATE-WORD  ;  IMMEDIATE
```

Figure Eight

```
:  USE-A-MACRO  WORD1  WORD2  A-MACRO  WORD3  WORD4  ;
```

Figure Nine

The **[COMPILE]** words defer the actions of the immediate words **IF** and **DO** until the run time of **DO'**, and the **IMMEDIATE** makes that run time occur at the compile time of **TST**, as desired. A new problem is introduced by this, because the *non*-immediate words **DO'**, **2DUP**, **-** and **0>** will also act at the compile time of **TST**. This can be solved by changing the action invoked by the non-immediate words from one of execution to one of inserting code pointers. This is done by inserting **COMPILE** before all non-immediate words, resulting in the code in figure seven.

In general, one can write a macro containing both immediate and non-immediate words by using the code in figure eight in the manner shown in figure nine.

To recapitulate, the immediate words are all preceded by **[COMPILE]**, which delays their execution from the compile time of **A-MACRO** to the run time of **A-MACRO**. Since **A-MACRO** is an immediate word, its run time is the compile time of **USE-A-MACRO**, which is exactly when the immediate words would have executed if they had been typed directly into the definition of **USE-A-MACRO**. The non-immediate words are all preceded by **COMPILE**, which makes **A-MACRO** insert the address of **IMMEDIATE-WORD** into the code for **USE-A-MACRO** when **A-MACRO** is run, *i.e.* at **USE-A-MACRO**'s compile time. This is the same action that the colon compiler would have taken if the non-immediate words had been typed into **USE-A-MACRO**'s definition directly.

Editor's note: We like Jeffrey's code very much. Readers who use his ideas are encouraged to send us examples of your macros, especially of their less obvious uses.

FORTH-79

Ver. 2 For your APPLE II/II+

The complete professional software system, that meets ALL provisions of the FORTH-79 Standard (adopted Oct. 1980). Compare the many advanced features of FORTH-79 with the FORTH you are now using, or plan to buy!

FEATURES	OURS	OTHERS
79-Standard system gives source portability.	YES	
Professionally written tutorial & user manual	200 PG.	
Screen editor with user-definable controls.	YES	
Macro-assembler with local labels.	YES	
Virtual memory.	YES	
Both 13 & 16-sector format.	YES	
Multiple disk drives.	YES	
Double-number Standard & String extensions.	YES	
Upper/lower case keyboard input.	YES	
LO-Res graphics.	YES	
80 column display capability	YES	
Z-80 CP/M Ver. 2.x & Northstar also available	YES	
Affordable!	$99.95	
Low cost enhancement option:		
Hi-Res turtle-graphics.	YES	
Floating-point mathematics.	YES	
Powerful package with own manual, 50 functions in all, AM9511 compatible.		

FORTH–79 V.2 (requires 48K & 1 disk drive)	$ 99.95
ENHANCEMENT PACKAGE FOR V.2	
Floating point & Hi-Res turtle-graphics	$ 49.95
COMBINATION PACKAGE	$139.95

(CA res. add 6% tax: COD accepted)

MicroMotion
12077 Wilshire Blvd. # 506
L.A., CA 90025 (213) 821-4340
Specify APPLE, CP/M or Northstar
Dealer inquiries invited.

FORTH-79

Version 2 For Z-80, CP/M (1.4 & 2.x), & NorthStar DOS Users

The complete professional software system, that meets ALL provisions of the FORTH-79 Standard (adopted Oct. 1980). Compare the many advanced features of FORTH-79 with the FORTH you are now using, or plan to buy!

FEATURES	OURS	OTHERS
79-Standard system gives source portability.	YES	
Professionally written tutorial & user manual.	200 PG.	
Screen editor with user-definable controls.	YES	
Macro-assembler with local labels.	YES	
Virtual memory.	YES	
BDOS, BIOS & console control functions (CP/M).	YES	
FORTH screen files use standard resident file format.	YES	
Double-number Standard & String extensions.	YES	
Upper/lower case keyboard input.	YES	
APPLE II/II+ version also available.	YES	
Affordable!	$99.95	
Low cost enhancement options;		
Floating-point mathematics	YES	
Tutorial reference manual		
50 functions (AM9511 compatible format)		
Hi-Res turtle-graphics (NoStar Adv. only)	YES	

FORTH-79 V.2 (requires CP/M Ver. 2.x).	$99.95
ENHANCEMENT PACKAGE FOR V.2:	
Floating point	$ 49.95
COMBINATION PACKAGE (Base & Floating point)	$139.95

(advantage users add $49.95 for Hi-Res)
(CA. res. add 6% tax; COD & dealer inquiries welcome)

MicroMotion
12077 Wilshire Blvd. # 506
L.A., CA 90025 (213) 821-4340
Specify APPLE, CP/M or Northstar
Dealer inquiries invited.

Fixed-Point Logarithms

Nathaniel Grossman
Los Angeles, California

This paper shows how to compute logarithms using (fixed point) Forth. The algorithm is implemented for nine-decimal-place logarithms, but the algorithm and the implementation easily can be modified to yield logarithms to any number of decimal places and to any logarithmic base.

Logarithms were invented by John Napier, Baron Merchiston, and Napier's logarithmic tables were first published in 1614. Napier conceived of logarithms as a tool for simplifying the laborious computations in astronomy. (Earlier, he had invented "Napier's bones," cleverly marked rods for carrying out multiplication without memorizing multiplication tables.) Napier's tables, because of their obvious utility, were eagerly adopted, one consequence being general acceptance of the decimal point, a notation first used in modern form in the tables.

(Even long after Napier's time, trigonometric tables contained only function values scaled to integers. Sines and cosines were considered as lengths of sides in a right triangle whose hypotenuse was the scaling factor, *e.g.* 10,000,000 for "seven-place" tables. Our modern conception of trigonometric functions as ratios of sides is due to Euler, who introduced such notions in 1748.)

A logarithm is a function that assigns to every positive number x a second number log x in such a way that

$$\log(xy) = \log x + \log y$$

and

$$\log(x/y) = \log x - \log y$$

Thus, logarithms convert multiplication into addition and division into subtraction. They also convert the taking of powers into multiplication and the taking of roots into division, and further application of logarithms converts these into addition and subtraction, respectively. In the days before mechanical and electronic calculators, logarithms afforded the only way to bypass the tedium of calculation and, in some cases, the only way to perform certain calculations.

With the invention of the geared mechanical calculator (Robert Hooke's diary contains in its entry for January 22, 1672, the remark "Libnitius [Leibniz] shewd Arithmetical engine [to the Royal Society]") and, later, the electronic computer, the computational significance of logarithms withered away. Nevertheless, logarithmic expressions are an indispensable ingredient of many scientific formulas. Even those who make little direct use of such formulas may find occasion to plot data in semi-log or log-log displays.

As a novice who was led to Forth by reading how it was harnessed to control a gigantic model railroad, I was initially stunned and demoralized when I found out that Forth implementations need not contain floating point and higher mathematical function routines. How, I wondered, could one carry out mathematical computations in Forth?

Luckily, tonics were at hand. *Forth Dimensions* offered examples of mathematical functions constructed within Forth. Attending my first meeting of the Los Angeles FIG Chapter, I was gently and patiently reminded that Forth is *extensible*, so that it can be tailored to each user's needs. I knew that I could not abandon this elegant language, and I decided to push on, using Forth as it was created to be used.

Because square roots and trigonometric functions had already been dealt with in *Forth Dimensions* (IV/1), I settled on logarithms as my goal. In the Spring of 1983, I had taught a course on the Historical Development of the Calculus in which I devoted several hours of lectures to methods used for construction of the earliest logarithm tables by Napier, Briggs and Burgi. I remembered that those tables were constructed by very simple arithmetic steps repeated many times, and that the computations were, in effect, in fixed-point arithmetic. After considering the various ways in which logarithms and logarithmic tables could be and have been constructed, I decided to return to the fountainhead.

Construction of Algorithms

There are many ways to compute logarithms. As I have settled on one of the oldest methods, I will begin by discussing some of the "newer" methods and explaining why I discarded them.

1. The more modern tables were computed using infinite series, specifically Taylor series that can be constructed by means of the Calculus. In general, not every entry was obtained directly from a series of computations. Instead, "enough" values were calculated directly by series and the gaps were filled in by interpolation or differencing schemes, much as "ordinary" surveyors densify a network starting from benchmarks left by the Geodetic Survey.

The key series is

$$\ln(1 + x) = x - x^2/2 + x^3/3 - x^4/4 + \dots$$

valid if $|x| < 1$. It is necessary to truncate the series into a polynomial at a stopping place suitable for the precision sought. The resulting polynomial is then inefficient because there will be polynomials of lower degree that will give the desired precision throughout the specified x-domain. For example, there is a polynomial of degree eight that gives $\ln(1 + x)$ for $0 \leq x \leq 1$ with, at most, a $3*10^{-8}$ error[1]. (Other polynomials can be found in [2].) The advantage of the special approximating polynomials is that they approximate the function to uniform precision over the whole range of validity. The disadvantage is that nastier coefficients must be stored, so they do not have the convenient recursive coefficient calculation such as Bumgarner exploited in his computation of sines (*Forth Dimensions* IV/1). Furthermore, what if the requirements are changed to require logarithms with higher precision? The chosen polynomial now falls short and there may not be another, more suitable polynomial lying near at hand. This method must be discarded because it is not extensible to fit needs that may arise (although it might be fine, even best, for a specific application).

2. The possibility of simply storing a table of logarithms must be considered. This method might be suitable for a specific application. In deciding how many entries to store, a balance must be struck among the precision required, the types of interpolation, if any, acceptable, the density of arguments and the lookup time. Because Forth is intended for smaller computers with limited storage, this method in general appears counter to Forth philosophy. Besides, the typing of long tables is prone to errors and simply dull.

3. There is a wonderfully swift, second-order, iterative algorithm for computing logarithms. Start with

$$y_1 = (x^2 - x^{-2})/4$$

and

$$y_2 = (x - x^{-1})/2$$

then compute $y_2, y_3, \ldots$ by the formula

$$y_{k+1} = y_k \sqrt{\frac{2y_k}{y_k + y_{k-1}}}$$

The limiting value of y_k as k grows larger is ln x, the natural logarithm of x. This formula would seem most suitable for systems with square root and floating point. Requiring the extra load just to calculate logarithms is counter to Forth philosophy.

4. The CORDIC algorithm is implemented in many pocket calculators and in other computers for obtaining mathematical functions. A description of an algorithm for part of the CORDIC output has been printed in *Forth Dimensions* (IV/1, V/3). Again, this method entails extra steps not required if one seeks only to calculate logarithms.

5. Other methods are available, some of which are described in [2]. In particular, there is a very simple algorithm[2] that produces the binary digits of $x = \log_2 y$ for any y with $1 \leq y \leq 2$. Those digits appear one-by-one as flags after an absurdly simple calculation involving only squaring followed by a comparison. Unfortunately, the squaring operation produces a sequence of numbers whose values again lie between 1 and 2, but whose digits after the point grow exponentially with the number of squarings. The algorithm is hardly suitable for implementing on small computers where Forth is most at home.

Selection of the Logarithm

Before beginning to describe the method implemented, that of factor-logarithms, I must settle one important point: *which* logarithm to compute? The fundamental properties of logarithms pin down the logarithm function only up to a scale factor. That factor can be fixed by naming the number (the *logarithmic base*) which has logarithm equal to 1. Thus, a table of logarithms to one logarithmic base can be converted to a table of logarithms to a new base by multiplying the originals by a suitable constant.

Certain logarithmic bases lead to logarithmic functions with special properties that make them more useful for one or another purpose. The *natural* logarithm, whose functional designation frequently is "ln," is called natural because its use simplifies certain formulas in the Calculus. The *denary* (or *common* or *base 10*) logarithm has a characteristic called the "characteristic" that makes it very convenient for ordinary arithmetic calculations. (The natural logarithm is not convenient for arithmetic.) I use neither of these, choosing instead the *binary* (or *base 2*) logarithms. Binary logarithms occur naturally in formulas arising in Information Theory. The binary logarithm also has a "characteristic" that makes it convenient for binary arithmetic such as many computers use internally. The binary logarithmic function is denoted by $\log_2$, and it is fixed by requiring that $\log_2 2 = 1$.

Note that the logarithm chosen for use in semi-log or log-log plotting is not crucial, because it is to be expected that the plot will be scaled to paper size. Similarly, the use of natural logarithms in scientific formulas, while making sense during derivation of the formula, is of no advantage and might even be a disadvantage when the emphasis switches from "derivation-time" to "computation-time." Many simple scientific formulas involve half-lives, doubling times, fifty-percent extinction times, distance travelled for attenuation to half-strength and the like. For such calculations, base 2 logarithms and exponentials are most convenient.

Having, I hope, justified computing binary logarithms, I will close this section by describing the factors for converting from one logarithmic base to another. The notation $\log_a$ is standard for indicating logarithms to the base a. If b is a second base, then the logarithms of x to the bases a and b are related by the formulas

$$\log_b x = \log_a x \log_b a = \log_a x / \log_a b$$

The natural logarithm ln is the same as $\log_e$, where e is a certain number about 2.7. Thus

$$\ln x = \ln 2 * \log_2 x$$

Rather than multiplying by the value of ln 2 to the number of places to which $\log_2$ is kept, which would require multiple-precision arithmetic, ln 2 can be written as the quotient of two single factors. One pair that falls just short for nine-place logarithms can be found in *Forth Dimensions* (IV/4):

$$\ln 2/16.384 = 846/19997$$

with relative error $-12 * 10^{-9}$. Multiplication by ln 2 can be effected as

```
846 1000 U*/ 16384 19997 U*/
```

where the word **U*/** is defined in the first of the accompanying screens.

The Algorithm

If the number x is restricted by $0 \leq x < 1$, the natural logarithm ln then satisfies

$$0 \leq \ln(1 + x) < x$$

The binary logarithm then satisfies

$$0 \leq \log_2(1 + x) < x/\ln 2$$

Hence, forcing x toward 0 also forces $\log_2(1 + x)$ toward 0.

The goal is $\log_2 N$, where N is a positive number (in the present case, a positive integer). Write $N = 2^c N_1$, where c is a non-negative integer (called the binary characteristic of N) and $1 \leq N_1 < 2$. The fundamental property of $\log_2$ yields the equation

$$\log_2 N = c + \log_2 N_1$$

The number $\log_2 N_1$, for which $0 \leq \log_2 N_1 < 1$, is called the *mantissa* of N_1. (Strictly speaking, it is the binary mantissa.)

Write $N_1 = 1 + M_1$, where $0 \leq M_1 < 1$. If M_1 is close enough to 0, then $\log_2 N_1$ is also small, so $\log_2 N$ equals the integer c plus a small non-negative correction. Unfortunately, M_1 cannot be counted upon to be close to 0: take $N = 255$, whence $c = 7$ and M_1 is close to 1. Therefore, it is necessary to take steps to *force* M_1 — or rather a proxy for M_1 — toward 0.

By a sequence of binary right shifts followed by subtractions, the non-zero binary digits of N_1 to the right of the point can be removed one by one. Figure one provides an example. The horizontal lines indicate subtractions, and the lines enclosed in parentheses represent subtractions that were cancelled because they left remainders less than 1.

```
                     N1 =  1.1101
(1 - 2^-1)                  (11101)
                           ---------
 1 - 2^-2                     11101
                           ----------
                           1.010111
 1 - 2^-2                     1010111
                           -----------
                           1.00000101
(1 - 2^-3)                    (100000101)
                           ---------------
(1 - 2^-4)                      (100000101)
                           ----------------
(1 - 2^-5)                        (100000101)
                           ------------------
 1 - 2^-6                             100000101
                           ------------------
                           1.00000000111011
```

Thus, in a mix of binary and decimal,

$$(1.1101)(1 - 2^{-2})^2(1 - 2^{-6}) = 1.00000000111011$$

By shifting six steps to the right, the first six places after the point have been swept clear to zeros. Observe that more than one shift in a given position may be required. In general, there will be a product decomposition

$$N_1(1 - 2^{-1})^{a_1}(1 - 2^{-2})^{a_2} \ldots (1 - 2^{-k})^{a_k} = 1 + M_{k+1}$$

where

$$0 \leq M_{k+1} < 2^{-k}$$

and each a-number is a non-negative integer.

Suppose the goal is to compute $\log_2 N$ to n *binary* places. Recall that

$$0 \geq \log_2 1 + M_{k+1} < < M_{k+1}/\ln 2$$

Because

$$1/\ln 2 < 1.5 < 2$$

then

$$(\log_2(1 + M_{k+1}) < 2^{-k+1})$$

To be sure that the value of

$$\log_2(1 + M_{k+1})$$

can have no effect on the first n digits after the point, it will suffice to choose k so large that

$$2^{-k+1} < 2^{-n}$$

that is, $k > n + 1$

That done, we write

$$l_i = \log_2[1/(1 - 2^{-i})]$$

for $i = 1,2,\ldots,k$. Then the fundamental property of logarithms yields

$$\log_2 N_1 = a_1 l_1 + a_2 l_2 + \ldots + a_k l_k$$

valid to n binary places. Because each a-coefficient is an integer, only addition will be required to generate $\log_2 N_1$ by this formula.

The values $l_1,\ldots,l_k$ must be generated and stored in memory in advance. These are the (binary) *factor-logarithms* that give the method its name.

Multiuser/Multitasking
for 8080, Z80, 8086

TaskFORTH™

The First
Professional Quality
Full Feature FORTH
System at a micro price*

LOADS OF TIME SAVING PROFESSIONAL FEATURES:

- ☆ Unlimited number of tasks
- ☆ Multiple thread dictionary, superfast compilation
- ☆ Novice Programmer Protection Package™
- ☆ Diagnostic tools, quick and simple debugging
- ☆ Starting FORTH, FORTH-79, FORTH-83 compatible
- ☆ Screen and serial editor, easy program generation
- ☆ Hierarchical file system with data base management

* Starter package $250. Full package $395. Single user and commercial licenses available.

If you are an experienced FORTH programmer, this is the one you have been waiting for! If you are a beginning FORTH programmer, this will get you started right, and quickly too!

Available on 8 inch disk under CP/M 2.2 or greater also various 5¼" formats and other operating systems

FULLY WARRANTIED, DOCUMENTED AND SUPPORTED

master charge — DEALER INQUIRES INVITED — VISA

Shaw Laboratories, Ltd.
24301 Southland Drive, #216
Hayward, California 94545
(415) 276-5953

Each l-value carries as many binary places as the mantissa must have. The number of places (n) then determines the number (k) of l-values to be stored.

Note that the shift subtractions cause non-zero digits to propagate toward the right. If the arithmetic registers are of fixed width, full accuracy is maintained until the digits run out the low end of the registers, after which false borrowings may propogate erroneous digits to the left.

I have implemented the factor-algorithm in Forth having "normal" specifications. To get nine decimal places — with an error of a few units in the ninth place because of roundoff in the stored factor-logarithms and truncations in the shifts — it suffices to carry thirty binary digits after the point and to store the first fifteen binary factor-logarithms.

Description of the Screens

I have implemented the algorithm on a VIC-20 computer using the HES VIC-FORTH dialect, a subset of FIG-Forth. (VIC-FORTH contains the 6502 **U/** bug described in *Forth Dimensions* V/1, but the divisions called in the screens are in the safe region.) The early screens supply some of the standard arithmetic words and double-number extensions not in VIC-FORTH. Here are details of the screens.

Screen #1: Binary shifts are natural in a binary computer and they should be coded at low level for speed. I carry them at high level, which leads to considerable slowing of execution. In order not to lose digits, it is necessary to use triple-numbers. **U*/** is the arithmetic operation required and **DRTSHIFT** multiplies a ud by 1, yielding an intermediate ut, then divides by 2 to give the right-shifted ud.

Screen #2: The factor-logarithms are double-numbers and they are stored in a *double table*, a new data type created by the defining word **DTABLE**.

Screen #3: Here is the actual double table. The initial entry (1.000000000) is *never* called: if $1 \leq x < 2$, then

$$(1 - 2^{-1})*x < 1$$

I put it in so as not to upset **DTABLE**, which is generic.

Screen #4: The word **CHARACTERISTIC** shifts a double-number argument to the right, keeping count of

```
( SCR #1: 9-PLACE BINARY LOGARITHMS OF 9-DIGIT INTEGERS   )
( NATHANIEL GROSSMAN, 9/19/83 --- HES VIC-FORTH            )
: T*    ( UD,UN --- UT )
  DUP ROT U* >R >R              ( MULT UPPER PRECISION PART )
      U*                        ( MULT LOWER PRECISION PART )
  0 R> R>  D+ ;                 ( ADD BOTH PARTS            )
: T/    ( UT,UN --- UD )
    >R  R  U/ SWAP              (  DIV UPPER PRECISION PART )
  ROT 0 R U/ SWAP               (  DIV LOWER PRECISION PART )
  ROT  R> U/ SWAP DROP          (  DIV REMAINDER            )
  0 2SWAP SWAP D+ ;             (  ADD PARTS                )
: U*/  ( UD,UN,UN --- UD )
  >R T* R> T/ ;
: DRTSHIFT  ( UD --- UD; DBL RIGHT SHIFT )
  1 2 U*/ ;
;S
```

```
( SCR #2: LOG2 --- BINARY LOGARITHMS  NG,9/19/83 )
( UD -> DBLMANTISSA / CHARACTERISTIC        )
( 0 < UD < 2**30 = 1,073,741,824            )
: D,          , ,               ( STORE DBL ) ;
: D@      DUP @ SWAP 2 + @ SWAP ( FETCH DBL ) ;
: D0=     OR 0 =                ( DBL = 0 ? ) ;
: DTABLE
   <BUILDS
   DOES>
    SWAP 1 - 4 * + D@ ;

   -1 VARIABLE CHAR
    0 VARIABLE SHIFT-COUNT
    1 VARIABLE SHIFT-DIVISOR
16384 CONSTANT ?TOO-BIG  ;S
```

```
( SCR #3: LOG2 CONT NG,9/19/83 )
DTABLE FACTOR-LOG2  ( -LOG2[1-2**-M], M = 1 TO 15 )
1.000000000 D, .415037499 D, .192645078 D,
 .093109404 D, .045803690 D, .022720077 D,
 .011315313 D, .005646563 D, .002820519 D,
 .001409570 D, .000704613 D, .000352263 D,
 .000176121 D, .000088058 D, .000044028 D,

: D<   ( FROM "ALL ABOUT FORTH" )
  ROT 2DUP =
  IF      ROT ROT DMINUS D+ 0 <
  ELSE    SWAP < SWAP DROP
  ENDIF   SWAP DROP ; ;S



```

```
( SCR #4: LOG2 CONT  NG, 9/19/83 )
: CHARACTERISTIC    ( UD --- UD; STORE CHAR VALUE 0 TO 30   )
   2DUP             ( SAVE COPY OF UD                        )
   -1 CHAR !        ( INITIALIZE VARIABLE CHAR               )
   BEGIN            ( SEEK LARGEST POWER OF 2 AT MOST UD     )
    1 CHAR +!       ( INCREASE FOR COUNTING                  )
    DRTSHIFT        ( DBL RIGHT SHIFT                        )
    2DUP D0=        ( NEXT POWER TOO LARGE?                  )
   UNTIL            ( YES? THEN BAIL OUT OF LOOP             )
  2DROP ;           ( CLEAN STACK                            )
: LEFT-ALIGN        ( UD --- UD; SHIFTS LEFT FOR MANTISSA    )
   30 CHAR @ -      ( HOW MANY LEFT SHIFTS?                  )
    0 DO
      2DUP D+       ( SHIFTS TO LEFT IN DBL                  )
     LOOP ; ;S
```

```
( SCR #5: LOG2 CONT  NG,9/19/83 )
: UPDATE-INDICES       (   ---  )
   1 SHIFT-COUNT +!
   SHIFT-DIVISOR @ DUP +    ( DOUBLE THE SHIFT DIVISOR )
   SHIFT-DIVISOR ! ;
: SHIFT-STEP            ( UD --- UD, FACTOR GENERATOR )
   2DUP                 ( UD,UD                       )
   SHIFT-DIVISOR @
   1 SWAP               ( UD,UD,N,1                   )
   U*/
   DMINUS D+ ; ;S       ( FRESH COFACTOR              )



```

```
( SCR #6: LOG2 CONT  NG,9/19/83 )
: CHECK-SHIFT   ( UD --- UD,F )
  2DUP 0 ?TOO-BIG D< ;   ( TOO MUCH SUBTRACTED?          )
: NEW-COFACTOR  ( UD,F --- UD,F )
  IF          2DROP 0    ( TOO MUCH                      )
  ELSE  2SWAP 2DROP 1    ( NOT TOO MUCH; DROP INSURANCE )
  ENDIF ;
: FACTOR2       ( UD --- UD,F; PRODUCE ONE FACTOR OR SKIP )
  2DUP
  SHIFT-STEP             ( SHIFT AND SUBTRACT            )
  CHECK-SHIFT            ( TOO MUCH SUBTRACTED?          )
  NEW-COFACTOR ;         ( NEXT NUMBER TO FACTOR         )
;S
```

```
( SCR #7: LOG2 CONT  NG,9/19/83 )
: INIT'IZE-MANT2  ( UD1,UD2 ---  UD2,UD1,UD1 )
  0 SHIFT-COUNT !        ( INITIALIZE                )
  1 SHIFT-DIVISOR !      ( INITIALIZE                )
  0 S->D                 ( MAKE SLOT TO HOLD MANTISSA )
  2SWAP ;                ( MOVE SLOT DEEPER INTO STACK)
: MANT2-SETUP    ( UD2,UD1,UD1 --- UD2,UD3  )
  CHARACTERISTIC
  LEFT-ALIGN ;
: DBL/LN(2)          ( UD --- UD )
  13436 10000 U*/ ;
: LEFT-OVERS         ( UD --- UD )
  0 ?TOO-BIG DMINUS D+ ( ISOLATE LEFT-OVER SUMMAND  )
  DBL/LN(2) D+ ;       ( SCALE AND ADD              )
;S
```

```
( SCR #8: LOG2 CONT  NG,9/19/83 )
: MANTISSA2     ( UD --- UD )
  INIT'IZE-MANT2  MANT2-SETUP  ( PREPARE FOR MANTISSA CALC )
  15 0                         ( ENTER SHIFT-SUBTRACT LOOP )
   DO
    UPDATE-INDICES
     BEGIN   FACTOR2
     WHILE   SHIFT-COUNT @      ( IF THERE IS AN ADDEND     )
             FACTOR-LOG2        ( BRING IT UP               )
             2SWAP >R >R        ( AND                       )
             D+     R> R>       ( ADD IT IN                 )
     REPEAT
    LOOP
    LEFT-OVERS ; ;S             ( LAST SCRAP, AND MANTISSA2 )
```

```
(SCR #9: LOG2 CONCLUDED  NG,9/19/83 )
: LOG2   ( UD --- UD,N; 0 < ARGUMENT < 2**30 = 1,073,741,824)
  2DUP 2DUP
  0 S->D 2SWAP D<        ( NOT TOO SMALL?              )
  ROT ROT
  0 ?TOO-BIG D<          ( NOT TOO LARGE?              )
  AND
   IF                    ( JUST RIGHT! STACK           )
    MANTISSA2            ( WILL HOLD SGL CHARACTERISTIC )
    CHAR @               ( BELOW DBL MANTISSA          )
   ELSE
    CR CR ." NOT IN LOG2'S DOMAIN " CR CR
   ENDIF ; ;S
```

the shifts, until the result becomes 0. Then **LEFT-ALIGN** shifts the original argument to the left until its leading digit (= 1) occupies the bit 31 in the double register.

Screen #5: **UPDATE-INDICES** keeps track of the binary place (i) which is currently being acted upon and generates the divisor (2^i) that effects the shift. **SHIFT-STEP** carries out the shift-subtraction.

Screen #6: **CHECK-SHIFT** looks to see if too much has been subtracted. **NEW-COFACTOR** either validates the new cofactor — if the subtraction was not too much — or backs up and restores the old cofactor. All the factoring steps are gathered together in **FACTOR2**.

Screen #7: The first two words prepare all the registers and indices for computation of the mantissa. If it were possible to divide by 2^{16}, then the double table **FACTOR-LOG2** could contain sixteen numbers and

$\log_2(1 + M_{17})$

would always contain only zeros in the first decimal places. But that division is not possible, at least not at a reasonable cost in time.

There is a simple way to deal with the left-overs after fifteen shift-subtractions. A more elaborate expansion than that used before gives

$$\ln(1 + M_{16}) = M_{16} - \tfrac{1}{2}M_{16}^2 + \ldots$$

and

$\tfrac{1}{2}M_{16}^2$

is always zero to nine decimal places. But it is not always valid just to replace $\ln(1 + M_{16})$ by M_{16}. The difficulty comes from the representation of all numbers by integers. Thus, 1.xxx...x is represented as

$$2^{30}[1 + (J/2^{30})]$$

Hence,

$\log_2(1 + M_{16})$

— approximated numerically by $M_{16}/\ln 2$ — is actually represented by the integral part of

$$(J/2^{30})/\ln 2$$

Taking scaling into account, this is addition of $(1.3436)(J)(10^{-9})$ in decimal. The word **LEFT-OVERS** carries out the final adjustment, using the scaling word **DBL/LN(2)**.

Screen #8: The word **MANTISSA2** accepts a *double*-number considered as a positive integer less than 2^{30} = 1,073,741,824 (including, therefore, all integers of at most nine digits) and returns a double number considered as a point followed by nine decimal digits. (Leading zeros will be suppressed if the mantissa is **D.**)

Screen #9: Here is the goal word, "the last for which the first was made." Enter a positive double number less than 2^{30} followed by **LOG2**. Then the first-out number on the stack will be the characteristic, and this will be followed by the double-number mantissa. (Separation of characteristic and mantissa is completely analogous to the same separation for common logarithms and has no analogue for natural logarithms.) If the argument is negative or too large, an appropriate error message appears.

Example: Type **2050. LOG2 . D.** and then execute. The first number printed will be the characteristic 11 and the second will be the binary mantissa 1408199. Thus, $\log_2 2050$ = 11.001408199.

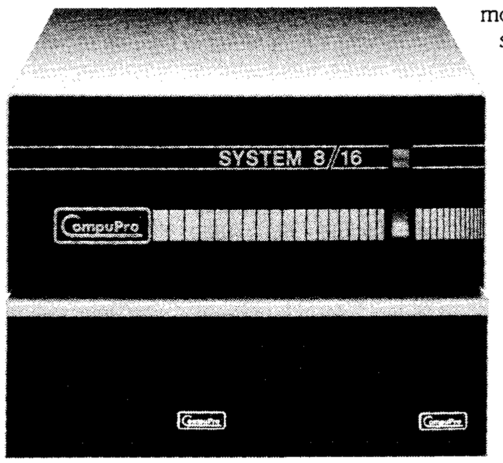
It is easy to modify the screens for

(Continued on page 17)

PIECE OF MIND

CompuPro's System 816.
The fastest, most cooperative computer you can buy.

OEMs and systems integrators are busy people. Too busy to waste time with an uncooperative computer system. That's why every System 816 from **CompuPro** is built to work long and hard without a whine or a whimper.



More Dependable.

With ten years of pioneering successes built into it, the System 816 is backed by the industry's longest warranty coverage. Depending on your needs, our warranties range from 12 to 36 months. Most other computer manufacturers expect you to be satisfied with 90 days, which typically covers parts only.

You can also depend on complete hardware and software support, flexible configurations and upgrades, and system training.

More Powerful.

The System 816 squeezes more performance out of the IEEE-696/S-100 bus than any other system you can buy. A choice of CPUs—and up to 4 Mbytes of our exclusive M-Drive/H™ RAM disk—give multiple workstations all the speed and power they can ask for. Standard RAM memory is expandable to one megabyte or more.

Disk storage capacity ranges up to 4.8 Mbytes on floppy drives and as much as 320 Mbytes per controller on hard disk.

More Versatile.

All family members share a common modular architecture. So it's a simple matter to upgrade or reconfigure any of them to keep up with your needs. All the while maintaining complete software compatibility up and down the line.

And the S-100 bus allows you the flexibility to plug in any compatible board to add graphics capabilities or boards for your own unique applications.

You also get your choice of operating environments, including CP/M®, CP/M-86®, Concurrent CP/M-86™, MP/M-86™ and CP/M-68K™, and our own CP/M®8-16™ and MP/M™8-16™.

At the programming level, the System 816 family supports Pascal, C, FORTH, BASIC, COBOL, PL/1, FORTRAN 77™ and more.

More Information.

Your customer's satisfaction is important to both of us, so don't get stuck with a system that's more of a hindrance than a help. Send in the coupon and find out what peace of mind is all about.

For business, scientific and industrial computing solutions, call **(415) 786-0909** ext. 506 for the location of our dealers worldwide, or the **Full Service CompuPro System Center** nearest you.

☐ Send me your free System 816 brochure.

☐ Send me the name of my nearest Full Service CompuPro System Center or dealer.

NAME ______________________

TITLE ______________________

ADDRESS ______________________

CITY ______________ STATE ______ ZIP ______

Mail to: CompuPro, Attn: Sales Dept.
3506 Breakwater Court, Hayward, CA 94545

CompuPro®
A *GODBOUT* COMPANY
3506 Breakwater Court, Hayward, CA 94545

CP/M and CP/M-86 are registered trademarks and CP/M-68K, MP/M-86, Concurrent CP/M-86 and FORTRAN 77 are trademarks of Digital Research Inc. CP/M 8-16 and MP/M 8-16 are compound trademarks of Digital Research Inc. and **CompuPro**.

Prices and specifications subject to change without notice.
System 816 front panel design shown is available from **Full Service CompuPro System Centers** only.
1983 COMPUPRO

(Continued from page 15)

varying needs. Cutting the double table **FACTOR-LOG2** to a table holding seven five-place entries and then replacing the double operations by single operations, five-place logarithms can be obtained, and there is room to round off to four places. By inserting quadruple operations (with some quintuple intermediates) and loading a **QTABLE** containing thirty quadruple logarithmic values, it will be possible to obtain logarithms to eighteen places.

It is more likely that single-number logarithms will be needed than quadruple ones. Nevertheless, the algorithm is extensible, without ado, to any needed precision. Much more than half the work needed to obtain the factor-logarithms to a large number of places has been carried out by H.S. Uhler, who published in 1942 a now-scarce book[4] presenting — among other tables — natural factor-logarithms to 137 decimal places! The factorization scheme for decimally presented numbers is far more elaborate than the scheme we gave above for binary-represented numbers. Uhler's book has an introductory essay which, besides explaining why he needed 137 decimal places, describes the mind-boggling precautions he took to avoid copying and entering errors more common in the days of mechanical calculators.

References

1. Abramowitz, M., and I.A. Stegun, *Handbook of Mathematical Functions*, National Bureau of Standards Applied Mathematics Series, 55. (Reprinted by Dover Publications.) Table 4.3 presents twenty-five-place factor-logarithms without instructions for their use.

2. Lyusternik, L.A., *et al.*, *Handbook for Computing Elementary Functions*, Pergamon Press, 1965.

3. Salzer, H.E., Radix tables for finding logarithms to twenty-five decimal places, in *Tables of Functions and of Zeros of Functions*, National Bureau of Standards Applied Mathematics Series, 37. A bare-bones table containing fourteen twenty-eight-place natural-factor logarithms, with instructions and examples.

4. Uhler, H.S., *Original Tables to 137 Decimal Places of Natural Logarithms for Factors of the Form* 1 ± n $\cdot 10^{-p}$, *Enhanced by Auxiliary Tables of Logarithms of Small Integers*, New Haven, Connecticut, 1942. Besides historical remarks, there are examples and discussion of computation of the tables.

Letters *(Continued from page 4)*

Substitute constants for numeric literals. When the Forth compiler encounters a word that is not in the dictionary, it converts the word to a number and compiles **LIT** and the number for a memory use of four bytes per referenced number. If the number were a **CONSTANT**, only a two-byte reference to the **CONSTANT** would be necessary. The size of a **CONSTANT** is 7 + n where n is the number of digits in the number. Thus, you need four or five references to a particular number before saving code space. Another thing to consider is that **CONSTANT** executes more quickly than does **LIT** (on the 8080, at least). With this as a background, I went through my Forth nucleus source and compiled the list of number usage shown in table one (I started with a Laboratory Microsystems 8080 source, running under CP/M and converted mostly to Forth-79 with some enhancements).

#	Used	#	Used	#	Used
-1	2	011	1	02D	2
0	44	012	2	02E	2
1	17	013	3	030	1
2	7	014	1	03A	1
3	16	015	1	040	5
4	4	016	1	044	1
5	5	018	2	C/L(050)	1
6	2	019	1	05B	1
7	4	01A	1	060	1
8	3	01F	1	07B	1
9	2	BL(20)	12	080(BPS)	5
0A	2	021	2	0A0	1
0B	3	022	4	0C0	1
0C	4	024	1	0CD	1
0D	2	025	1	0FF	1
0E	1	029	1	7FFF	2
0F	3	02A	1	8000	1
010	4	02C	1	A081	1

The FIG model defines 0, 1, 2, 3 and BL as constants, saving 159 bytes. There isn't that much more to be saved in the nucleus by use of this technique, as the table shows, the constant 5 saving two bytes. However, it would probably save *applications* space if all the single-digit numbers were defined as constants, as well as 040 and 080.

Do not compile **EXIT** for any word that never gets to the end of its definition. Instead of defining a new word to terminate a definition without compiling **EXIT**, use **+-2 ALLOT** at the end of the affected definitions (which are **INTERPRET**, **QUIT**, **ABORT**, **COLD** and **WARM**).

Define messages in screens 4 and 5 for the sign-on message in **ABORT**, the sign-off message in **BYE** and the message in **LIST** (provided disk is available). In my system, this would save 3C hex bytes, nothing to sneeze at.

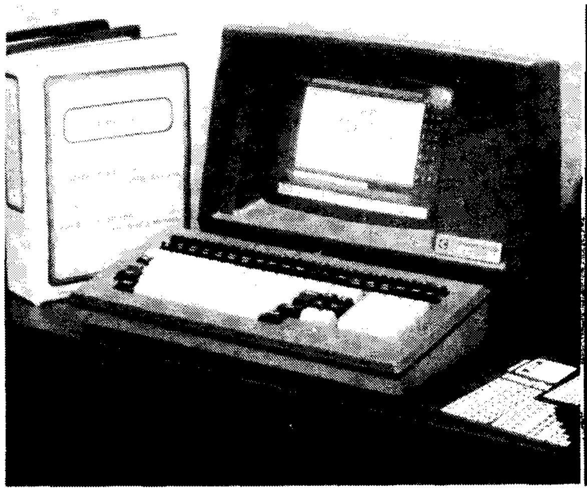
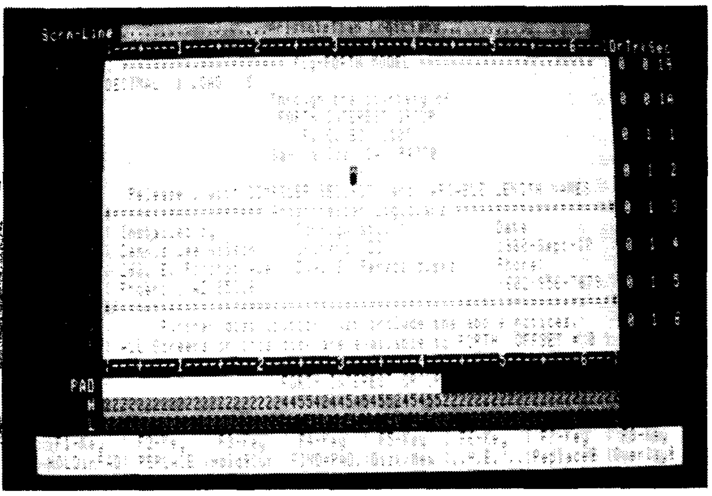
Cross-compile internally used words using headerless code to strip out the name and NFA bytes. This could be done for **LIT**, **BRANCH**, **0BRANCH**, **1BRANCH**, **(LOOP)**, **(+LOOP)**, **(DO)**, **(;CODE)**, **(.")**, **-->** and possibly some others. However, this would make life difficult for such programming aids as decompilers, tracers, debuggers, *etc.*, since they couldn't find those words. Doing this would save fifty-seven hex bytes.

I'm sure others have thought of even different techniques.

Sincerely,

David W. Harralson
11105 Acama St. #4
North Hollywood, CA 91602

(Continued on page 22)

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

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

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

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

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

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

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

Cromemco CDOS formats, Single Side, S/D Density

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

Cromemco CDOS formats, Double Side, S/D Density

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

Other formats are being considered, tell us your needs.

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

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

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

A Paradigm for Data Input

Michael Ham
Iowa City, Iowa

Programs frequently collect data interactively from the user. In some cases, the program should inform the user that an input is unacceptable. For example, if the user wants to delete a record and enters an ID for which no record is on file, the program must let the user know that there is no such record, instead of mysteriously rejecting the input: a silent rejection might lead the user to think that the deletion was accomplished rather than that the ID was bad.

But for cases in which only a few possible entries are valid, a well-designed program will ignore invalid input. Some programs accept bad input and return the message, "INVALID," but if the program knew the input was invalid, it should not have accepted it in the first place. It is more tactful to overlook the user's error.

For example, when the program expects single-character input for menu selection, the user can tell by the absence of action that something went wrong. When this kind of input is refused, the effect from the user's point of view is that all keys are inoperative except those that produce valid input. The words shown in figure one are for single-character input, but they can easily be extended to accommodate two-character or three-character input.

The words **OK?**, **ECHO** and **CAP** are generally useful: **OK?** is used to get rid of input that failed the edit, **ECHO** displays on the screen the input that passed the edit and **CAP** converts lower-case letters to upper case, so that the program will not be case sensitive.

The **CK**xx words can be written as the occasion requires. **CKA-E** checks the input for A (ASCII 65) through E (ASCII 69) inclusive; and **CKA**,1,5 checks for the three specific characters A, 1 (ASCII 49) and 5 (ASCII 53).

The **GET**xx words are also written to order, but they all follow the same pattern: a **BEGIN...UNTIL** loop enclosing **KEY** (which waits for a key to be pressed and leaves the ASCII value on the stack), a **CK**xx on the input and **OK?** to wipe out invalid input. Note that input is not acknowledged (via **ECHO**) until the loop is exited, which occurs only when a valid datum arrives. (Often the screen will be designed to list the valid responses.)

Exercise: Many times, valid input lies in a contiguous range of ASCII values. Write **GETRESP** (l u -- n), which expects on the stack the lower and upper ASCII values for valid input and, like the words in figure one, accepts and acknowledges only valid data.

```
: OK?  ( n f -- 0 or n 1 )   IF 1 ELSE  DROP 0  THEN ;

: ECHO  ( n -- n )   DUP EMIT ;

: CAP  ( n -- n )   DUP 96 >  OVER 123 <  AND  IF  32 -  THEN ;

: CKA-E  ( n -- n f )   CAP  DUP  64 >  OVER  70 <  AND ;

: CK0-9  ( n -- n f )   DUP  47 >  OVER  58 <  AND ;

: CKA,1,5  ( n -- n f )   CAP DUP  65 =  OVER 49 =
  OR  OVER  53 = OR ;

: GETA-E  ( -- n )   BEGIN  KEY CKA-E OK?  UNTIL  ECHO ;

: GET0-9  ( -- n )  BEGIN  KEY CK0-9 OK?  UNTIL  ECHO ;

: GETA,1,5  ( -- n )   BEGIN  KEY CKA,1,5 OK?  UNTIL  ECHO ;

When checking purely for alphabetic input, a simpler definition
of  CAP  is  : CAP  95 AND ;   But note that this definition,
unlike the longer one, wreaks havoc on numeric input.
```

Figure One

Perkel Software Systems

MARX FORTH V1.4
NOW PUBLIC DOMAIN

Marx Forth package includes:
- Complete source code
- Screen editor
- Double number word set
- Forth style macro assembler
- Standard Marx Forth extension word set

Extensions include:
- Case
- *Arguments - Results*
- Printer control
- Cursor control
- File system
- *Disk directories*
- String word set
- Recursion

Internal advancements include:
- Links in front of names
- Fast math
- No names on internal words
- Super fast compiler
- New 83-standard circular DO-LOOP
- DO-LOOP executes 0 times if arguments are equal
- LEAVE leaves immediately
- Multiple WHILEs
- Vocabulary trees without vocabulary links
- Compiler security
- 1 byte relative branches for conditionals
- Smart CMOVE
- Machine code where it counts

TARGET COMPILER INCLUDED

Ever write a great Forth application and then want to sell it without having to sell and support a complete Forth system? With the Marx Forth Target Compiler, you can turn your favorite application into stand-alone headerless code files that don't require Forth in the system to run.

This compiler will compile your existing applications with no modifications to your source program. It handles immediate words, create does words, and multiple vocabularies as gracefully as any other words.

SOURCE CODE LISTINGS.......... $35
CP/M, Northstar DOS, TRS-80, Atari.
Marx Forth written in Marx Forth. Package includes manual and hex dumps. Disk version available for Northstar DOS and CP/M only. $150.

THE *SLICK* PROJECT

SLICK is a new Forth-like super compiler that has all the power of Forth and more. The main design criterion behind SLICK is eliminating the readability problems associated with Forth. Recursive descendant compiler compiles postfix, prefix, and infix notation. New variables eliminate @ and !. The text parsor has been greatly enhanced, eliminating the need to separate words with spaces in some situations. Development is geared towards being able to compile code from other languages as well. *All* SLICK code can be target compiled to stand-alone machine code.

I'm looking for software houses interested in distributing this product. *Anyone interested please write.*

Perkel Software Systems
1452 NORTH CLAY
SPRINGFIELD, MO. 65802
(417) 862-9830 or (417) 883-3709
Consulting Services available

Toward Eliminating Forth Screens

Robert R. Wagner
Portland, Maine

Forth screens are a real hindrance to the program development process. Specifically:

1) They are difficult to work with.

a) Inserting new lines of code soon requires the programmer to move lines from one screen to the next and, in doing so, to have to deal with -->.

b) Inserting code in a line requires care and manual manipulation to avoid losing code at the end of a line.

c) Directory utilities require the user to update a directory list (of screens) whenever a screen or set of screens has to be "bumped" up for an expanding application.

d) An application cannot be loaded by name, or requires a **LOADER** word which links the name to the screen (and also requires updating).

2) Forth screens waste space in disk storage; each space is stored as a separate character. This discourages the writing of code having a structured appearance.

Most commercial Forth systems run under a disk operating system (DOS). Storing Forth code in DOS files greatly improves the situation:

1) The entire application is a single file, stored and retrieved by a (hopefully) descriptive name.

2) The DOS catalog/directory utilities are available for use.

3) The entire file can be edited as a unit, using either the DOS editor or a nice editor in Forth for full text files.

4) In most DOS's, trailing blanks in a line are suppressed (usually by a carriage return) and, in many systems, leading blanks (more than one) are stored as [control code, count] in the disk file.

I know of no commercial Forth system that now works with DOS files in this way, but creating systems which do so will permit eliminating the

```
SCR # 94
.” RUTNES TO COPY DOS FILE TO SERIES OF SCRNS" (   RRW 11/13/83) <
( Must not themselves be DLOADed from a DOS file. )             <
CASES ASSEMBLER BASE @ HEX FORTH DEFINITIONS                     <
: GETDOS ;  0 VARIABLE BLKNUM ( BLOCK #)                         <
0 VARIABLE LINPNTR ( LINE CNTR IN BUFFER)                        <
0 VARIABLE INBLK 13E ALLOT INBLK FCBIN ! ( SETS AN INPUT FCB)    <
0 VARIABLE MAXBLOK ( FOR MAX PERMITTED BLOCK #)                  <
0 VARIABLE DOSFLG ( Nonzero if loading from a DOS file)          <
0 VARIABLE THERE ( adr of 1st byte in assigned load buffer)      <
( Remainng lines on this scrn are system-specific (FLEX-6809}.) <
( Other systms require diff. ASSEMBLER code & systm cll addr(s) <
D406 CONSTANT FMSCALL ( FLEX addr for general file mngmnt use)  <
( Open DOS file for input with addr of name on stack:)          <
: OPENIN ( " DR#.FILENAME.EXT" --  ) 1 FCBIN @ OPEN ;            <
CODE GETCHR .B CLR, FCBIN  LDX, FMSCALL JSR, 0= NOT IF, FFFF     <
  # LDD, ( FFFF ON STACK FOR ERROR ) ENDIF, 98 EXG, PUSHD,  --> <

SCR # 95
.” ACCOUT, EXIT, LINOUT "                          (   RRW 11/13/83) <
 : BUFFRSET ( -- blkaddr ) 0 LINPNTR ! BLKNUM @ BLOCK ;          <
 : NEXTBUFF ( oldbuffaddr -- newbuffaddr ) UPDATE DROP           <
   1 BLKNUM +! BUFFRSET CR ." STACK:" S? ;                       <
( Output accumulated line to buffer: )                           <
: ACCOUT ( -- )  0 BEGIN DUP PAD C@ < WHILE                      <
     1+ DUP PAD + C@ ROT DUP 1+ ROT ROT C! SWAP  REPEAT          <
  DROP 40 PAD C@ - -DUP 0= NOT IF                                <
     0 DO 20 OVER C! 1+ LOOP                                     <
  ENDIF 0 PAD C! LINPNTR @ 3 = IF                                <
     NEXTBUFF ELSE 1 LINPNTR +! ENDIF ;                          <
: EXIT ( -- ) CLOSEIN QUIT ;                                     <
( Try to output the line, close and quit if impossible:)         <
: LINOUT ( -- ) BLKNUM @ MAXBLOK @ > IF                          <
    ." Buffer overow" CLOSE IN QUIT ENDIF ACCOUT ;           --> <
                                                                 <

SCR # 96
.” OVERFLOW, BLANKREST, PUTBACK "                  (   RRW 11/13/83) <
( PAD pntr=41: Walk back to prev BL, line -> buffr, shft remdr) <
: OVERFLOW ( -- )                                                <
  PAD 41 + BEGIN 1- DUP C@ 20 = UNTIL DUP PAD > NOT              <
  IF DROP 40 ELSE DUP BEGIN DUP C@ 20 = WHILE 1- REPEAT          <
    DUP PAD > NOT IF DROP ELSE SWAP DROP THEN PAD -              <
   THEN DUP PAD C! LINOUT 41 SWAP - DUP                          <
  PAD C! DUP PAD 42 + SWAP - PAD 1+ ROT CMOVE ;                  <
                                                                 <
: BLANKREST ( of bytes in buffr) ( -- ) LINPNTR @ 40 * B/BUF     <
  SWAP - DUP ROT DUP ROT 20 FILL +  NEXTBUFF ;                   <
( Restore everything--close file)                                <
: PUTBACK ( -- )                                                 <
   CLOSEIN ' ENCLOSE CFA ' WORD 1E + ! 0 DOSFLG !                <
   ' QUIT CFA ' ERROR 25 + ! ( put vectr to QUIT back in ERROR) <
     7FFF THERE @ 2- ! ( mark the buffer as unused) ;        --> <

SCR # 97
.” WRAPUP, ;S, FMSERR NEXTCHR ADDTOBUFF  "         ( RRW 11/08/83) <
: WRAPUP ( everything, depnding on whethr DLOAD OR DGET) ( -- )<
   DOSFLG @ ( fls for DLOAD)                                     <
   IF DOSFLG @ 1- -DUP  IF DOSFLG ! ELSE PUTBACK  THEN 0         <
   ELSE ( not FLOADing) CLOSEIN                                  <
     PAD C@ 0= NOT IF LINOUT ENDIF                               <
      BEGIN LINPNTR @ 0= BLKNUM @ B/SCR MOD 0= * NOT             <
      WHILE BLANKREST REPEAT                                     <
     CR CR ." WARNING--Last NOT FLUSHed!!! " QUIT                <
   THEN ;                                                        <
: ;S ( redef) ( -- ) DOSFLG @ IF PUTBACK THEN R> DROP ;          <
: FMSERR ( -- ) FCBIN @ 1+ C@ DUP 8 = IF ." END-OF-FILE " CR     <
      DROP   ELSE ." FMS ERROR # " DECIMAL . ENDIF WRAPUP ;      <
: NEXTCHR ( -- ) GETCHR DUP 0< IF DROP FMSERR   THEN ;           <
: ADDTOBUFF ( CHR-1 -- ) PAD C@ 1+ SWAP OVER PAD + C! DUP PAD C!<
  64 > IF OVERFLOW THEN ;    ( 062083)                       --> <
```

```
SCR # 98
 ." GETDATA, DGET, NQUIT, ?DELIM "            (      RRW 11/13/83) <
 ( Get and process a single chararacter from the file:  )          <
 : GETDATA ( -- ) NEXTCHR BEGIN DUP DO-CASE                        <
   D CASE DROP  LINOUT NEXTCHR DUP A = IF DROP NEXTCHR ENDIF       <
          END-CASE                                                 <
   20 R > 7F R < + IFCASE DROP ." NON-PRINT CHAR." EXIT END-CASE<
   ( deflt case ) ADDTOBUFF NEXTCHR END-CASES AGAIN ;              <
 ( **MAJOR USER WORD--Get a DOS txt file into series of scrns*:) <
 : DGET ( " DR.NAME.EXT"-3,STRTNG SCR #-2,MAX SCR #-1 --- )       <
   B/SCR * 3 + MAXBLOK ! B/SCR * BLKNUM ! OPENIN BUFFRSET 0 PAD !<
   ." STARTING " GETDATA ;                                        <
 ( New version of QUIT:                     )                     <
 : NQUIT DOSFLG @ IF PUTBACK THEN QUIT ; '                        <
 ( Returns flag=1 if delim=chr, 0 otherwise:              )       <
 : ?DELIM ( delim, chr -- delim,chr,flag )                        <
      OVER SWAP DUP D = IF DROP BL THEN SWAP OVER = ;      -->    <
SCR # 99
 ." DENCLOSE,DLOAD,                     "       (  RRW 11/13/83) <
  ( Revisd form of ENCLOSE for reading frm DOS file:         )   <
 : DENCLOSE ( addr1 delim -- addr1 n1 n2 n3      )                <
   SWAP DROP ( adress) 0 BEGIN DROP NEXTCHR ?DELIM  NOT UNTIL DUP<
   IF ( not a null)0 ROT ROT ( cnt=0,delim,chr) BEGIN             <
     ROT SWAP OVER THERE @ + C! ( Str chr) 1+ ( cnt) SWAP NEXTCHR<
     DUP  IF ( not a null) ?DELIM   ELSE ( a null) 1 THEN         <
    UNTIL DROP DROP ( chr & delim )                               <
   ELSE ( null) SWAP DROP ( delim) THERE @ C! 1 ( cnt) PUTBACK    <
   THEN THERE @ 0 ROT  0 ;                                        <
 ( **** MAJOR USER WORD--load from a DOS (FLEX} FILE: *********) <
 : DLOAD ( " Dr.filename" -- ) 2 DOSFLG ! FO BUFFER THERE !       <
    BLK @ >R IN @ >R 3 BLK ! ( for interp by 0 word)              <
    OPENIN ' DENCLOSE CFA ' WORD 1E + ! ( pnts to DENCLOSE now)   <
     ' NQUIT CFA ' ERROR 25 + ! ( put vector to NQUIT in ERROR)   <
    INTERPRET  CR          R> IN ! R> BLK !  ;     BASE ! ;S      <

 ." ROUTINES TO OUTPUT FORTH FILES TO FLEX TYPE" ( RRW 07/09/83)
  FORTH DEFINITIONS BASE @ DECIMAL
  0 VARIABLE FCBO 318 ALLOT FCBO FCBOUT !
 : SENDSCRN ( 1st scr,curr scr -- )
    OVER OVER = NOT
      IF 0 OVER (LINE) + 16 - DUP C@ 40 =
        IF 16 BLANKS ELSE 2+ 14 BLANKS THEN
      THEN 16 0 DO
                 I OVER .LINE 13 EMIT
               LOOP ;
 : SRCESCRNS 1+ OVER DO
                        I SENDSCRN
                      LOOP ;
 : SENDOUT ( STRTSCR#-3,ENDSCR#-2," DR.NAME.EXT"-1 --- )
   WRITE SRCESCRNS CLOSEOUT ;
  BASE !         ;S
```

standard Forth screen. This author knows of only two previously published moves in this direction. Dr. Donald Delwood in *Forth Dimensions* (IV/3) described a system running under DEC RT11 or RSX11 which links a file to a set of screens (starting at screen one). Apparently, one then **LOAD**s, **LIST**s or **EDIT**s in a conventional way. In a letter in *Forth Dimensions* (IV/2), Derek Vair stated that he intended to do for CP/M-86 files about the same thing described below for FLEX.

The proper approach to this problem requires a redesign of the screen-oriented part of the Forth system. The code presented below is an improper, but helpful, method for doing it with an existing system — in this case, Talbot's tFORTH running under the FLEX DOS. In a proper system, **DGET** would be eliminated and a **LIST** or **EDIT** would operate directly on a DOS file. **SENDOUT** would be replaced by a (very different) **FLUSH** to DOS file or, possibly, an escape-from-edit command. One feature of Forth screens might be desirable: when an application is already in the edit buffer, it should be **LIST**able or **LOAD**able directly from the buffer. Now, on to the code.

First, some explanation of the FLEX/tFORTH-specific words used in the code to follow. You will have to write or incorporate the equivalent words for your system. (*Note: the File Control Block (FCB) is called the File Descriptor Block in DEC, CP/M and other systems.*)

CLOSEIN and **CLOSEOUT** Close FCB whose address is in the user variable FCBIN and FCBOUT, respectively.

FCBIN and **FCBOUT** User variables containing addresses of FCB's normally used for read and write, respectively.

OPEN (*addr1*, flag, *addr2* --) Open DOS file for I/O (addr1 is the address of the length byte of the string containing the DOS file name; flag contains zero for read or one for write; addr2 is the address of the start of the FCB to use).

WRITE Used as "cccc" **WRITE**. File cccc is opened for write using the FCB whose address is in FCBOUT, and the switch in **EMIT** is set to cause characters to be sent to cccc instead of the terminal until **CLOSEOUT** is encountered.

Some **ASSEMBLER** words in tFORTH+:

CODE An assembler macro which sets up the Forth dictionary links (to be followed by the code itself).

NEXT, Compiles code required to link back into the next word of Forth.

PUSHD, Pushes the contents of the D (=A,B) registers onto the Forth parameter stack, then performs **NEXT,**.

The code itself contains three operative top-level words:

1) **DGET** Gets a DOS file into a series of screens.

2) **SENDOUT** Sends a series of screens out to a DOS file.

3) **DLOAD** Loads directly from a named DOS file.

Note that the listing of **SENDOUT** is not of a Forth screen. It is **DLOAD**ed directly from a DOS file.

The key to the entire system is the new loader word **DLOAD**, which operates by substituting the CFA of a new version of **ENCLOSE**, called **DENCLOSE**, in **WORD**. **ENCLOSE** is the word buried in the **INTERPRET** loop which gets the next group of characters for interpretation, whether from a block or from the TIB. In **DLOAD** the following is done:

1) Set a flag, **DOSFLG**, to indicate a **DLOAD**ing in progress.

2) Assign a buffer, with address at **THERE**, for **DENCLOSE** to use for accumulating characters.

3) Save pointers to the source stream in which **DLOAD** appeared.

4) Replace **CFA(ENCLOSE)** by **CFA(DENCLOSE)** in **WORD**.

5) Replace **CFA(QUIT)** by **CFA(NQUIT)**. This could be done once and for all when screen ninety-seven is loaded, but

(Continued on next page)

would cause problems if **NQUIT** were later included in a **FORGET**.

6) Enter the interpret loop.

7) (After leaving the loop, usually via **NQUIT**) replace pointers to the original text stream for continuing normal interpretation.

DENCLOSE is the replacement for **ENCLOSE** during a **DLOAD**. Its structure is:

```
BEGIN
  Get characters from file management system (FMS), dropping delimiters
UNTIL a non-delimiter is found (keep on stack)
IF not a null
  Place successive characters into the buffer whose address is at THERE, terminating when a null or a delimiter is encountered.
ELSE (a null)
  Store it, execute PUTBACK to restore everything for normal operation.
ENDIF
```

Leave parameters on stack to allow code in **WORD** to move the word to locations starting at **HERE**. (These parameters are the same as those produced by **ENCLOSE**, except that S-4 happens to be the *fixed* address for the location of the **ENCLOSE**d word.

In addition to the above null-induced termination (which should not occur in a FLEX text file), other modes of termination are **;S**, the "proper" termination (re-defined on screen ninety-six), and End-of-File condition error from the FMS, dealt with in **WRAPUP** by substituting a null the first time, executing **PUTBACK** the second time.

This solution allows the user to maintain most Forth source code in DOS named files, but it does require one screen for the storage of the **DLOAD** sequences for each application. **DLOAD**ing from a file being **DLOAD**ed is not permissible. My own preference in a final system would be to limit **DLOAD**ing to operation from the terminal or from special files containing only **DLOAD** commands, thus keeping the components of an application "up front," where they can be seen. In any case, the approach presented here represents a useful first step toward an improved system for developing and using Forth code.

Letters *(Continued from page 17)*

Postfix Flag-Waver

Dear Editor,

Intelligent minds question everything. In a recent letter (*Forth Dimensions* V/3), a reader wonders why Forth control structures, such as **IF**, have to employ reverse Polish syntax. He suggests the form:

```
IF (cond) THEN (true) ENDIF
```

In this syntax, **IF** has no function save readability.

We might be tempted to agree that this syntax is preferable from the standpoint of readability, but to ignore the suggestions simply because the weight of Forth tradition is against it. It turns out, though, there's a very good reason for using the reverse Polish syntax: flags can be passed as arguments, using the same mechanism used for numbers (the stack). This allows us to factor out a condition from a definition.

Suppose we have this word, using Forth's traditional syntax:

```
: ?ON/OFF ( t=on -- )
  IF ." ON " THEN ;
```

As you can see, it receives a flag from the stack and makes a decision. In the proposed syntax, we would have to write:

```
: ?ON/OFF ( t=on -- )
  IF THEN ." ON " ENDIF ;
```

leaving nothing between **IF** and **THEN**. Or we could take **IF** out completely, to show that the argument is coming in from outside the definition. But then, where does the **IF** go?

Infix notation for control structures is only appropriate for languages which insist upon passing data and flags via local variables.

Sincerely,

Leo Brodie
17714 Kingsbury St.
Granada Hills, CA 91344

Errata

Dear Marlin,

Since writing about code definitions callable by colon- or code-defined words (*FD* V/3), I have discovered a slight error in the definitions. I have also switched to a 6502-based system. So, in 6502 machine language, the correct definitions are shown below:

```
CODE (CALL) XSAVE STX, 0 JSR,
  XSAVE LDX, NEXT JMP, C;
' (CALL) 3 + CONSTANT CALL-ADDR
: CALL CALL-ADR ! (CALL) ;
: SUBROUTINE CREATE DOES> CALL ;
```

The **ASSEMBLER** vocabulary must be invoked before using **SUBROUTINE** to define words.

Thank you for your interest in my work.

Sincerely,

David Held
P.O. Box 483
Hermosa Beach, CA 90254

Just One EXIT in CASE

Dear FIG:

I enjoyed puzzling over Marc Perkel's letter titled, "Yet Another CASE Statement" in *Forth Dimensions* (V/3). I do, however, have one suggested improvement. If there is no match, the **EXIT** compiled by **ENDCASE** is executed twice. While this does not cause an error in the results, I would prefer if the **EXIT** executed only once. This can be accomplished by replacing **COMPILE DROP** in the definition of **ENDCASE** with **COMPILE R> COMPILE 2DROP**.

```
:ENDCASE
12COMPILE R> COMPILE 2DROP
  HERE SWAP !
  COMPILE EXIT [COMPILE] [ ;
  IMMEDIATE
```

Sincerely,

Ed Schmauch
Conoco, Inc.
P.O. Box 1267
Ponca City, OK 74603

(Continued on page 32)

1984 Rochester Forth Applications Conference

With a Focus on Real-Time Systems

June 6 - 9, 1984
University of Rochester
Rochester, New York

The fourth Rochester Forth Applications Conference will be held at the University of Rochester and sponsored by the Institute for Applied Forth Research. The focus of this year's Conference is on real-time systems, which includes such areas as process control, data acquisition, smart instrumentation, laboratory systems, robotics, computer vision, spacecraft navigation, music synthesis and voice recognition.

Call for Papers

There is a call for papers on the following topics:

1. Real-time software.
2. Forth applications, including, but not limited to: real-time, business, medical, space-based, laboratory and personal systems; and Forth microchip applications.
3. Forth technology, including finite state machines, control and data structures, and hybrid hardware/software systems.

Papers may be presented in either platform or poster sessions. Platform papers will be reviewed for conference direction and paper suitability. Please submit a 200 word abstract by April 1st, 1984. Papers must be received by May 1st, 1984. Abstracts and papers should be sent to the conference chairman, Lawrence Forsley, Laboratory for Laser Energetics, 250 East River Road, Rochester, New York 14623.

Conference Format

Sessions will begin Wednesday, June 6th, and end Saturday, June 9th, and include a group of invited speakers in the area of Real-Time Systems. Submitted papers will be scheduled in platform and poster sessions. There will also be working groups addressing topics of current interest, and demonstrations by participants and vendors. Sessions will be held on the University of Rochester's River Campus.

The registration fee of $300 includes all sessions, activities, meals, and the Conference Proceedings. Full time students may register for $150. Attendees have the option of student dormitory housing at the rate of $100 single/$75 double for five days. Those staying on campus will find a car unnecessary, as there will be a shuttle from the airport and train. A list of nearby motels is also available. For more information, write to Diane Ranocchia, Institute for Applied Forth Research, 70 Elmwood Ave., Rochester, New York 14611 or call 716-235-0168.

Registration Form

Name ____________________

Address ____________________

____

Telephone ____________________

Are you planning a:
10-20 minute talk? ______ poster? ______ demonstration? ______
If you are, we must have your abstract by April 1st, 1984.

Registration fee: $300 ($150 for full time students) ____________________

University housing: dormitory single $100, double $75 ____________________

If double, have you a specific roommate? ____________________
(If not, you will be assigned one. Non-smoking roommate? yes no)

There is a vegetarian meal option. Check if you want it. ____________________

Amount enclosed: ____________________

Please make checks payable to the Rochester Forth Conference. Mail registrations to: Rochester Forth Conference, 70 Elmwood Ave., Rochester, NY 14611.

A More General ONLY

Paul E. Condon
San Carlos, California

I have implemented a version of Bill Ragsdale's **ONLY** in MVP-FORTH on a CP/M machine. (See the Forth-83 Standard, Experimental Proposals.) The implementation is somewhat more general than the proposal published in the Forth-83 Standard, and I offer it for the consideration of, and experimentation by, all interested Forthers.

A special feature of this implementation is that it allows dynamic switching between the three popular methods of handling vocabulary chaining. Using these modifications, one may try things each way and see which is really preferred; one can opt for freedom of choice and consider this approach itself as a prototype. Perhaps, rather than having a standard way of handling vocabulary chaining, one might choose a standard for the way in which the method of chaining is specified.

The essential change that makes dynamic switching easy to implement is a small addition to the parameter list of vocabulary words (see screen #101). MVP-FORTH follows the FIG model in organization and memory management. The header part of a dictionary entry contains a name field of variable length, followed by a link field (fixed-length pointer to the next entry in search order). The body of the entry is a definition of the named word. It contains a pointer to some executable machine code and a parameter list of variable length. In the particular case of vocabulary words, there are two pointers in the parameter field. One points to the first, or head, word in the vocabulary in search order (note: most people would say it points to the *last* word, by which they mean the most recently defined word; I call it the head of the list, and also refer to the tail of the list — the last word examined *in* search order) and the other points to another vocabulary word definition. I have added to these pointers a third

```
SCR #100
( LOAD SCREEN FOR 'ONLY' SYSTEM )

DECIMAL
FORTH DEFINITIONS
210 LOAD      ( UTILITY WORDS FROM MVP )

102 111 THRU ( ONLY etc. )
225 LOAD      ( SUPPLEMENTAL WORDS FROM MVP )
60 LOAD       ( EDITOR VOCABULARY FROM MVP )
90 LOAD       ( ASSEMBLER VOCABULARY FROM MVP )





SCR #101
( ONLY implementation    )

( The new voc.def format has two more words to manage the TAIL
 of the vocabulary list as well as the HEAD of the list.      )

(   nfa:  <voc.name>                                          )
(   lfa:  point to nfa of next word in search order           )
(   cfa:  point to machine code for this word                 )
(   pfa:  A081  dummy word for managing HEAD of list          )
(      +2  point to first word in search order  ( last entered )
(      +4  point to next voc.def.                             )
(      +6  A081  2nd dummy word for managing TAIL of list     )
(      +8  point to pfa, i.e. 1st dummy word of next voc.def
(                                       in search order.      )
(  Items at +6 & +8 are new to this implementation.           )


SCR #102
( an implementation of ONLY             PECondon 10/9/83  )

FORTH DEFINITIONS
0 VOC-LINK !            ( WON'T BE USING OLD VOC.DEF.S ANY MORE )
VOCABULARY &0X        ( A NAME UNLIKELY TO CONFLICT WITH OTHERS )

VARIABLE PERMANENT ( VOC TO WHICH TRANSIENT IS LINKED )
VARIABLE FIG?      ( FIG-MODEL VS. FORTH-79 MODEL )
VARIABLE RELINK?   ( Ragsdale's ONLY, or not )
VARIABLE 'FORTH    ( POINT TO PFS+2 OF new FORTH )
VARIABLE 'ONLY     ( POINT TO PFA+2 OF ONLY )



```

```
SCR #103
( WPICK, WBEFORE, WREMOVE, WATTACH  )
( WORDS TO PICK A WORD-DEFINITION OUT OF CONTEXT AND PUT  )
(  IT AT CURRENT.  )


 : WBEFORE CONTEXT @
   BEGIN DDUP @ DUP IF - ELSE DDROP DROP 0. THEN
     WHILE @ PFA LFA REPEAT ;
 : WREMOVE SWAP PFA DUP LFA @ ROT ! ;
 : WATTACH CURRENT @ @ OVER LFA ! NFA CURRENT @ ! ;
 : WPICK [COMPILE] ' NFA WBEFORE WREMOVE WATTACH ;
```

```
SCR #104
( ALSO, TOSS, X-MODE )

 WPICK IMMEDIATE
 WPICK DEFINITIONS
 : ALSO CONTEXT @ PERMANENT ! ;
 : TOSS PERMANENT @ DUP ' &OX 2+ = NOT
                    OVER 'ONLY @  = NOT AND
     IF 6 + @ 2+ PERMANENT ! THEN ;

 :  FIG-MODE  -1 FIG? !  0 RELINK? ! ;
 :    79-MODE   0 FIG? !  0 RELINK? ! ;
 : ONLY-MODE   0 FIG? ! -1 RELINK? ! ;
 FIG-MODE
```

```
SCR #105
(  <REM-VOC>    )

 : <PCHK>   (  CHECK A VOC.POINTER IN <REM-VOC>  )
   (   NXT, PFA, PNT --- NXT, PFA )
   DUP @ CFA 3 PICK = IF 3 PICK 2+ SWAP ! ELSE DROP THEN ;
 : <REM-VOC>   ( PFA --- )
   DUP 8 + @ SWAP                         ( --- NXT, PFA )
   PERMANENT <PCHK>
   VOC-LINK @                           ( --- NXT, PFA, PTR )
   BEGIN        ( LOOK FOR VOC.S THAT ARE LINKED TO THIS ONE )
     DDUP 4 + @ =
     IF 3 PICK OVER 4 + ! THEN  ( LINK TO SUCCESSOR )
     @ DUP 0=
   UNTIL DROP DROP DROP ;
```

one that is used to point to the next vocabulary *in search order*. (Also note: the A081 is a "magic" trick that makes the entry "look like" a word definition header to words that search the dictionary.)

In the FIG model and in Forth-79, the tail links to the parameter field of some other vocabulary word. In Ragsdale's work, the tail word in each vocabulary is "closed," *i.e.* contains a null pointer in the link field, and higher-level management of the search order is done in a push-down stack. I have replaced this with a simple link of the tail word back to the new pointer in the vocabulary word definition. With this change of structure in place, it is easy to re-connect and re-arrange vocabularies by manipulating the pointers in the parameter fields of the vocabulary words. The link of pfa + 4 was originally implemented to support a smart **FORGET**. Now it is useful in supporting other functions whose domain is all vocabulary definitions.

In the **ONLY** approach, a distinction is made between transient and resident vocabularies. My code uses the variable **PERMANENT** to support this distinction. The word **ALSO** sets **PERMANENT** equal to **CONTEXT** and, as a consequence, the transient vocabulary becomes part of the resident list of vocabularies.

The word **TOSS** was suggested by George Shaw (*Dr. Dobb's Journal*, September 1983). It removes the first vocabulary from the resident list and places it at the end of the transient list. My version is not quite what Shaw specified, because the previous contents of the transient vocabulary are not lost in my **TOSS**.

The word **VOCABULARY** is re-defined so that it builds the new, longer vocabulary word definition. It also contains some conditional branching so that it can support the FIG model, the Forth-79 Standard and Ragsdale's **ONLY** approach to vocabulary chaining. **RELINK?** is a flag variable that controls whether or not a vocabulary is re-connected when it is invoked. **FIG?** is a flag variable that controls the initial linking that is set up when a new vocabulary word is defined.

Other words are:

ORDER Prints the search order of vocabularies.

VOCS Prints a list of all vocabulary words.

SEAL Removes **ONLY** from the search order.

ONLY A special vocabulary which removes all other vocabularies from the search order when it is invoked, and which automatically invokes **ALSO**.

One complication that deserves special mention is the vocabulary **&OX**. In MVP-FORTH there is a magic word, called **X**, whose name in the dictionary is really a single ASCII null. This word must *always* be in the search order, if **INTERPRET** is to terminate correctly. Rather than placing a copy of **X** in every vocabulary, I have removed **X** from **FORTH** and placed it in a special vocabulary. I called this vocabulary **ALWAYS** for a while, but decided it didn't deserve tying up a good English word. So I gave it a name that isn't too likely to conflict with any application, but is slightly indicative of why it is there.

One consequence of this work is a much improved understanding of vocabularies on my part. When several vocabularies are defined, they are always placed in a tree structure. **CONTEXT**, **CURRENT** and **PERMANENT** are pointers to particular places in this tree. It is a tree with a single root, because all vocabularies must link eventually to **&OX**, which contains the only copy of **X** in the whole dictionary. In my implementation of **ONLY**, the tree is re-arranged in a two-step process. First, **<REM-VOC>** removes the selected vocabulary from the tree, and all vocabularies that were linked to it are linked instead to the following vocabulary. Then the free vocabulary is linked to the vocabulary pointed to by **PERMANENT**. The removal step ensures that a circular linking can never be formed. Because there is no stack of pointers to vocabularies, there is no need to enforce a limit to the number of vocabularies in the resident list, and I have not implemented any limit. If a vocabulary is already correctly linked when it is invoked, no action it taken.

```
SCR #106
( new VOCABULARY   )

HEX ' VOCABULARY NFA 20 TOGGLE ( SMUDGE OLD VERSION )
: VOCABULARY
  CREATE  A081  ,  HERE  4  +  ,
  HERE  VOC-LINK  @  ,  VOC-LINK  !
  (  NEW STUFF:  )
  A081  ,  FIG?  IF  CURRENT  ELSE  'FORTH  THEN
  @ CFA ,

DOES> RELINK? @
  IF DUP 8 + @ PERMANENT @ CFA = NOT ( NEED RELINKING? )
    IF DUP <REM-VOC> PERMANENT @ CFA OVER 8 + ! THEN
  THEN ( OLD STUFF: ) 2+ CONTEXT ! ;
DECIMAL

```

```
SCR #107
( ONLY, SEAL, WORDS, ORDER  )
FORTH DEFINITIONS VOCABULARY ONLY ' ONLY 2+ 'ONLY !
: SEAL  ' ONLY <REM-VOC> ;
: WORDS  [ ' VLIST CFA ] LITERAL EXECUTE ;
: ORDER
  CR ."   context -> " CONTEXT @ CFA
  BEGIN
    DUP PERMANENT @ CFA = IF CR ." permanent -> " THEN
    DUP NFA COUNT 31 AND DUP
     OUT @ + C/L > IF CR THEN
    TYPE SPACE
    8 + @ DUP ' &OX =
  UNTIL DROP
  CR ."   current -> " CURRENT @ CFA NFA ID. SPACE ;
: VOCS VOC-LINK @ CR BEGIN DUP 4 - NFA ID. SPACE @ DUP 0=
 UNTIL DROP ;
```

```
SCR #108
( new FORTH and some magic relinking #1 )
DECIMAL
CURRENT @ CFA NFA 32 TOGGLE    ( SMUDGE OLD VERSION)
VOCABULARY FORTH               ( USING NEW FORMAT )
' FORTH 2+ DUP 'FORTH !        ( BIND 'FORTH       )
LATEST SWAP !                  ( COPY HEAD POINTER)
FORTH DEFINITIONS ALSO         ( INVOKE NEW DEFINITION )
' &OX ' FORTH 8 + !            ( LINK FORTH TO &OX )








```

```
SCR #109
  0 ( work on X and ! linkings )
  1
  2  ' XOR  ( X FOLLOWS XOR IN MVP-FORTH SEARCH ORDER )
  3  LFA DUP
  4      @ DUP
  5        ' &OX 2+ !  ( PUT X INTO &OX VOC. )
  6        PFA LFA DUP ( POINT TO LF OF X )
  7               @ ROT !  ( REMOVE X FROM FORTH )
  8               0 SWAP ! ( SEAL   X           )
  9
 10  ' FORTH 6 + ' ! LFA ! ( LINK ! BACK TO FORTH )
 11
 12
 13
 14
 15


SCR #110
  0 ( move new words into the ONLY vocabulary  )
  1
  2  LATEST ' ONLY 2+ ! ( ONLY HEAD POINTS TO NEW WORDS )
  3  ' &OX  ' ONLY 8 + ! ( ONLY TAIL POINTS TO NULL WORD VOC.  )
  4  ' ONLY 6 + ' &OX LFA ( LINK TAIL OF NEW WORDS TO ONLY, WAIT )
  5  ' ONLY ' FORTH 8 + ! ( LINK FORTH VOC. TO ONLY VOC. )
  6  DUP @ CONTEXT @ ! ( REMOVE NEW WORDS FROM CONTEXT )
  7  !   ( DO LINE 4, WHICH WAS DEFERRED )
  8
  9  ONLY-MODE ONLY DEFINITIONS
 10  : SUL [ ' ONLY CFA ] LITERAL EXECUTE ; ( NEW NAME FOR ONLY )
 11  : ONLY RELINK? @
 12    IF [ ' &OX 2+ ] LITERAL PERMANENT ! ( ZAP PERMANENT    )
 13         ONLY   ( INVOKE VOCABULARY )  ALSO
 14    ELSE ONLY THEN ;
 15  ONLY FORTH ALSO FIG-MODE

SCR #111
  0 ( PATCHES TO POINT TO NEW FORTH )
  1
  2  FORTH DEFINITIONS
  3  HEX : X! >R BEGIN R@ SWAP ! DUP WHILE REPEAT DROP R> DROP ;
  4   0 11DE ( in COLD ) 193C ( in FREEZE ) ' FORTH X!
  5   0 706 ( in <ABORT> ) 18AD 18BB ( in FORGET ) ' FORTH CFA X!
  6   FORGET X!
  7  DECIMAL
  8
  9
 10
 11
 12
 13
 14
 15
```

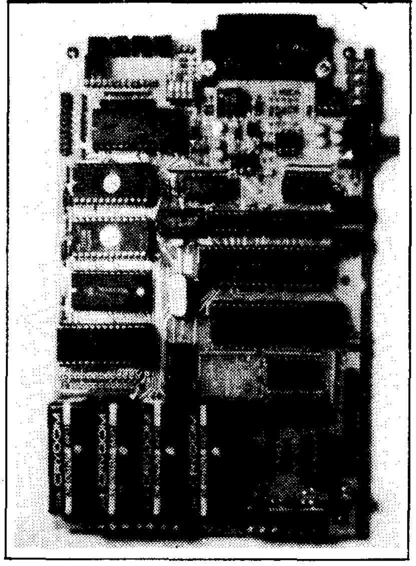
In building this system, I first compiled the definitions for the new words into the **FORTH** vocabulary and then manipulated the links so that they are moved to the **ONLY** vocabulary. By doing this, I avoided using a meta-compiler (which I don't have). The only really tricky part was moving **X** in one fell swoop, so that **INTERPRET** never noticed that it was temporarily missing. To do some of the moving of definitions, I wrote **WPICK**. The word ticks the following word in **CONTEXT** and re-connects the links so that it becomes the first word in search order at **CURRENT**.

Finally, I found five places in MVP-FORTH where other words contain compiled references to the word **FORTH**. Since I have introduced a new version of this word, I patch these references in screen #111. Naturally, these patches are implementation-specific and will have to be changed by anyone attempting to use this work on another system.

My first impression of working with **ONLY** is that I like it, but that I am having trouble remembering to use **ALSO** when it is needed.

Introducing
PROATROL™

The PROATROL™ single board computer is both a low cost development system for programming in ***FORTH***, and a powerful target system for a wide range of real time control applications.



The 5" x 8" stand alone board is based on the Rockwell single chip* FORTH *Microcomputer which contains a* FORTH *operating system in its internal ROM.

- RS 232 and 20 ma serial interface
- 4 sockets for plug in I/O modules (Opto 22, Crydom, etc.
- 4 half bridge power drivers
- 8 optically isolated inputs
- 8 channel 8 bit ADC, radiometric or referenced
- 4 potentiometers for level/speed settings
- 4 position dip switch
- 4 Schmitt trigger inputs
- 4 open collector outputs
- 8 expansion I/O lines
- Power fail interrupt/reset
- In circuit EPROM and EEPROM programming
- +5v regulator, DC/DC convertor
- Buss expansion connector
- Sockets for up to 14kb of memory

PROATROL™ has it all together!

Develop your program, configure your I/O options, debug your system and GO FORTH!

Only $495.00

To order send remittance to:

PROA Corporation
4019 EDITH BLVD. NE BLDG. 2B
ALBUQUERQUE, NM 87107
For further information call
(505) 344-2106

Recursion of the Forth Kind

Michael Gwilliam
Ronald Zammit
Arcata, California

Many of us have used Forth in unusual ways, as it allows many innovative solutions. A demonstration, often shown to novices, is the redefining of a word using the same word. What is presented here is the strange case of a *new* word being used to define itself. It is being employed while it is being defined! Sound impossible? Well, let's look at such a word, keeping in mind that this may serve no practical use except to test your understanding of Forth.

Most of us have seen the model for ; in the *Systems Guide to fig-FORTH.* It is given as in figure one.

```
: ; ?CSP COMPILE ;S
SMUDGE [ ; IMMEDIATE
```

Within this model, the ; is used, implying that it is already defined. Now let's look at another model in figure two, called ;; to avoid confusion with the previous example.

```
: ;;
 ?CSP COMPILE ;S SMUDGE
 [COMPILE] [ [ SMUDGE
 IMMEDIATE ] ;; SMUDGE
```

Notice that this word does not have a ; terminating it. This word will work in the same manner as the ; but uses itself when defined. Let's examine how this works:

: ;; Create a dictionary header named ;; and set **STATE** to 1 to compile.

?CSP Compile **?CSP**.

COMPILE Compile **COMPILE**.

;S Compile **;S**.

SMUDGE Compile **SMUDGE**.

[COMPILE] [Force compilation of **[**.

[Set **STATE** to 0, stop compilation and begin execution.

SMUDGE Toggle (set) the smudge bit to allow ;; to be found in the dictionary, even though it is not a complete definition.

IMMEDIATE Set the precedence bit, forcing execution of ;; when compiling.

] Set **STATE** to 1 to begin compiling.

;; The interpreter looks for the definitions of ;; and can locate the uncompleted definition, as it is smudged. Since the word ;; is immediate, it will execute.

At this point, the words compiled under the ;; header are executed.

?CSP check stack pointer.

COMPILE Compile the next word, which is...

;S The primitive **;S** is added to the end of the definition of ;; ending its definition in the same manner as all ; definitions.

SMUDGE Toggles the smudge bit of ;; making the header unlocatable during a dictionary search.

[Set **STATE** to 0, stop compilation and begin execution.

Now the next word in the input message buffer is executed.

SMUDGE Again toggles the **SMUDGE** bit, making the header locatable during a dictionary search.

Once ;; has been entered, it may be used to define new words in the same manner as ; is used. While this is a tricky and redundant definition, we hope that your understanding of ;; will add to your overall appreciation of Forth.

Quick Sort in Forth

Marc Perkel
Springfield, Missouri

The famous Quick Sort, or partition sort, is generally considered the fastest general purpose sort (hence its name). The basic concept behind it is relatively simple. Suppose we had an array of numbers from one to 100 in random order. If we were to sort these numbers using the Quick Sort, we would first select the middle number which, let's say, was a seventy-two. Then we scan from the beginning for a number that is not less that seventy-two.

Once we have found one, we scan backwards from the end for a number that is not greater than seventy-two. When we have found these two numbers, we exchange them. Then we continue scanning and exchanging until, somewhere in the middle, the two pointers cross. At this point we have divided the array into two arrays, the lower one with all the numbers less than seventy-two and the upper one with all the numbers greater than seventy-two. We now repeat the process on both arrays, creating four, then eight, then sixteen, *etc.* arrays.

As the original array is divided into more and more arrays, the size of these pieces gets smaller and smaller. When a piece gets down to one item, it is considered sorted and is no longer dealt with. Thus, the process doesn't go on forever. When the array is divided completely into one-byte pieces, the sort is complete.

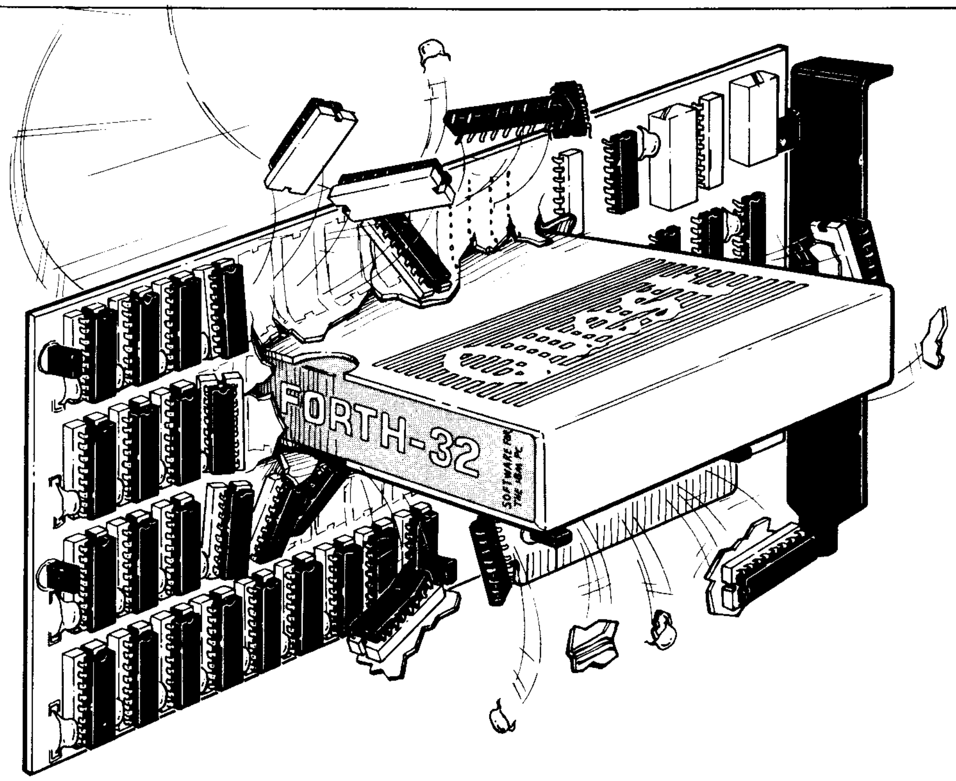
Lines five through 0A contain the code for one dividing pass. This pass leaves four addresses on the stack, which represent the ends of two smaller arrays. The rule is to sort the smaller array first. This prevents the recursion from going any deeper than the logarithm of the number of items to be sorted. If you were sorting 1000 items, nine levels is as deep as you can go. Lines 0B and 0C bring the smaller array up first.

With the ends for the two arrays on the stack, the sort calls itself, thus treating each array just like the original array. These, then, each call themselves, as long as the pieces have any sortable size.

Recursion of this kind is handled gracefully by Forth because of its two stacks. Each recursion leaves a pointer to the unfinished work of the rerun stack, while the second array to be sorted accumulates on the data stack. Most other languages would require an array to be created to accumulate the second array, simulating the data stack.

For those who are using FIG-Forth and don't have a word **MYSELF**, the phrase **[SMUDGE] SORT [SMUDGE]** will work instead. This sort is written to sort bytes and can easily be re-written to sort anything you want. This sort can be displayed in a very impressive manner by those who have memory-mapped video. (This means TRS-80 owners.) You can block move RAM to screen memory and sort the screen. This allows you to watch it happen. Lots of fun!

```
Screen A0 (*) 160
  0  ( Recursive Quick Sort - Sorts an array of bytes. )
  1
  2  VARIABLE MIDDLE
  3  : EXCHANGE 2DUP C@ SWAP C@ ROT C! SWAP C! ;
  4  : SORT ( start end - )
  5     2DUP 2DUP OVER - 2/ + C@ MIDDLE !        ( pick middle one )
  6        BEGIN                           ( scan right then left )
  7          SWAP BEGIN DUP C@ MIDDLE @ < WHILE 1+ REPEAT
  8          SWAP BEGIN DUP C@ MIDDLE @ > WHILE 1- REPEAT
  9          2DUP > NOT IF 2DUP EXCHANGE 1 -1 D+ THEN
  A          2DUP >                      ( until partitions cross )
  B        UNTIL SWAP ROT                     ( sort both pieces )
  C     2OVER 2OVER - ROT ROT - < IF 2SWAP THEN
  D     2DUP < IF MYSELF ELSE 2DROP THEN         ( smallest first )
  E     2DUP < IF MYSELF ELSE 2DROP THEN ;     ( then large piece )
  F
```



Break Through the 64K Barrier!

FORTH-32™ lets you use up to one megabyte of memory for programming. A Complete Development System! Fully Compatible Software and 8087 Floating Point Extensions.

Quest™

Quest Research, Inc.

303 Williams Ave.
Huntsville, AL 35801
(205) 533-9405

Call today toll-free or contact a participating Computerland store.

800-558-8088

Now available for the IBM PC, PC-XT, COMPAQ, COLUMBIA MPC, and other PC compatibles!

IBM, COMPAQ, MPC, and FORTH-32 are trademarks of IBM, COMPAQ, Columbia Data Products, and Quest Research, respectively.

Within WITHIN

Gary Nemeth
Rocky River, Ohio

The word **WITHIN** is not standard, yet it shows up in many applications — it performs a range check. Because it is not standard, there are several ways it is defined. The parameters are always the value to test, and the two range limits, resulting in a true/false on the stack. Several different versions of the word are defined and used in the attached console listing. They are shown on a Forth-79 system, and work on other types of systems as well. Standard words are proposed in conclusion, as well as a math puzzle.

The word **WITHIN1** is the first definition which comes to mind. The test value comes first (lowest on the stack), followed by the low and high range values. Take a look at the code before reading on. After **WITHIN1** has executed, a true value on the stack indicates that the test value was within range. This version shines when used to edit dates or analog control settings. The phrase **DUP 1 12 WITHIN1** keeps the month value, then stacks true if okay. Likewise, **AGE 13 19 WITHIN1** establishes a teenager.

We could have stopped right there, and that **WITHIN** would have become standard. But two forces have caused other versions to appear: Forth **DO...LOOP** parameters use a limit value one higher than the highest manifest index (because this is very convenient when **ALLOT**ing and using zero-origin subscripts), and the words **?ERROR** and **ABORT"** use a true value to signal error.

The word **WITHIN2** uses this limit idea. Give that word the value to test, the lowest allowed value, then one more than the highest value. The result is then, algebraically: low $\leq$ n $<$ limit. The month phrase becomes **DUP 1 13 WITHIN2 IF . ." IS A MONTH " THEN**. This seems to be what Forth, Inc. uses.

Now suppose you wish to **ABORT"** when a range check fails. The sequence **WITHIN2 NOT** yields a true when the range check fails, so **ABORT"** or **?ERROR**

```
SCR # 166
 ( WITHIN WITHIN )

 : WITHIN1 ( n\low\high   -- true ok )
              >R  1- OVER <   SWAP R> 1+ <  AND ;
 : WITHIN2 ( n\low\limit  -- true ok )
              >R  1- OVER <   SWAP R>   <  AND ;

 : WITHIN3 ( n\low\limit  -- true bad )
              >R  OVER >   R>  1- ROT  <   OR ;
 : WITHIN4 ( n\limit\low  -- true bad )
              >R  1- OVER <   SWAP R>  <   OR ;

 : W1      6 -5  CR DO I . I -2 1 WITHIN1 . SPACE LOOP ;
 : W2      6 -5  CR DO I . I -2 2 WITHIN2 . SPACE LOOP ;
 : W3      6 -5  CR DO I . I -2 2 WITHIN3 . SPACE LOOP ;
 : W4      6 -5  CR DO I . I 2 -2 WITHIN4 . SPACE LOOP ;
3  DUP  1 12 WITHIN1 . .     1 3  OK
12 DUP  1 12 WITHIN1 . .     1 12  OK
12 DUP  1 12 WITHIN2 . .     0 12  OK
12 DUP  1 12 WITHIN3 . .     1 12  OK
12 DUP  12 1 WITHIN4 . .     1 12  OK
W1 W2 W3 W4
-5 0  -4 0  -3 0  -2 1  -1 1  0 1  1 1  2 0  3 0  4 0  5 0
-5 0  -4 0  -3 0  -2 1  -1 1  0 1  1 1  2 0  3 0  4 0  5 0
-5 1  -4 1  -3 1  -2 0  -1 0  0 0  1 0  2 1  3 1  4 1  5 1
-5 1  -4 1  -3 1  -2 0  -1 0  0 0  1 0  2 1  3 1  4 1  5 1   OK
: ERRANT  >R  1- OVER <  SWAP >R <  OR ;  OK
FORGET ERRANT  OK
: ERRANT  >R  1- OVER <  SWAP R> <  OR ;  OK
0 0 0 ERRANT . 1  OK
0 1 0 ERRANT . 0  OK
: WITHIN  1+ SWAP ERRANT  NOT ;  OK
 0 1 12 WITHIN . 0  OK
 1 1 12 WITHIN . 1  OK
12 1 12 WITHIN . 1  OK
13 1 12 WITHIN . 0  OK
  OK
```

would then report the error and **ABORT**. More kindly, the sequence **WITHIN2 NOT IF** could be employed to correct the errant value. These are frequent scenarios for editing input, and bring the definition **WITHIN3**. It is identical to the one for **WITHIN2** followed by **NOT**.

My personal preference is **WITHIN4**, which I call **ERRANT**. This word takes the range limit values in the same way **DO** takes its parameters, and leaves its result ready for **ABORT"**. Define a word which brings the **DO** parameters to the stack, such as **: PARTS 39 0 ;** Then the phrases **PARTS DO** or **PARTS-.** or **DUP PARTS ERRANT ABORT" OUT OF RANGE! "** come naturally.

An aside: the definitions of **WITHIN2** and **WITHIN4** are mysteriously similar. They provide a rare glimpse of a non-trivial theorum result. Could George Boole have proved the definition of **WITHIN4**, given that of **WITHIN2**?

I haven't seen a definition where the test value comes last, such as (low-limit n -- tf). Range limit values seem like constants and are conveniently specified after the test value. Otherwise, one would code **DUP 1 13 ROT WITHIN** *etc.*, which is more work than necessary.

One of these words should become standard, or mentioned in the optional word set for standard usage. The first one or the last seem to be better choices than the others. If **ERRANT (WITHIN4)** and **WITHIN (WITHIN1)**, as shown last in the listing, were standard, then newcomers could more easily structure their applications.

Letters *(Continued from page 22)*

A More General **CASE**

Dear *Forth Dimensions*:

Dan Lerner's "**CASE** as a Defining Word" inspired me to stir the pot one more time. My version allows for range testing at the expense of one extra cell per case. Quite simply, mine works thus:

The PFA points indirectly to the **OTHERWISE** function to be executed. It points directly to the limit of a +6 **LOOP** which steps through a table testing the case index. An example will make this clear — the code in figure one compiles as shown in figure two.

BETWIXT compiles exclusive limits for testing and a CFA. **OF** sets up exclusive limits for a single case and uses **BETWIXT**. **OTHERWISE** compiles the CFA of the default case and puts a pointer to it in the PFA of the structure.

I plan to implement the **DOES>** portion of **CASE** as a **;CODE** structure. As is, though, the structure is transportable across Forth systems. FIG model systems will use an extra cell because of the implementation of the **DOES>** structure, but will run just the same.

Modifications that could be made to this code include: inclusive range testing, testing for outside of a range, unsigned tests, machine code (for speed) and error checking in **CASE** and **OTHERWISE** to protect against incomplete structures (for instance, a **CASE** structure with only the **OTHERWISE** part could crash the system at run time).

Martin Schaaf
32 Crest Rd.
San Anselmo, CA 94960

Figure One

```
         CASE       TEST
      5  OF         DUP
    3 7  BETWIXT    SWAP
         OTHERWISE  DROP
```

```
nTEST1111cccc AAAA 0006 0004 (DUP) 0007 0003 (SWAP) (DROP)
              |--------->---------------->-----------|
NFA  LFA CFA  PFA
```

Figure Two

```
SCR #17
( CASE AS A DEFINING WORD     MAS 83.10.28 )
: CASE CREATE HERE 0 , DOES> DUP @ @ ROT ROT
       DUP @ SWAP 2+ DO DUP I 2@ WITHIN IF
                         2DROP I 4 + @ 0 LEAVE THEN
       6  +LOOP  DROP EXECUTE ;
: BETWIXT  , , [COMPILE] ' CFA , ;
: OTHERWISE  HERE SWAP ! [COMPILE] ' CFA , EXIT ;
: OF 1 - DUP 2+ BETWIXT  ;
EXIT

( Note, WITHIN performs an exclusive range test where the
  stack picture is thus:  ( n, lower.limit, upper.limit - f )

```

Slash Through Complexity with . . .

LEVEL I — *THE ELEGANT EDGE*™

A Step Beyond Language — "A Programming Environment"

- **FORTH 83 Compatible**
- **Enhanced Graphics**
- **Expanded Wordset**
- **Transportability**
- **Full I-O Redirection**
- **Interpret Buffer**
- **Operating System Compatibility**
- **Faster Compilation and Execution**
- **Complete Documentation**

$175⁰⁰

Telosx and Elegant Edge are trade marks of Rising Star Industries.

For more information, fill out this coupon and mail to:
Rising Star Industries
24050 Madison, Suite 113
Torrance, CA 90505 (213) 378-9861

Name: ______________________

Address: ______________________

City: ______________________

State: ______________ Zip ________

Phone (   ) ______________________

FORML 1983: A Review

The Fifth FORML Conference took place, as usual, in the lovely Asilomar Conference Center near Monterey, California. Of the seventy attendees, twenty were guests who came simply to enjoy the Thanksgiving holiday among friends, deer and racoons, separated from the Pacific Ocean only by a few yards of sand dunes and rare cypress. The setting was spectacular and the program of keen interest, but there is no doubt that much of the value of the annual affair lies in the interaction between attendees. The conference atmosphere and the calm physical surroundings create an environment in which ideas can grow and find constructive criticism. Interchanges can be rapid-fire and direct, and participants benefit from the broad perspective found among fellow attendees.

All this is, naturally, leading somewhere. FORML is much more than listening to dry recitations for three days. To find out just how much more it is, you really have to go and participate in the collision of opinions and facts, experiences and philosophies. That experience would seem to be as much a part of FORML as the sharing of knowledge, techniques and code.

Following are the recollections of several conference participants. The large amount of material presented at FORML precludes anything but a thumbnail sketch in these pages. As always, the papers will be published in a book later this year, along with additional material not yet seen — even attendees who received a conference binder should keep an eye out for the published proceedings.

Forth Implementations
Reviewed by Roger Wallace

Four papers on Forth implementations were contributed to this session: 1) Timothy Huang, "First Chinese Forth — A Double-Header Approach." 2) Ray Duncan, "8086 Forth+ — A 32-Bit Forth Implementation for the Intel 8086/88 Microprocessor." 3) Michael A. Perry, "F83 — A Public-Domain Model Implementation of the Forth-83 Standard." 4) Harvey Glass, "The Implementation of Extensions to Provide a More Writable Forth Syntax."

Ray Duncan was not able to give his paper but, fortunately, his written explanation of his thirty-two-bit 8086/88 implementation is fairly detailed. He explains why this thirty-two-bit Forth+ runs slower than the 16-bit 8086 implementation.

Huang's paper on Chinese Forth was delivered by Mike Perry, who has worked in Taiwan on this project, and who even attended a FIG meeting there. This implementation is called, "Dai-E Forth, Level II," and it adheres to the '83 Standard. Mike gave a very lucid explanation, which is largely contained in the written report. The goal of this work is to produce a "word processor" in Chinese in Forth. Since grade schools are attended six days a week for ten hours per day, there is ample time for extensive computer training. While there is a phonetic alphabet used in the earlier grades, the goal of this work is to provide software for adult use, which can manipulate the classic Chinese characters.

It seems that most Chinese adults have largely forgotten, or at least ceased to actively use, the phonetic alphabets. There is a different thirty-seven-letter and five-tone symbol phonetic alphabet used on Taiwan and on the mainland. The alphabet used on Taiwan is called the "BER PER MER FER," after the sound of the first four characters. Each English Forth word was translated to a Chinese phrase of more than two characters, in order to avoid the homonym (different character with identical pronunciation) problem, which is severe in Chinese. The English math operators and characters (such as ; : , and [ ) are preserved.

A separate header and body approach proposed by Klaus Schleisiek (*Forth Dimensions* II/5) allows a double set of headers, one English and one Chinese, to be used. A complete list of the Forth-83 Standard words is included, with both the Chinese character and the Chinese phonetic letters listed.

Then Mike Perry changed hats and presented his own contribution, describing the public-domain model of the Forth-83 Standard. He described the philosophical reasoning which induced him to provide a relatively sophisticated public-domain model (compared to the bare-bones F79 model). He is hoping to derive greater public acceptance for F83, since a new user can immediately have available a good full-screen editor, assembler, debugger, decompiler, source screen locator, compile error locator, shadow-screen manager, screen printing utility and other utilities which will make one's indoctrination to Forth much more pleasant. About forty bugs have been found in this '83 Model, none of which are serious. Next Spring, Mike will issue a version correcting as many bugs as are reported by that time.

F83 even includes a simple multitasker. All told, there are 420 screens. The blocks are embedded in CP/M files. An important feature of this very interesting new F83 model is that it is issued by an organization founded by Mike, called "NO VISIBLE SUPPORT SOFTWARE." It is his belief that this is the only way he can provide this software without spending the next few years on the telephone, holding hands.

Professor Harvey Glass gave an excellent talk on extensions which make a more writable Forth syntax. These extensions provide a programming syntax which requires no explicit parameter stack operations. There is straightforward recursive capability and a simple way of handling forward references. These extensions are implemented entirely in high-level Forth, and make the new dialect read much like a reverse-Polish version of Pascal or ALGOL. The result is both more readable and more writable. The extensions to Forth and their implementation, described in this paper, provide a dialect of the language that is useful for both veteran Forth programmers and for those not now familiar with Forth.

Naturally, there was extensive discussion between the audience and Mike Perry and Professor Glass about these extremely interesting presentations.

Programming Techniques
Reviewed by Robert Berkey

As an implementor of Forth systems, the conference session on programming techniques will continue to hold special interest for us. That this is true for others was indicated by the large number of papers (nine) presented by the authors and distributed in the conference binders.

Mike Perry's paper, "A Simple Multi-Tasker for Forth" presented the multi-tasker drawn from the public-domain system he created with Henry Laxen. For a public-domain system, multi-tasking is a notable feature. In classic Forth style, this multi-tasker uses round-robin, as opposed to time-slice, scheduling. One of the exciting innovations here are two words, **MULTI** and **SINGLE**, which enable and disable the round-robin loop.

Our own paper, "**COMPILER** and **INTERPRETER** Co-routines," develops an idea from Robert Patten. He is one of the wellsprings of Forth innovations, but, because he is so generous with his ideas, gets less credit than he surely deserves. This particular idea addresses problems experienced by systems in which **]** (right bracket) is used as the compiler loop. First impressions are that this technique of making the compiler loop and the interpreter loop work as co-routines may entirely replace the usage of right bracket as the compiler loop. It will also get usage in systems in which the two loops had been combined, as Guy Kelly plans to do with his PC public-domain system.

Kim Harris and Michael McNeil have teamed up to tackle multiple exit loop issues in the paper, "Proposed Extensions to Standard Loop Structures." As we work with these gentlemen, we know that tradeoffs are still being weighed and that constructive input is entertained. For example, should each exit have its own, independent termination body? For programming usage, we are ready for these capabilities.

Wil Baden is well known among Orange County FIGgers, but as a Northern California resident, it was our pleasure to make his acquaintance at this year's FORML. His paper, "Modern Control Logic," discusses **THENIF**, a new control structure name which, when used, eliminates the many **THEN**s which from time to time appear at the end of certain colon definitions. In an example, he shows how this also unclutters other cases of **IF** statements. He also implements **LEAVE**, which leaves from **BEGIN** loops, and **EXIT**, which exits from **DO** loops. Wil later showed us his new public-domain system which incorporates these techniques.

Klaus Schleisiek, as usual, would get any awards for having come the farthest distance — West Germany. He has again taken on a major Forth topic, this one being "Error Recovery," for applications. His technique, unlike **ABORT**, **ABORT"** or **QUIT**, allows controlled recovery from any variety of error conditions.

The timeliness of this topic shows in Don Colburn's paper, "User Specified Error Recovery." Don shows how his multi-FORTH can alter the behavior of **ABORT"** but still allow error handlers to be nested by saving data on the return stack. Although clearly a different solution than Klaus', there is a remarkable similarity of ideas in these two papers, and interested readers will want to analyze both.

Chuck Moore might not have much interest in this paragraph, as he runs his entire CAD-CAM in 28K, but those of us who are memory gluttons will want to know about Glen Haydon's overlays titled, "Virtual Vocabularies." Pre-compiled vocabularies are kept on disk and, as needed, are made available and executed in the **V-BUFFER**.

Tom Zimmer's "Improved Block Maintenance" also has a memory problem, but his attention has been focused on mass storage. What do you do if half of your data is spaces? The answer is to compress those spaces. And while doing so, you create a block access structure that also allows convenient insertion of new blank screens in the middle of other blocks, and also allows automatic backup copies of edited screens. Then you write a FORML paper which also points out that this can be implemented from scratch in half a day.

The final paper, "Tail Recursion Convoluted or Return Stack Optimization", by Glenn Tenney is, logically enough, at the tail end of the other papers in this session. Glenn is remembered by conference attendees for his unobtrusive video taping of the conference sessions. About his paper, when the last word in a colon definition is itself a colon definition, the word ;; can be used in place of ;. ;; backs up and converts the docolon call and the exit into a branch which, when used in speed-critical code, will produce a useful time savings without penalty.

Thanks to Mike, Jonathan and Marlin for Frisbee on the beach at sunset. [*Editor's note: It was cold, windy and rainy, but great fun with good friends!*]

Arithmetic and Floating Point
Reviewed by Sidney Bowhill

Five papers were presented in this session. In the first, Charles Springer investigated the use of rational arithmetic, in which fractions are represented by relatively prime integers in the numerator and denominator. This has the advantage that the answers to mathematical calculations are given exactly, without approximation. However, he found that in summing a simple series, the numerator and denominator soon overflow, typically after ten or eleven terms.

Jonathan Sand described the exception handling in the IEEE Floating-Point Standard. The exception requirements are for overflow, underflow, inexact, division by zero and invalid operation. He has implemented the traps for these exceptions as Forth variables, and he described formats to invoke a trap, assign a name, inable, disable and test for enable/disable. As part of the trap handler, the floating-point accumulators and the address of the current operation are saved as variables. All enabled traps are invoked during the final rounding procedure. He also described the use of these traps in the interactive mode during the debugging phase.

Sidney Bowhill presented two papers. The first gave a set of screens that can be compiled to give a floating-point system for the MOS Technology 6502 at any precision in the mantissa from two to fifteen bytes. The machine-language code is based on a 239-byte segment published by Steve Wozniak in 1976. The Forth implementation uses a fifteen-deep floating-point stack and two floating-point accumulators, all in the dictionary. Execution speed of the system at low precision is comparable to or better than existing commercial packages. Multiplication of two floating-point numbers with eight-byte mantissa precision takes sixteen msec.

In a second paper, Bowhill discussed the problems of implementing transcendental functions in a variable-precision floating-point Forth system. The conventional approach using polynomial approximations is impractical, since a different set of coefficients would be required for each precision. He found that power series were adequate for exponential, sine and cosine, since the denominators of the coefficients are factorial in form; for the arctangent function, a continued-fraction approach was found satisfactory. For the logarithm, it was first necessary to produce an argument in the range one to 1.414, following which a modified power series could be used.

Klaus Schleisiek presented a paper with Uwe Lange in which the logarithm of a number could be extracted one bit at a time. This algorithm seems worth exploring as being possibly faster than the power-series approach.

A panel discussion covered, among others, the topics of whether a floating-point system should be regarded as an indispensable part of a Forth package (the predominant view was affirmative); whether floating-point numbers should be placed on the parameter stack, or their addresses placed on the parameter stack, or they should be part of a separate floating-point stack (the predominant view was the latter); and the usefulness of hardware floating-point chips such as the AMD 9511 and the Intel 8087.

File Systems
Reviewed by Ray Talbot

The session on file systems began with a presentation by P. Morton on his data security and file management system. This is a package written in high-level Forth (MVP dialect). The FMS is a general, fairly standard file system, which is designed to function within a Forth block environment. The main interest of those present lay in the data security aspect, as that is not a standard feature of most file systems, much less Forth. The FMS has automatic procedures for maintaining integrity of data files, even through events such as power failures.

W.M. Bradley discussed his proposals for a portable file system interface. This is a set of Forth words which may be used to interface with host operating systems (*e.g.* CP/M, UNIX, MS-DOS, *etc.*) or as a framework for a pure Forth system. The proposed system is patterned after the C I/O interface to UNIX. In addition to the pros and cons of a Forth file system and the justification for the proposed word set, he presented two full source code implementations — one for CP/M-80 and one for 68000 UNIX. Both are largely high-level and, hence, transportable.

P. Midnight discussed a generalization of the CP/M file system interface employed in the Laxen and Perry F83 implementation. The scheme employs file control blocks which permit more flexible access to multiple host files and/or internal Forth files.

There was general discussion on the usual range of file-related topics, the basic question being, "1024-byte, fixed-length blocks: are they good or bad?" The answer was, "Yes." There was some sentiment that source code text not be restricted to fixed-length blocks, but that fixed blocks are good for data files.

In general, file systems in Forth seem to be a gradually accepted necessity for viability in the marketplace.

TAKE FORTH TO WORK

NOW YOU CAN RUN FORTH ON THE OFFICE IBM MAINFRAME. GIVE YOURSELF THE FORTH ADVANTAGE ON THE JOB!

FORTH/370

for large IBM and equivalent computers

- IBM 370, 4341, 3033, etc.
- Based on fig FORTH
- Program compatible with micro FORTH systems
- Editor and Assembler
- Runs under VM/CMS or MVS/TSO
- 32 bit word, 64 bit double word and floating point
- Files compatible with host operating system

FORTH/370 IS AVAILABLE ON A 30 DAY FREE TRIAL. ONLY *ONE* LICENSE FEE OF $1350. REQUIRED FOR *ALL* OF A FIRM'S CPUs.

Source code may be purchased.

WARD SYSTEMS GROUP
8013 Meadowview Drive
Frederick, Maryland 21701
(301) 695-8750

THE FORTH SOURCE™

MVP-FORTH

Stable - Transportable - Public Domain - Tools

You need two primary features in a software development package . . . a stable operating system and the ability to move programs easily and quickly to a variety of computers. MVP-FORTH gives you both these features and many extras. This public domain product includes an editor, FORTH assembler, tools, utilities and the vocabulary for the best selling book "Starting FORTH". The Programmer's Kit provides a complete FORTH for a number of computers. Other MVP-FORTH products will simplify the development of your applications.

MVP Books - A Series

- ☐ **Volume 1,** *All about FORTH* by Haydon. MVP-FORTH glossary with cross references to fig-FORTH, *Starting FORTH* and FORTH-79 Standard. 2nd Ed. $25
- ☐ **Volume 2,** *MVP-FORTH Assembly Source Code.* Includes CP/M®, IBM-PC®, and APPLE® listing for kernel $20
- NEW ☐ **Volume 3,** *Floating Point Glossary* by Springer $10
- NEW ☐ **Volume 4,** *Expert System* with source code by Park $25
- NEW ☐ **Volume 5,** *File Management System* with interrupt security by Moreton $25

MVP-FORTH Software - A Transportable FORTH

- ☐ **MVP-FORTH Programmer's Kit** including disk, documentation, Volumes 1 & 2 of MVP-FORTH Series (*All About FORTH*, 2nd Ed. & Assembly Source Code), and *Starting FORTH*. Specify ☐ CP/M, ☐ CP/M 86, ☐ CP/M+, ☐ APPLE, ☐ IBM PC, ☐ MS-DOS, ☐ Osborne, ☐ Kaypro, ☐ H89/Z89, NEW ☐ Z100, ☐ TI-PC, ☐ MicroDecisions, ☐ Northstar, ☐ Compupro, ☐ Cromenco, ☐ DEC Rainbow $150
- ☐ **MVP-FORTH Cross Compiler** for CP/M Programmer's Kit. Generates headerless code for ROM or target CPU $300
- ☐ **MVP-FORTH Meta Compiler** for CP/M Programmer's kit. Use for applicatons on CP/M based computer. Includes public domain source $150
- ☐ **MVP-FORTH Fast Floating Point** Includes 9511 math chip on board with disks, documentation and enhanced virtual MVP-FORTH for Apple II, II+, and IIe. $450
- ☐ **MVP-FORTH Programming Aids** for CP/M, IBM or APPLE Programmer's Kit. Extremely useful tool for decompiling, callfinding, and translating. $150
- ☐ **MVP-FORTH** by ECS for IBM-PC or ATARI®. Standalone with screen editor. License required. $100
- ☐ **MVP-FORTH** by ECS for IBM-PC or ATARI. With color animation, multitasking sound, utilities, and license. $175
- ☐ **MVP-FORTH Professional Application Development System (PADS)** for IBM-PC, or APPLE. A three level integrated development system with complete documentation. $500
- NEW ☐ **MVP-FORTH Expert System** for development of knowledge-based programs for Apple, IBM, or CP/M. $80
- NEW ☐ **MVP-FORTH File Management System (FMS)** with interrupt security for IBM, Victor 9000, or CP/M $200

FORTH DISKS

FORTH with editor, assembler, and manual.

☐ **APPLE** by MM	$100	☐ **Z80** by LM	$100
☐ **APPLE** by Kuntze	$90	☐ **8086/88** by LM	$100
☐ **ATARI®** valFORTH	$60	☐ **68000** by LM	$250
☐ **CP/M®** by MM	$100	☐ **VIC FORTH** by HES, VIC20 cartridge	$50
☐ **HP-85** by Lange	$90	☐ **C64** by HES Commodore 64 cartridge	$60
NEW ☐ **HP-75** by Cassady	$150	NEW ☐ **Timex** by HW	$25
☐ **IBM-PC®** by LM	$100		
☐ **NOVA** by CCI 8" DS/DD	$175		

Enhanced FORTH with: F-Floating Point, G-Graphics, T-Tutorial, S-Stand Alone, M-Math Chip Support, MT-Multi-Tasking, X-Other Extras, 79-FORTH-79, 83-FORTH-83.

☐ **APPLE** by MM, F, G, & 83	$160	☐ **Extensions** for LM Specify IBM, Z80, or 8086	
☐ **ATARI** by PNS, F,G, & X.	$90	☐ Software Floating Point	$100
☐ **CP/M** by MM, F & 83	$160	☐ 8087 Support (IBM-PC or 8086)	$100
☐ **Apple, GraFORTH** by I	$75	☐ 9511 Support (Z80 or 8086)	$100
☐ **Multi-Tasking FORTH** by SL, CP/M, X & 79	$395	☐ Color Graphics (IBM-PC)	$100
☐ **TRS-80/I or III** by MMS F, X, & 79	$130	☐ Data Base Management	$200
☐ **Timex** by FD, tape G,X, & 79	$45	Requires LM FORTH disk.	
☐ **Victor 9000** by DE,G,X	$150		

☐ **fig-FORTH Programming Aids** for decompiling, callfinding, and translating. CP/M, IBM-PC, Z80, or Apple $150

CROSS COMPILERS Allow extending, modifying and compiling for speed and memory savings, can also produce ROMable code. •Requires FORTH disk.

☐ CP/M	$300	☐ IBM•	$300
☐ 8086•	$300	☐ Z80•	$300
☐ 68000•	$300	☐ Apple II/II+	$300

FORTH COMPUTER

☐ **Jupiter Ace**	$150
☐ 16K RAM Pack	$50
☐ 48K RAM Pack	$125

Key to vendors:

CCI Capstone Computing Inc.	LM Laboratory Microsystems
DE Dai-E Systems	MM MicroMotion
FD Forth Dimension	MMS Miller Microcomputer Services
HW Hawg Wild Software	PNS Pink Noise Studio
I Insoft	SL Shaw Labs

FORTH MANUALS, GUIDES & DOCUMENTS

☐ **ALL ABOUT FORTH** by Haydon. See above.	$25	☐ **1980 FORML Proc.**	$25
☐ **FORTH Encyclopedia** by Derick & Baker	$25	☐ **1981 FORML Proc** 2 Vol	$40
NEW ☐ **The Complete FORTH** by Winfield	$16	☐ **1982 FORML Proc.**	$25
☐ **Understanding FORTH** by Reymann	$3	☐ **1981 Rochester FORTH Proc.**	$25
NEW ☐ **FORTH Fundamentals,** Vol. I by McCabe	$16	☐ **1982 Rochester FORTH Proc.**	$25
NEW ☐ **FORTH Fundamentals,** Vol. II by McCabe	$13	NEW ☐ **1983 Rochester FORTH Proc.**	$25
NEW ☐ **Beginning FORTH** by Chirlian	$17	☐ **A FORTH Primer**	$25
☐ **FORTH Encyclopedia Pocket Guide**	$7	☐ **Threaded Interpretive Languages**	$23
☐ **And So FORTH** by Huang. A college level text.	$25	☐ **METAFORTH** by Cassady	$30
☐ **FORTH Programming** by Scanlon	$17	☐ **Systems Guide to fig-FORTH**	$25
☐ **FORTH on the ATARI** by E. Floegel	$8	☐ **Invitation to FORTH**	$20
☐ **Starting FORTH** by Brodie. Best instructional manual available. (soft cover)	$18	☐ **PDP-11 User Man.**	$20
☐ **Starting FORTH** (hard cover)	$23	NEW ☐ **FORTH-83 Standard**	$15
NEW ☐ **68000** fig-Forth with assembler	$25	☐ **FORTH-79 Standard**	$15
		☐ **FORTH-79 Standard Conversion**	$10
		☐ **Tiny Pascal fig-FORTH**	$10
		☐ **NOVA fig-FORTH** by CCI Source Listing	$25
		☐ **NOVA** by CCI User's Manual	$25
		NEW ☐ **Jupiter** ACE Manual by Vickers	$15

☐ **Installation Manual for fig-FORTH,** $15

Source Listings of fig-FORTH, for specific CPU's and computers. The Installation Manual is required for implementation. Each $15

☐ 1802	☐ 6502	☐ 6800	☐ AlphaMicro
☐ 8080	☐ 8086/88	☐ 9900	☐ APPLE II
☐ PACE	☐ 6809	☐ NOVA	☐ PDP-11/LSI-11
☐ 68000	☐ Eclipse	☐ VAX	☐ Z80

Ordering Information: Check, Money Order (payable to MOUNTAIN VIEW PRESS, INC.), VISA, MasterCard, American Express. COD's $5 extra. Minimum order $15. No billing or unpaid PO's. California residents add sales tax. Shipping costs in US included in price. Foreign orders, pay in US funds on US bank, include for handling and shipping by Air: $5 for each item under $25, $10 for each item between $25 and $99 and $20 for each item over $100. All prices and products subject to change or withdrawal without notice. Single system and/or single user license agreement required on some products.

MOUNTAIN VIEW PRESS, INC.

PO BOX 4656 MOUNTAIN VIEW, CA 94040 (415) 961-4103

Techniques Tutorial

Multi-Tasking, Part II

Henry Laxen
Berkeley, California

```
: LOCAL    (S base addr -- addr' )
  UP @ -    +   ;
: SLEEP    (S addr -- )
  0 ( NOP ) SWAP ENTRY LOCAL C!    ;
: WAKE     (S addr -- )
  207 ( RST ) SWAP ENTRY LOCAL C!    ;
: STOP     (S -- )
  UP @ SLEEP   PAUSE    ;
```

Figure One

```
 1 : TASK:    (S size -- )
 2    CREATE   TOS HERE #USER @ CMOVE    ( Copy the User Area )
 3    HERE ENTRY LOCAL LINK !    ( I point to him )
 4    ENTRY   UP @ -ROT   HERE UP !  LINK !    ( He points to me )
 5    DUP HERE +   DUP RP0 !   100 - SP0 !   SWAP UP !
 6    ( Reserve space for return stack )
 7    HERE #USER @ +   HERE DP LOCAL !
 8    HERE SLEEP   ALLOT   ;
 9
10 : SET-TASK   (S ip task -- )
11    DUP SP0 LOCAL @    ( top of stack )
12    2- ROT OVER !   ( Initial IP )
13    2- OVER RP0 LOCAL @ OVER !    ( Initial RP )
14    SWAP TOS LOCAL !    ;
15
16 : ACTIVATE   (S task -- )
17    R> OVER SET-TASK   WAKE    ;
```

Figure Two

```
400 TASK: COUNTING

VARIABLE  #TIMES

: COUNTER COUNTING ACTIVATE BEGIN  1 #TIMES +!  PAUSE AGAIN ;

COUNTER
```

Figure Three

Last time, we saw how to implement the low-level portion of a multi-tasker. We learned that, in Forth, tasks must cooperate with each other and give up control of the CPU at various points. We saw how the **PAUSE** and **RESTART** words work and how they very efficiently save the status of a task and restore it. This time, we will take a look at how to create tasks and, once started, how to manage them.

Just for the record, let me restate that tasks are linked together in a circular list via the **LINK** user variable. A task is active if the **ENTRY** user variable contains an RST instruction, and is inactive if it contains a JMP instruction.

The human (I want to say user but will refrain) interface to this mechanism is displayed in figure one. Let's take a look at what each word does and how it works. First, **LOCAL** is a tool that allows one to access a **USER** variable within a specified task. It just computes the actual address of a user variable, given the starting address of the required task. **SLEEP** installs a NOP machine instruction into byte zero of the **ENTRY** user variable. Since byte one contains a JMP instruction, the effect of **SLEEP** is to guarantee that the next task will get control immediately without this task doing anything. Notice that there is only one instruction (a JMP) executed for each inactive task. This is extremely low overhead. The **WAKE** word is the inverse of **SLEEP**. It installs an RST instruction into byte zero of **ENTRY**. This will eventually cause the **RESTART** word to be executed, and awaken this task. Finally, the **STOP** word simply puts the current task to sleep and passes control to the next task. **WAKE** and **SLEEP** both require an argument, which is a pointer to the task that they are to act on, while **STOP** acts on the current task and, hence, requires no argument. The names for these functions are extremely apt and I wish the credit for them was mine; but I am afraid they belong to Charles Moore. Thank you, Chuck.

Now that we know how to start and stop tasks once they exist, let's take a look at what must be done to set up a task in the first place. The code associated with this appears in figure two. The **TASK:** word sets up a task of a specified size. The **SET-TASK** word initializes a task so that it is ready to run and the **ACTIVATE** task allows you to associate a high-level definition with the task. Let's look at each word in more detail.

Tasks are allocated as part of the dictionary. Also, each task must have its own user area, return stack, parameter stack and dictionary space. This setup is handled in **TASK:**, which is a defining word that creates a task with a given name and of a specified size. When the name of the task is executed, it returns a pointer to itself. A simple **CREATE** suffices for this function, since the word it defines returns its parameter field address.

Next, a copy of the current task's **USER** area is copied to the new task. On line two we set up the current task's **LINK** pointer to point to the new task, and on line three we make the new task point to the old entry point of the current task. We also save a pointer to the current task on the stack. On line five we set up the size of the return stack and the empty parameter stack of the new task, and restore the User Pointer to point to the current task. On line six we initialize the new task's dictionary pointer and, finally, on line seven we put the new task to sleep and allocate space for it in the dictionary of the current task.

SET-TASK sets up a task for its first execution. It places the initial values of the IP and the return stack pointer onto the new task's parameter stack, and stuffs the new task's initial parameter stack value into the TOS user variable for the new task. In essence, **SET-TASK** behaves as though the new task has just done a **PAUSE**, and is ready to do a **RESTART**. This is what you would expect. Finally, **ACTIVATE** uses **SET-TASK** to make the new task point to the code following the **ACTIVATE** word, and **WAKE**s up the new task.

Last but not least, let's see how we actually set up another task. Figure three illustrates this. On line one we define a **COUNTING** task and allocate 400 bytes for its use. On line three we simply define a variable called **#TIMES** which will hold the number of times we have counted. Line five defines a word called **COUNTER** which specifies that the **COUNTING** task is to be **ACTIVATE**d by explicit use of the **PAUSE** word. This is absolutely vital, since this task performs no I/O, hence it must explicitly give up control of the CPU at specified moments. To start running the task, simply execute the word **COUNTER**. Now you can watch the behavior of the task by periodically displaying the contents of the variable **#TIMES**. You will be able to see it incrementing very rapidly. If you want to stop the new task from executing, you need only type **COUNTING SLEEP**. Again, you can query the value of **#TIMES** and, indeed, verify that the task has suspended operation. To start it up again, just type **COUNTING WAKE** and you will once again be able to see the variable **#TIMES** incrementing.

This has been an extremely simple example of a background task. Other applications can be far more useful. For example, you can use the multitasker as a mechanism for implementing print spooling and windowing, as well as pipes and filters. I hope these two articles on multi-tasking are a starting point for your own experimentation. Until next time, may the Forth be with you.

Copyright © 1983 by Henry Laxen. All rights reserved. The author is Vice-President of Research and Development for Paradise Systems, Inc., 150 North Hill Drive, Brisbane, CA 94005, manufacturers of the MultiDisplay Card for the IBM-PC and other computer-related products.

Chapter News

John D. Hall
Oakland, California

We have four new chapters!

Central Indiana FIG Chapter
Indianapolis, Indiana

Detroit Atari FIG Chapter
Detroit, Michigan

Cleveland FIG Chapter
Cleveland, Ohio

Iowa City FIG Chapter
Iowa City, Iowa

Cleveland Chapter

November 22: At the first meeting of the chapter, there was an excellent turnout of thirteen local FIG members and three non-FIGgers. The meeting was mostly organizational, but they took time out to introduce each other and to find their common interests. Gary Bergstrom, who did the leg work to get this chapter started, gave a short talk on a code compiler program/article he has been working on. He promises it will be submitted to *Forth Dimensions* soon.

Melbourne Chapter

September 22: At the chapter meeting, some inconclusive discussion on screen transfers was due to the leading lights of the previous meeting not appearing at this one. However, Wesley Summers worked away in the background and got some transfer code working.

October 7: After a change of venue to Paul Fraser's house, Paul showed the group his AussieByte board and Lance Collins talked about his screenless Forth developments.

November 4: The chapter had a joint meeting with the Hitachi Peach Users Group at Templestowe Technical School. Graeme Hedley gave a very good talk on 6809 Forth. He also showed a speech synthesis system done by a graduate student (in Forth, of course) at Latrobe University.

Detroit Atari Chapter

October: The first order of business was to elect officers. Tom Chrapkiewicz was elected manager and Todd Meitzner was elected assistant manager. Copies of the various public-domain Forths were made for those requesting them. An informal discussion of the various versions of Forth available followed. Almost every version of Forth for the Atari computer was represented. The most popular versions seemed to be VALFORTH, APX and the Bay Area public-domain versions. The group elected, for discussion purposes, the FIG-Forth standard, which will be compatible with these Forths.

Todd Meitzner demonstrated a kaleidoscope program and a Rubik's Cube display program. The source code for these two programs was discussed. The chapter's first endeavor for future meetings will be to run a tutorial on Forth using *Starting Forth* by Leo Brodie.

(Continued on next page)

Orange County Chapter

October: The group took up the discussion of Forth-83 with Bob Snook, pointing out the pitfalls of the new truth flag. Further discussion followed about the articles in *Forth Dimensions* and *Dr. Dobb's Journal* by Wil Baden. To get the newer members of the group up to speed in Forth, Roland Koluvek gave a tutorial about the FIG-Forth vocabulary structure presented in *Forth Dimensions*, and reviewed again the **ONLY...ALSO** structure.

Northern California Chapter

November 26: Since this meeting was so soon after the FORML sessions of November 23-25, several FIG members from out of town used the opportunity to visit and share their ideas. It is really very nice to have the new faces, fresh ideas and lively discussions that occur when guests drop in. Wil Baden, from the Orange County Chapter, gave us a talk about an 83-Standard Forth he has for the Apple. It is a public-domain implementation and is available from him. It has many extensions, including an assembler, editor, debugger and the directory structure he presented in *Forth Dimensions*. Bob Berkey presented an idea about how a change to the null word could clean up much of the problem with running out of input while in the compile state.

In the afternoon session, the out-of-towners introduced themselves. They were Jon Rible from Massachusetts, Pierre Morton from France and temporarily in the Bay Area, and Wil Baden from Orange County. (Wil made the comment that he was asked to be an officer of the Orange County group, which he had to turn down; but since he did so much preaching about Forth, he decided to accept the position of "chaplain.") During the Rumors portion of the meeting, Dr. Ting showed us a clock he has for sale. It runs in "reverse-Polish" (backward). A FORML report was presented, and it seems each year FORML gets better and better. There were nearly eighty in attendance, with fifty participants.

After the FORML presentation, Glenn Tenney, moderator, held a "Doctor Is In" session and took written questions from the audience. Some of the questions were: "What is floored division?" "Can Forth handle interrupts?" "What is the difference between **[COMPILE]**, **COMPILE** and]?" And, "Explain the text interpreter and the inner interpreter." Simple questions, right Glenn? He handled them so well that he had to call for help only once. Dr. Ting stepped in for the last question, and when he was through, we all had a new and much clearer insight into the world of Forth interpreters. Dr. Ting has promised to submit a paper to *Forth Dimensions* on this topic. Look for it in an upcoming issue!

Chapters in Formation

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

Contact:

Michael Perry
1446 Stannage Ave.
Berkeley, CA 94702

Charles Shattuck
206 Irene Ave.
Roseville, CA 95678

Ron C. Estler
Ass't. Prof. of Chemistry
Ft. Lewis College
Durango, CO 81301

Thomas Hand
617 Manor Place
Melbourne, FL 32907

John Forsberg
17740 SW 109th Place
Perrine, FL 33157
305/252-0108

Harvey Glass
College of Engineering
Univ. of South Florida
Tampa, FL 33620

Ron Skelton
1220 Winding Branch Circle
Atlanta, GA 30338
404/393-8764

Tony Sanger
1931 Sam's Creek Rd.
Westminster, MD 21157
301/875-2915

Gary Zajsc
100 South Zajac Dr.
Altoona, PA 16602

Richard C. Secrist
Facility & Mfctr. Automation, Inc
117 Flint Rd.
Oak Ridge, TN 37830

Matt Lawrence
8409 Jamestown
Austin, TX 78758
512/834-8455

John London, Jr.
211 E. Grace St.
Richmond, VA 23219
804/233-7237

Thomas C. Kuffel
18221 - 29th Place NE
Seattle, WA 98155

Arnold Pinchuk
2130 Menasha Ave.
Manitowoc, WI 54220

Jean-Daniel Dodin
77 Rue du Cagire
31100 Toulouse
France
16-61/44.03.06

Klaus Schleisiek
P.O. Box 202264
2000 Hamburg 20
West Germany
04103-13255

Lam, Kwak Yin F
Mass Transit Railway Corp.
GPO Box 9916
Hong Kong

Hugh Dobbs
Computer Studies Dept.
Newton School
Waterford, Ireland

Joan Verdaguer
Apdo. de Correos 24257
Barcelona
Spain

Dr. Robert Johnson
Institute of Physical Chemistry
Box 532, Uppsala Univ.
Uppsala, Sweden

FORTH INTEREST GROUP

MAIL ORDER

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

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

☐ "Starting FORTH, by Brodie. BEST book on FORTH. (Paperback)		$18	$22
☐ "Starting FORTH" by Brodie. (Hard Cover)		$23	$28
☐ PROCEEDINGS: FORML (FORTH Modification Conference)			
☐ 1980, $25USA/$35Foreign			
☐ 1981, Two Vol., $40USA/$55Foreign			
☐ 1982, $25USA/$35Foreign			
ROCHESTER FORTH Conference			
☐ 1981, $25USA/$35Foreign			
☐ 1982, $25USA/$35Foreign			
☐ 1983, $25USA/$35Foreign	Total	$______	
☐ STANDARD: ☐ FORTH-79, ☐ FORTH-83. $15USA/$18Foreign EACH.	Total	$______	
☐ Kitt Peak Primer, by Stevens. An in-depth self-study book.		$25	$35
☐ MAGAZINES ABOUT FORTH: ☐ BYTE Reprints 8/80-4/81			
☐ Dr Dobb's Jrnl, ☐ 9/81, ☐ 9/82, ☐ 9/83			
☐ Poplar Computing, 9/83 $3.50USA/$5Foreign EACH.	Total	$______	
☐ FIG T-shirts: ☐ Small ☐ Medium ☐ Large ☐ X-Large		$10	$12
☐ Poster, BYTE Cover 8/80, 16"x22"		$ 3	$ 5
☐ FORTH Programmer's Reference Card. If ordered separately, send a stamped, self addressed envelope.		Free	
	TOTAL	$______	

NAME______________________ MS/APT__________

ORGANIZATION______________________ PHONE(   )__________

ADDRESS______________________________

CITY______________ STATE_____ ZIP__________ COUNTRY__________

VISA#______________ MASTERCARD#______________

AMERICAN EXPRESS#______________ Card Expiration Date__________

(Minimum of $15.00 on Charge Cards)

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

ORDER PHONE NUMBER: (415) 962-8653

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

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

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

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

Address Correction Requested