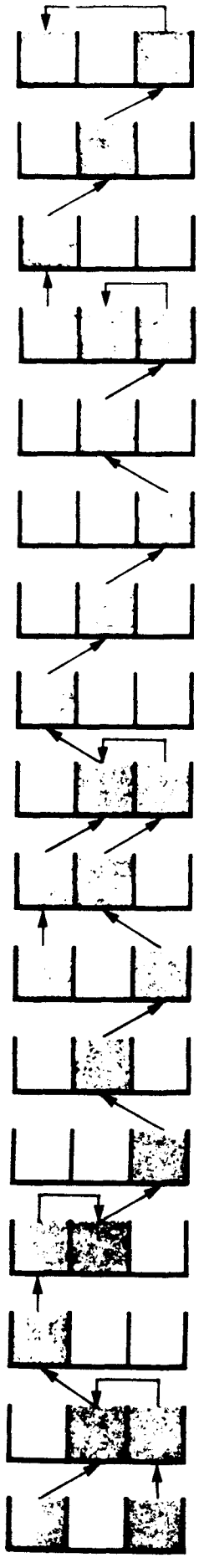

FORTH DIMENSIONS

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

Volume II
Number 1
Price $2.00

INSIDE

FORTH DIMENSIONS

Published by Forth Interest Group

Volume II No. 1 May/June 1980

Publisher Roy C. Martens

Subscription to FORTH DIMENSIONS is free with membership in the Forth Interest Group at $12.00 per year ($15.00 overseas). For membership, change of address and/or to submit material, the address is:

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

HISTORICAL PERSPECTIVE

FORTH was created by Mr. Charles H. Moore in 1969 at the National Radio Astronomy Observatory, Charlottesville, VA. It was created out of dissatisfaction with available programming tools, especially for observatory automation.

Mr. Moore and several associates formed FORTH, Inc. in 1973 for the purpose of licensing and support of the FORTH Operating System and Programming Language, and to supply application programming to meet customers unique requirements.

The Forth Interest Group is centered in Northern California, although our membership of 1100 is world-wide. It was formed in 1978 by FORTH programmers to encourage use of the language by the interchange of ideas through seminars and publications.

IMPORTANT DATES

April 26 — FIG Monthly Meeting, 1:00 pm, at Liberty House, Hayward, CA. Come to the FORML Workshop at 10:00 am and stay on.

May 20 — National FIG Meeting, Disneyland Hotel, Anaheim, CA at the NCC Personal Computing Festival. Dinner in the evening and technical sessions all day. Contact: Jim Flournoy, (408) 779-0848.

May 24 — FIG Monthly Meeting, 1:00 pm, at Liberty House, Hayward, CA. Come to the FORML Workshop at 10:00 am and stay on.

June 8-13 — American Chemical Society

June 21 — So. Cal. FIG Meeting, MSI Data Corp., 300 Fischer Ave., Costa Mesa, CA. Noon.

June 28 — FIG Monthly Meeting, 1:00 pm, at Liberty House, Hayward, CA. Come to the FORML Workshop at 10:00 am and stay on.

PUBLISHER'S COLUMN

Don't let your membership in FIG crash. Renew today! It's easy. Just send in your check for $12.00 ($15.00 overseas) and you'll be all set for the next six issues of FORTH DIMENSIONS and the FIG notices. If you are in doubt as to whether your membership is up, just look at the address label. If it reads "Renew March 1980" then its time to get that check off. Do it today.

The next issue of FORTH DIMENSIONS is going to be super. It will be a technical issue with all the entries submitted in the Case Contest. Make sure that you receive this important issue, renew your membership in FIG today.

This may sound like a hard pitch for your membership but FIG needs you. The only way that we can keep on publishing FORTH DIMENSIONS and spreading the FORTH word is by having your support. In fact, how about getting others to sign up.

Roy Martens

KIM HARRIS COURSE

A five day intensive course on programming with FORTH will be held July 21-25 at Humbolt State University in Arcata, California. The course will cover the FORTH approach to producing computer applications including: (1) analyzing the requirements of a problem, (2) designing a logical solution, and (3) implementing and testing the solution. Topics will include the usage, extension, and internals of the FORTH language, compiler, assembler, virtual machine, multitasking operating system, mass storage virtual memory manager, and file system. Computers will be available for demonstrations and class exercises. The course will be taught by Kim Harris, and Humbolt State University will give 4 units of credit through the office of Continuing Education. Tuition for the course is $112 per student. The text will be "Using FORTH"; copies will be available at the course for $25 each. Housing is available in very nice dormitory rooms for $9 per person per night or at several nearby motels. Cafeteria meals may be purchased individually or at $10.25 per day. For more information and registration materials write, before June 23:

Prof. Ronald Zamnit
Physics Department
Humbolt State University
Arcata, California 95521

FORML NEWS

FORML (FORTH Modification Laboratory) is a research group coordinating individual efforts on the technical evolution of FORTH. Workshop meetings are held the fourth Saturday of each month at 10:00 a.m. at the Liberty House, Hayward, CA. (Make a day of it by staying for the FIG meeting in the afternoon.) Working groups determine and document: the objectives (what problems need to be solved), status of topic (what has already been done), the challenges (what has to be done), the methods (the appropriate approach), the list (detailed topics and problems), the specifications (requirements of valid solutions). You can input directly to the technical committees or to FIG Chairman Kim Harris (see Files DBMS). The committees and leaders are:

Committee	Leader
Numeric Extensions & Floating Point	LaFarr Stuart P.O. Box 1418 Sunnyvale, CA 94088 (408) 296-6236
MetaFORTH - Nucleus	Armand Gambera TTI Inc. 555 Del Rey Ave. Sunnyvale, CA 94086 (408) 735-8080
Concurrency, Multitasking, Executive Communication Synchronization	Terry Holmes 808 Coleman, #21 Menlo Park, CA 94025
Strings	John Cassady 11 Miramonte Road Orinda, CA 94563 (415) 254-2398
Documentation	John S. James P. O. Box 348 Berkeley, CA 94701 (415) 526-8815
Graphics	Howard Pearlmutter 1055 Oregon Ave. Palo Alto, CA 94303 (415) 856-0450
Files DBMS	Kim Harris 1055 Oregon Ave. Palo Alto, CA 94303 (415) 856-0450

FORML needs your help. Come to the next meeting!

THIS IS THE BEGINNING!

THE BEGINNING OF FIG TWO!

THE BEGINNING OF FORTH DIMENSIONS II!

IT'S TIME TO RENEW!

RENEW YOUR MEMBERSHIP IN FIG!

RENEW YOUR SUBSCRIPTION TO FORTH DIMENSIONS!

DO IT ALL FOR ONLY $12.00!

DO IT TODAY!

IT'S EASY!

CHECK THE LABEL FOR RENEW DATE!

IF IT READS "Renew Mar. 1980" SEND A CHECK!

SEND IT TO: FIG, P.O. Box 1105, San Carlos, CA 94070 RENEW NOW!

FORTH, for the Motorola 6809

Raymond J. Talbot, Jr.
7209 Stella Link, Suite 112
Houston, Texas 77025

68'FORTH is an implementation of fig-FORTH for the 6809 microprocessor. It is available on 5" disk configured for an SWTPC SS-50 Buss system with SWTPC MF-68 dual 5" disks and the TSC FLEX 9.0 disk operating system, but it is easily modifiable for other systems (write author for information).

The 6809 is a greatly improved version of the Motorola 6800 8-bit microprocessor. It is almost like having a 16-bit microprocessor, since there are several 16-bit instructions. It has two 16-bit index registers X and Y, and a 16-bit accumulator register D which may also be used as two 8-bit registers A and B. There are many addressing modes, including indirect, autodecrement, and autoincrement.

The two hardware stack registers make it ideal for FORTH — it is almost a FORTH machine in silicon. I have implemented FORTH by the following register assignments:

The FORTH variable stack — U stack register

The FORTH return stack — S stack register

The FORTH instruction pointer (IP) — Y index register

The FORTH register W (which points to the machine code being executed) is never stored (to save an instruction which is usually unnecessary), however, upon entry to a word's machine code, that address is in the -- X index register

Inside a word, one may use X and D without bothering to save their values on entry. If one wants a second index register (very handy for something like CMOVE), then one or more of Y, S, or U registers may be saved in memory (or on one of the stacks).

Before listing the code which makes the FORTH machine, let me describe the notation used to make dictionary entries with the TSC assembler MACRO facility:

```
LASTNM  SET     0               initialize last name address to be
                                  zero; this will mark beginning
                                  of dictionary
WORDM   MACRO                   macro called WORDM to make entry
NEXTNM  SET     *               sets NEXTNM equal to present location
                                  which will be first byte of name
        IFC     &4, IMMEDIATE   conditional compilation for IMMEDIATE
                                  words
        FCB     &1+$C0          first byte is no. of char. with sign
                                  and immed. bit
        ELSE
        FCB     &1+$80          no immed. bit
        ENDIF
        IFNC    &1,1            special case of a 1 character word
                                  will skip this
        FCC     /&2/
        ENDIF
        FCB     $80+'&3         last character has sign bit set
        FDB     LASTNM          link to previous word in dict
LASTNM  SET     NEXTNM          reset LASTNM to point to this word
        ENDM                    end of macro

A-10
```

The &n quantities refer to parameter to the MACRO. E.g.,

```
MACRO  4,BAS,E
```

will assemble as

```
84  42  41  53  C5        LI  NK
```

Where LI NK is the link address to the previous entry. This macro coupled with assembly of addresses allows one to write assembly language code that is essentially just colon definitions, e.g., the macro definition of COLON itself below.

Here is the assembly language listing for the portion of the code which defines the 6809 FORTH machine:

```
        WORDM   1,:,IMMEDIATE
COLON   FDB     DOCOL, QEXEC, SCSP, CURRENT, AT, CONTXT, STORE
        FDB     CREATE, RBRAK, PSCODE
DOCOL   PSHS    Y               push IP = Y to ret stack = S
        LEAY    2,X             increment Y to first parameter after
                                  CFA in W = X
NEXT    LDX     ,Y++            get W into X and then increment IP = Y
                                  to point to next instruction
NEXTP   JMP     [,X]            jump indirect to code pointed to by
                                  W = X
        WORDM   2,;,S
SEMIS   FDB     *+2
PSEMIS  LDY     ,S++            reset IP = Y to next address (found
                                  by popping from the ret stack = S).

A-11
```

Some arbitrarily chosen examples of the great economy achieved by this use of the stack registers is given by some words shown here (note: depending on location, some of the BRA have to be the long branch instruction LBRA).

```
            WORDM   ?, EXECUT,E
EXEC        FDB     *+2
            PULU    X               pull address from var. stack = U
                                      and put into W = X -- one of
                                      very few cases requiring W
            BRA     NEXTP
            WORDM   1,.+
PLUS        FDB     *+2
            PULU    D               get top item from stack into D
                                      accumulator, add second item,
                                      new top of stack
            ADDD    ,U
PUTD        STD     ,U              store sum back on stack
            BRA     NEXT
            WORDM   2,1,+
ONEP        FDB     *+2
            LDD     ,U              get top item into D
            ADDD    #1              add 1
            BRA     PUTD            put back onto D
            WORDM   1,.@
AT          FDB     *+2
            LDD     [,U]            get @ pointed to by add. on stack
            BRA     PUTD            replace top of stack with @
            WORDM   1,.!
STORE       FDB     *+2
            PULU    X,D             gets top into D, second into X
            EXG     X,D             exchange
            STD     ,X              store second into location pointed
                                      to by top
            BRA     NEXT
            WORDM   2,>,R
TOR         FDB     *+2
            PULU    D               pull top item from var. stack
            PSHS    D               push onto ret. sack
            BRA     NEXT
```


These various fragments of 6809 code can be compared with the corresponding 6800 code in the FIG 6800 ASSEMBLY SOURCE LISTING. Specifically, the 6809 NEXT routine takes 14 machine cycles whereas the 6800 NEXT routine takes 38 cycles.

The 68'FORTH implementation for the 6809 is essentially identical with the 6800 fig version except for the machine code for the words defined that way. Many words which are coded (like PLUS) are shorter in 6809 code because of the 16-bit math. For all of the colon-definition-like-words in 6800 fig-FORTH, I just used my WORDM MACRO; that keeps the source file short.

68'FORTH implements the full fig-FORTH vocabulary as given by the May 1979 6800 ASSEMBLY SOURCE LISTING and the fig-FORTH Installation manual. In addition, particular installation dependent code for EMIT, KEY, and disk read and write are given for a 6809 system with all disk I/O being done via the disk sector read/write routines of the TSC FLEX 9.0 disk operating system. FLEX formatted text files may be read or written in lieu of the terminal. (The word READ switches KEY to read a text file, similarly WRITE switches EMIT to write a text file). Consequently, it is possible to communicate data between FORTH and other FLEX programs (Horrors - BASIC even!!).

Another feature of 68' FORTH is something which should be part of any FORTH system which operates under a host DOS — It has a word _ (underscore on some terminals, left arrow on others) which is followed by a text string (delimited by carriage return or double quote). The text string is passed to the DOS command processor. While in 68'FORTH one can do any FLEX command, e.g., CAT (catalog of FLEX files), DELETE, RENAME, etc., by, e.g.

```
_ CAT 1.F (carriage return)
```

to get a catalog of all files on drive 1 with name starting with F. This type of facility is extremely useful for the exchange of data with other types of programs and for using FORTH in time-sharing systems where other people use other languages. For example, at Rice we operate a PDP-11/55 with the UNIX operating system and FORTH functions as just one time-shared process along with many others. As yet our PDP-11 FORTH does not have this _ word, but I plan to include it in order to take advantage of the many extremely useful utility programs which exist in UNIX. In particular, in that environment, we want to be able to transfer data between tapes and disks as background jobs while we are working with files interactively with FORTH. Also, for number crunching work, other languages are more convenient and faster than FORTH, so we plan to implement certain tasks in other languages to be done as background jobs supervised by UNIX while we use FORTH for just those interactive tasks for which it is ideal.

My main reason for pointing out these FORTH connections to another DOS

is to encourage the FORTH standards team to think about standard vocabulary words for these links. FORTH grew up and still largely exists as a stand alone operating system. However, already it is used in some places as simply one language in a multi-language time shared system. I know of two places -- here at Rice where we have begun a rudimentary connection between FORTH and UNIX, and at Kitt Peak National Observatory where their CDC 6400 has very elaborate inconnections between FORTH and CDC's SCOPE.

Editors Note -- This is an excellent example of conversion of a FIG assembly listing to another processor. However one more change is in order. NEXT on the 6809 is only 4 bytes long and the code jump to NEXT takes 3 bytes. On processors this powerful, the code for NEXT is repeated, in-line, wherever needed. This costs one byte but saves 3 clock cycles on each interpretive cycle. The time overhead of indirect threaded code is then 12 cycles. Similarly, PUTD should be expanded in line.

Recursion —

The Eight Queens Problem

Jerry LeVan
Dept. of Math Science
Eastern Kentucky University
Richmond, KY 40475

The Eight Queens problem has been often used as a textbook problem in programming, particularly to illustrate recusion. I present here a solution in FORTH.

Recusion is the technique of allowing a procedure (a FORTH word definition) to call itself. This is normally blocked during FORTH compilation, to allow a old word name to be used in a new definition of the same name. For example:

```
: HELP  CR  CR  HELP  CR  CR  ;
```

The new definition of HELP will execute a previous definition, but with two carriage returns before and after. This is a necessary and common capability.

How then to have a word call itself, if not by name? The answer is MYSELF. This word will compile a call to the word in which it is located:

```
:  DEMO

   - - - -

   IF  MYSELF  ELSE  ---  THEN  ;
```

In this example, if the test is true at IF, at MYSELF a call to DEMO will occur. This is accomplished by defining MYSELF as IMMEDIATE. At compile time, MYSELF executes and compiles the execution (code field) address of the most recent (actually incomplete) definition in the CURRENT vocabulary. The fig-FORTH definition is:

```
:  MYSELF
LATEST  PFA  CFA  ,  ;  IMMEDIATE
```

The Four Queen Problem at hand finds the board row and column locations on which eight chess queens would be safe from mutual attack. This example doesn't check for board rotations or reflections, so more answers are printed than necessary.

The output gives the calculation number on which the answer was found and a list of the eight row numbers, column by column on which the queens are located. Now it's your turn to DO-IT.

```
SCR # 57
  0 (  8 queens   by Jerry LeVan                          WFR-79DEC02 )
  1 : 2*   DUP  +  ;  ( double a value )
  2
  3 : MYSELF    ( allow a word to call itself, by recursion )
  4       LATEST  PFA  CFA  ,  ;  IMMEDIATE
  5
  6 : IARRAY   ( makes an array of 1's, as given by input )
  7     <BUILDS  0  DO  1  ,  LOOP
  8     DOES>  SWAP  2*  +  ;  ( leave address within array )
  9
 10  8 IARRAY A   ( these form workspace for the solutions )
 11 16 IARRAY B
 12 16 IARRAY C
 13  8 IARRAY X   ( this contains trial solutions )
 14 -->
 15
```

```
SCR # 58
  0 (  more 8 queens                                      WFR-79DEC02 )
  1 : SAFE
  2     SWAP  OVER  OVER  OVER  OVER
  3     -  7  +  C  @  >R
  4     +  B  @  >R
  5   DROP  A  @  R>  R>  *  *  ;
  6 : MARK
  7     SWAP  OVER  OVER  OVER  OVER
  8     -  7  +  C    0  SWAP  !
  9     +  B  0  SWAP  !
 10     DROP  A  0  SWAP  !  ;
 11 : UNMARK
 12     SWAP  OVER  OVER  OVER  OVER
 13     -  7  +  C  1  SWAP  !
 14     +  B  1  SWAP  !
 15     DROP  A  1  SWAP  !  ;  -->
```

```
SCR # 59
  0 (  more 8 queens                                      WFR-79DEC02 )
  1 0 VARIABLE TRIES
  2 : PRINTSOL   ( print one solution )
  3     ." found on try "  TRIES  @  6 .R  8  0
  4   DO  I  X  @  1+  5  .R  LOOP  CR   ;
  5 : TRY        8  0       ( search for answers )
  6   DO 1 TRIES +!  ?TERMINAL  IF  QUIT  THEN  DUP  I  SAFE
  7     IF  DUP  I  MARK
  8         DUP  I  SWAP  X  !
  9         DUP  7  <
 10           IF  DUP  1+  ?STACK  MYSELF  ELSE  PRINTSOL  THEN
 11         DUP  I  UNMARK
 12      THEN
 13    LOOP  DROP  ;
 14
 15 : DO-IT    0 TRIES !  0  CR  TRY  ;   ( This runs the problem )
```

A 'TINY' Pseudo-Code

Bill Powell
Sawbridgeworth, Herts
England

There are some interesting speed/ memory trade-offs which depend on the pseudo-code adopted in implementing FORTH. The discussion by David Sirag [1] for the PDP-11 shows DTC to be both faster and more compact, but less flexible (?) than ITC which is the de facto standard. But 6502 FORTH (Programma Consultants) appears to use the JSR/RTS structure. This is faster, but must lead to a lot of code since it now takes 3 instead of 2 bytes to reference each low level (CODE) routine in a high level (COLON) definition.

On my 6502, an 8 bit machine, it seems attractive to call the most frequently used FORTH words with a single byte. My 'TINY' code reads the first byte and then shifts it left to write bits 6 and 7 into the sign and carry flags. For codes $80 thru $FF (carry set) a branch is taken to a 2 byte COLON instruction. The even value we now have is used for the Lo byte for the Instruction Counter (IC). The Hi byte is read by the original IC before being saved on the return stack. This is still a two byte p-code which allows us a vast number of Colon definitions. But we no longer need the Code Address, saving 2 bytes. But we must start these entries at even addresses which will cost 1/2 byte on average.

Next the sign bit is tested. If clear, a branch is taken to a routine for low Literals which pushes the numbers 0 thru $3F on the stack. Then this routine drops back into the 'TINY' interpreter. These low literals (0-63) thus compile into fast single byte p-codes. Frequently used Variables can be stored at these memory addresses making this doubly useful, e.g., User Variables.

For codes $40 thru $7F the above branches fail and we drop into a nucleus CODE routine. This is done by a look-up table which costs two bytes per entry just as the Code Address normally does. We can support up to 64 CODE routines this way. Despite the time taken for bit testing this structure works out quite fast because only one byte has to be fetched. Of course we could arrange for more than 64 CODE entries by defining one that gives access to a three byte structure, but 64 should be enough.

The effect of these one byte instructions is to make the body of COLON definitions much smaller. Literals require 1, 2 or 3 bytes instead of 2, 3 or 4 bytes. On the other hand CONSTANTS and VARIABLES will usually require 3 instead of 2 bytes since in TINY they are compiled like Literals. But these words are infrequent in FORTH because parameters are passed on the stack.

		TINY	JSR/RTS	DTC	ITC
1. CODE	cycles	21	12	19	28
	d.bytes	5	1	3	5
(NEXT)	p.bytes	1	3	2	2
2. ': ;'	cycles	100	24	111	105
	d.bytes	1-1/2	1	5	4
	p.bytes	2	3	2	2
3. Storage	cycles	30 to 48	? to 58	? to 48	? to 63
e.g.	d.bytes	1	3	3	2
CONSTANT	p.bytes	1 to 3	3	2	2
4. Literals	cycles	30 to 48	? to 59	49 to 54	? to 48
	p. bytes	1 to 3	4 to 5	2 to 4	2 to 4
5. A Line	cycles	250	246	284	340
Lo level	p.bytes	14	29	24	23
6. A Line	cycles	****1237	1056	1412	1591****
Hi level	p.bytes	17	29	24	23
7. Program	d.bytes	285	155	465	455
Storage	p.bytes	930	1750	1440	1380
	total.bytes	****1215	1895	1905	1835****

Table comparing p-codes: d.bytes = dictionary overhead
p.bytes = length of p-code required

A-17

The table above analyses three forms of overhead:

1. Time overhead in cycles

2. Dictionary building overhead d.bytes

3 P-code required each time the entry is called p.bytes

Sections 1, 2 and 3 analyse FORTH words of type CODE, COLON, CONSTANT, and Section 4 analyses Literals.

In Section 5 we find the time over head to execute a line assumed to contain 6 CODE, 1 Literal, and 2 Storage (CONSTANT) words, as well as the space for its p-code. Some of the numerals have been assumed low.

Section 6 is like Section 5 except that 3 of the CODE words have been replaced by 3 COLON words of the type in Section 5. At this level we can get a good idea of comparative speeds of execution.

Section 7 gives the storage required for a program of 35 CODE, 20 storage, and 60 COLON words (drawing equally from Sections 5 and 6). This does not include the space for actual data nor for the machine code of the nucleus, but does include all p-code and dictionary overheads apart from the headers.

Taking ITC as 100% we see that the performance becomes:

	TINY	JSR/RTS	DTC
Time Overhead	78%	66%	89%
Space Required	66%	103%	104%

The benchmarks are for the 6502, but similar ranking seems likely for other 8 bit micros. Clearly, longer programs will favor TINY even more. On the other hand JSR/RTS may execute even faster than indicated because the nucleus can make freer use of the cpu registers.

An important aspect of FORTH is the access it gives the user to the structure of the language. Therefore I would still like to see ITC remain the preferred form because of its elegence and flexibility. But TINY has much to offer on small 8 bit systems.

[1] Sirag, D: "DTC v/s ITC for FORTH" FORTH DIMENSIONS Vol. 1, No. 3, Oct./ Nov. 1978

;s

FORTH in Literature

At the FORTH Convention, October, 1979, Dan Slater gave a short report on an experiment on communication with killer whales. By use of a touch sensitive plate, the orca could learn to physically equate touching a position with a concept or object. Interest was expressed in using the syntax of FORTH to define new items. By this method a man/whale vocabulary can be built.

The evening Charles Moore read a FORTH poem by Ned Conklin. It is loosely based on a classic of English literature.

```
:   SONG
      SIXPENCE  !
  BEGIN  RYE  @  POCKET  +!  ?FULL  END
  24  0  DO  BLACKBIRD  I  +  @  PIE  +!  LOOP
  BAKE  BEGIN  ?OPENED  END
  SING  DAINTY-DISH  KING  !  SURPRISE  ;
```


Bill Ragsdale has submitted two more. This is a familiar quotation, with apologies to Browning:

```
:    LOVE
     CR ." How do I love thee?"
     CR ." Let me count the ways."
          1 BEGIN  CR  DUP  .  1+ AGAIN  ;


:    RHYME
       JACK  DUP  NIMBLE  BE
             DUP  QUICK  BE
       CANDLE-STICK  OVER  JUMP  ;
```

Finally here is an actual, full poem. It is taken from "The Space Childs Mother Goose" by Frederick Winsor, Simon and Schuster, 1958. It consists of eleven stanzas and is almost recursive.

The first two screens compile the primitives from which the poem is recited, by loading of the last screen. The computer's recitation occurs stanza by stanza with the operator indicating his interest and approval by operating any terminal key at the REST after each stanza.

```
SCR # 108
  0 (  The Theory that Jack built                          WFR-79DEC15 )
  1 ( From The Space Child's Mother Goose, Frederick Winsor )
  2 : RECITE      110  LOAD  QUIT  ;   ( say this poem )
  3 : THE               ." the "   ;
  4 : THAT        CR    ." That "   ;
  5 : THIS        CR    ." This is "  THE   ;
  6 : JACK              ." Jack built"  ;
  7 : SUMMARY           ." Summary"  ;
  8 : FLAW              ." Flaw"  ;
  9 : MUMMERY           ." Mummery"  ;
 10 : K                 ." Constant K"  ;
 11 : MAZE              ." Erudite Verbal Maze"  ;
 12 : PHRASE            ." Turn of a Plausible Phrase"  ;
 13 : BLUFF             ." Chaotic Confusion and Bluff"  ;
 14 : STUFF             ." Cybernetics and Stuff"  ;
 15 : THEORY            ." Theory "  JACK  ; -->
```

```
SCR # 109
  0 (  More Poem                                           WFR-79DEC15 )
  1 : BUTTON           ." Button to Start the Machine"  ;
  2 : CHILD            ." Space Child with Brow Serene"  ;
  3 : CYBERNETICS    ." Cybernetics and Stuff"  ;
  4 : HIDING       CR  ." Hiding " THE  FLAW  ;
  5 : LAY       THAT  ." lay in "  THE  THEORY  ;
  6 : BASED     CR    ." Based on "  THE  MUMMERY  ;
  7 : SAVED        THAT  ." saved "  THE  SUMMARY  ;
  8 : CLOAK  CR    ." Cloaking "  K  ;
  9 : THICK IF THAT ELSE CR ." And " THEN ." Thickened " THE MAZE ;
 10 : HUNG          THAT  ." hung on " THE  PHRASE    ;
 11 : COVER IF THAT ." covered " ELSE CR ." To cover " THEN BLUFF ;
 12 : MAKE       CR   ." To make with "  THE  CYBERNETICS  ;
 13 : PUSHED     CR   ." Who pushed "  BUTTON  ;
 14 : REST       46  EMIT  10  SPACES  KEY  DROP  CR  CR  CR  ;
 15 : WITHOUT  CR ." Without Confusion, exposing the Bluff" ; RECITE
```

```
SCR # 110
  0 (  Recite our poem                                     WFR-79DEC15 )
  1 CR  CR  CR  THIS  THEORY  REST
  2 THIS  FLAW  LAY  REST
  3 THIS  MUMMERY  HIDING  LAY  REST
  4 THIS  SUMMARY  BASED  HIDING  LAY  REST
  5 THIS  K  SAVED  BASED  HIDING  LAY  REST
  6 THIS  MAZE  CLOAK  SAVED  BASED  HIDING  LAY  REST
  7 THIS PHRASE 1 THICK CLOAK SAVED BASED HIDING LAY REST
  8 THIS BLUFF HUNG 1 THICK CLOAK SAVED BASED HIDING LAY REST
  9 THIS STUFF 1 COVER HUNG 0 THICK CLOAK SAVED BASED HIDING
 10   LAY REST
 11 THIS BUTTON  MAKE 0 COVER  HUNG 0 THICK  CLOAK  SAVED
 12   BASED  HIDING  LAY  REST
 13 THIS CHILD PUSHED CR ." That made with " CYBERNETICS  WITHOUT
 14   HUNG CR ." And, shredding " THE MAZE CLOAK CR ." Wrecked " THE
 15   SUMMARY BASED HIDING CR  ." And Demolished " THEORY REST
```

FORTH, Inc. News

A series of free seminars and paid ($100-125) workshops is being offered; polyFORTH will be presented. The schedule is: Palo Alto, May 8 & 9; Rochester, NY, May 13; Boston, May 14 & 15; New York, June 10; Cherry Hill, June 11; Washington-Baltimore, June 12 & 13; Houston, June 16 & 17; New York, June 18; Palo Alto, June 24 & 25. For more information and/or to register: Call Kris at FORTH, Inc. (213) 372-8493.

FIG DOINGS

Intensive Short Course

The American Chemical Society is offering a number of five day, hands-on, in-depth lab courses on microprocessors and minicomputers. The participants will have access to a PDP-11 network running FORTH. Sessions are June 8-13, September 7-12 and December 14-19 at VPI, Blacksburg, VI at a cost of $485 for ACS members and $550 for non-members. For more information contact ACS Short Courses, 1155 Sixteenth St., N.W., Washington, DC 20036.

FIG GROUPS FORMED OR FORMING

San Diego - Call Guy Kelly (714) 268-3100 ext 4784 or Tom Boyle (714) 571-7711

Seattle - Call Dwight Vandenburg (206) 542-8370 or Terry Dettman (206) 821-5832

Mass. - Third Wednesday at 7:00 pm in Cochituate, MA. Call Dick Miller (617) 653-6136

Virginia - Call Joel Shprentz (703) 437-9218 or Paul van der Eijk (703) 354-7443

Texas - In Houston, call Jeff Lewis (713) 729—3320, in Dallas, call John Earls (214) 661-2928 and in Denton, call Dean Vieau (313) 493-5105

Arizona - Call Dick Wilson (602) 277-6611 ext. 3257

Oregon - Call Ed Kammerer (503) 644-2688

New York - Write Tom Jung, 704 166th St., Whitestone, NY

Michigan - Call Dwayne Gustaus (817) 387-6976

OTHER PUBLICATIONS

Dick Miller has sent the first issue of the MMS FORTH Newsletter. It's jam packed with news, tips and updates for MMS FORTH users on the TRS-80.

It's a service to registered owners, and Dick would be glad to send a sample copy to prospective users. Write Miller Microcomputer Services, 61 Lake Shore Road, Natick, MA 091760.

* * *

Thanks to Fig member Frank Dougherty (325 Beacon Street, Belvedere, IL 61008) for the writeup in the Blackhawk Bit Burners Newsletter. Frank discussed the language and our efforts, as well as the dialect STOIC.

FORTH for Microcomputers by John S. James originally published in Dr Dobbs Number 25 May 1978 has been reprinted first in ACM SIGPLAN NOTICES, Oct. 1978 and now in an IEEE Tutorial: MICROCOMPUTER PROGRAMMING AND SOFTWARE

SUPPORT, IMSONG LEE, EDITOR, IEE cat No. EH0 140-4 to quote from this publication "James gives a compact, but not necessarily easy, introduction to a stack oriented, interactive programming language called FORTH. A better tutorial presentation may be found in the manual, PROGRAM FORTH, A PRIMER, by Gregg Howe, Steward Observatory, University of Arizona, 1973." The current availability of this document is unknown.

More on STOIC-II

Technical Report TR-001

"EDIT79, A STOIC-II Programming Example" (63 pages) $7.00

This report represents and example of a non-trivial program written entirely in STOIC-II. The program, a text editor, was cross-compiled to produce a stand-alone object program, thus facilitating benchmark comparisons with the CP/M editor which it closely resembles. Included in the report are the benchmark results, a brief user's guide, and source code for the editor along with extensive comments.

Contact: Jeff Zurkow
Avocet Systems
804 South State Street
Dover, DE 19901

KIM HARRIS COURSE

A five day intensive course on programming with FORTH will be held July 21-25 at Humbolt State University in Arcata, California. The course will cover the FORTH approach to producing computer applications including: (1) analyzing the requirements of a problem, (2) designing a logical solution, solution. Topics will include the usage, extension, and internals of the FORTH language, compiler, assembler, virtual machine, multitasking operating system, mass storage virtual memory manager, and file system. Computers will be available for demonstrations and class exercises. The course will be taught by Kim Harris, and Humbolt State University will give 4 units of credit through the office of Continuing Education. Tuition for the course is $112 per student. The text will be "Using FORTH"; copies will be available at the course for $25 each. Housing is available in very nice dormitory rooms for $9 per person per night or at several nearby motels. Cafeteria meals may be purchased individually or at $10.25 per day. For more information and registration materials write, before June 23:

Prof. Ronald Zammit
Physics Department
Humbolt State University
Arcata, California 95521

RENEW NOW!

tiny-FORTH

A version of fig-FORTH tailored to the TRS-80, Level II with 16K bytes of memory minimum. I/O devices supported are: keyboard, display and cassette tape recorder. New words can be defined to control other devices. The editor is identical to the fig-FORTH editor and the output format is modified slightly to fit the TRS-80 display. Documentation includes: introduction, editor, graphics, assembler, advanced tiny-FORTH and applications. The style is tutorial with hundreds of examples that teach tiny-FORTH in a hands-on mode. $29.95 for tiny-FORTH cassette and full documentation for the Level II, 16K TRS-80 plus $1.50 for shipping and handling ($5.00 outside the US). The Software Farm, P.O. Box 2304, Reston, VA 22090.

KIM-1 FORTH

This version was written from the FIG model by Ralph Deane of Vancouver, Canada. It contains a complete programming system which has an interpreter/compiler as well as an assembler and editor. All terminal I/O is funnelled through a jump table near the beginning of the software and can easily be changed to jump to user-written I/O drivers. 6502 FORTH resides in 8K of RAM starting at $2000 and can operate with as little as 4K of additional contiguous RAM. $94.00 for 6502 FORTH system on KIM cassette. $16.50 for 6502 FORTH user manual. Eric C. Rehnke, 540 S. Ranch View Circle, #61, Anaheim Hills, CA 92087.

Heath H89 and H8

FORTH for the Heath 89 and 8 is possible with the fig-FORTH 8080, Version 1.1 (as demonstrated by Jim Flournoy at the January FIG meeting). Walter Harris implemented and brought up the code on his dual disc H8 and reassembled it for the H89. For more information, contact: FORTH Power, P.O. Box 2455, San Rafael, CA 94902.

HONEYWELL FORTH SYSTEM

Source Data Systems announces a language for non-programmer data definition, transaction definition, file definition and report generation for Honeywell Level 6 Minis. Designed for information management and retrevial when used together is SDS's Source Data Entry package. For more information, contact: Source Data Systems, 208 2nd Avenue, S.E., Cedar Rapids, IA 52406.

AMD 2901 FORTH PROCESSOR

Functional Automation unleashes the I/O thing, a FORTH based front end processor for its AMD 2901 based 64 bit wide microprogrammed computing engine. The system programming language is FASL (Functional Automation System Language) which is available users. For more information, contact: Functional Automation Inc., 3 Graham Drive, Nashua, NH 03060.

STOIC

STOIC, essentially an extension of FORTH, is a general purpose interactive

program, assembler, debugger, loader and operating system within a single consistant architecture. With core efficiency and high running speeds, the language is extremely flexible permitting the user to develop a working vocabulary of subroutines tailored to specific applications.

The entire package, including a library of predefined subroutines, is copyrighted but available to educational users. STOIC requires three discs, two are STOIC itself and the third contains a bootstrap that permits the entry of STOIC through CP/M and the continued use of CP/M disc I/O under STOIC. For more information, contact: Steven Burns, Massachusetts Institute of Technology, Room 20-119, Cambridge, MA 02139.

68'FORTH FOR 6809

68'FORTH is a 6809 implementation of the FORTH Interest Group standard vocabulary of this powerful language.

68'FORTH consists of full FORTH Interest Group standard (May 1979) vocabulary with names to 31 characters, 16 and 32 bit integer math, compiler error checking, and source text editor. System is supplied with additional vocabulary to simulate disk in memory (useful for modifying to work with other disk systems or enabling cassette-only operation), to use disk for virtual memory (allows large data sets to be used in small memory), to interface with FLEX 9.0 text files for input and output, and to perform standard FORTH disk block read and write. System is supplied on 5" floppy disk configured for SWTPC MF-68. Minimum memory requirement is 8k for FLEX plus 12K of user space. Documentation contains description of all vocabulary words, information on configuring for individual system, and basic tutorial for FORTH language. Information is available for reconfiguring to interface with other disk operatings systems.

FLEX 9.0 format 5" disk plus documentation: $39.95.

Talbot Microsystems
7209 Stella Link, Suite 112
Houston, TX 77025

PRODUCT RELEASE

8080 Assembler Available

John Cassady, who did the original fig-FORTH 8080 listing, has now released an 8080 FORTH assembler. John's assembler handled all Intel nmemonics and can easily be altered to Zilog, as it is published as source code. It handles structured assembly conditionals IF, ELSE, THEN, BEGIN, UNTIL, WHILE, and REPEAT. It is integrated with the FIG security package to verify correct structuring of conditionals, during assembly. John provides for named subroutines as well as CODE definitions.

Send $3.25 (includes postage) to John Cassady, 339 15th Street, Oakland, CA 94612.

PolyFORTH-CP/M

polyFORTH-CP/M is FORTH Inc.'s polyFORTH, interfaced to run on nearly any 32K or larger CP/M based system. When loaded, polyFORTH-CP/M finds and links-up to the CP/M I/O drivers, initializes itself, and responds "up" on the system console. At this point, true polyFORTH is running, that is, FORTH structured (screen oriented) diskettes must be used. It is important to realize that polyFORTH-CP/M does not run under CP/M, it runs in place of CP/M, utilizing only the CP/M I/O drivers.

The polyFORTH-CP/M system, as supplied by M&B DESIGN, is a value-added system. FORTH Inc.'s complete 8080 polyFORTH system is supplied, plus an additional diskette and manual containing interface material. Also provided, is a CP/M utility that allows transferring polyFORTH blocks (screens) to a CP/M file, and transferring a CP/M file to polyFORTH blocks. Source is supplied for the entire polyFORTH system, the polyFORTH-CP/M components, and the transfer utility.

polyFORTH-CP/M is available directly from M&B DESIGN for $4,000 on a wide variety of diskette formats. Contact:

M&B DESIGN
820 Sweetbay Dr.
Sunnyvale, CA 94086
(408) 243-0834

FORTH for Poly-88

fig-FORTH for the 8080 as implemented by John Cassady and modified by Kim Harris is now available for the Poly-88.

This version uses cassette and ram simulation of disc, and includes full use of upper and lower case characters as well as the Greek character set, as well as high speed graphics. An editor and an assembler are included.

The complete system price is	$50.00
FORTH on cassette (needs 8K RAM 2000-4000H)	25.00
Poly-88 Forth User's Guide	10.00
8080 fig-FORTH Source Listing	10.00
Installation Manual (F.I.G. Model)	10.00

(CA residents add 6% tax)

Write: Jeff Fox (415) 843-0385
2223 Byron
Berkeley, CA 94702

LOTS OF FORTH

ANCON provides a wide variety of FORTH products, including: Hobby versions; TRS-80 Cassette, $29.95; Heath H8-H89, $49.94; 8080 CP/M 8in, $49.95; 6809 5" Flex, $49.94.

Personel systems; TRS-80 Tape, $45.95; Disc, $65.95; 8080 CP/M 8" $125.00; pdp-11, $140.00; Northstar, Micropolis.

Commercial/Industrial/Scientific versions available for specific requirements. Jim Flournoy, ANCON, 17370 Hawkins Lane, Morgan Hill, CA 95037, (408) 779-0848.

RENEW NOW!

LYON'S DEN

[Editors note — George Lyons has corresponded on FORTH topics over the full life of FORTH DIMENSIONS. He addresses technical and philosophical topics. We're formalizing his contributions into a bylined column. Welcome to the Lyon's Den.]

I suspect that a paramount issue in decisions on whether to use FORTH or another language is a tradeoff between language convenience and compiler convenience. By implementing a complex syntax a PASCAL compiler, say, automates part of the programming task at the expense of a time-consuming source entry and processing operation. Standard FORTH seems to be at the other extreme, leaving more explicit details to be coded by the programmer, for the gain of easier processing with an interactive resident compiler.

The polarization of FORTH and its alternatives on this scale may only be due to the absence of standard FORTH vocabularies to provide the same degree of automation traditional languages supply. I wonder if a quantum jump in FORTH's popularity would result from supplying compilers for traditional languages implemented in FORTH. Possibly, transparently obvious FORTH language features could be provided achieving the same results. The areas where the greatest impact might come are PASCAL data structures, ALGOL procedure argument passing and dynamic local storage allocation, and APL matrix algebra.

If techniques for these operations in FORTH were widely known no one would make the mistake of referring to FORTH as a species of macro assembler. By demonstrating that traditional language convenience is available in FORTH users might be motivated to take advantage of the extensibility of FORTH to go beyond the limitations of the traditional approaches.

March 14, 1970

George B. Lyons
280 Henderson Street
Jersey City, N.J. 07302

Programma was nice enough to supply me with a reassembled version of their Apple-FORTH Kernel plus screens of the dictionary entries for my KIM-1. This was all entered by hand, painfully debugged, editor programs written (in their FORTH), etc. I then tried to duplicate the "PI" routine in the Dr. Dobbs Journal, only to find that Programma didn't carry extra bits of intermediate accuracy in the multiply routine. Then another week of spare time (midnight oil) work to rewrite the math routines to allow "*/MOD" to work properly. It finally worked.

I'm not bitter though. Through all of this I learned enough of FORTH-like programming to be more enthusiastic than ever, but disappointed that the example programs I've received from you are not usable by the Programma version. I am therefore eagerly awaiting the availability of the 6502 version of fig-FORTH for my KIM-1.

Edward J. Bechtel, M.D.
Newport Beach, CA

Should you have any books, manuals or other documents pertaining to FORTH which are available by special order, I would like to have a list. There seems to be a real need for textbooks or tutorials which will carry a user from the most simple FORTH constructions to the very elaborate ones like <BUILDS and DOES>. (See #1 below.)

For your information, I am working with the mmsFORTH implementation from Miller Microcomputing Services. I am quite satisfied with the system to date, and look forward to other extensions. I have distributed several FORTH programs to MMS which they may use in their newsletter. Should the FORTH Interest Group have a program exchange or publish programs, I will submit these programs to you also. (See #2 and #3 below.)

Andrew W. Watson
Vinton, VA

Editor...

#1 - Use the Mail Order page and see Information and New Products sections of FD.

#2 - Send programs to FD!

#3 - Address for Miller Microcomputing Services, 61 Lake Shore Road, Natick, MA 01760.

I want to tell you how impressed I am at the quality of the Installation Manual and the 6800 Assembly Source Listing!

The 6800 listing provided everything I needed to build an identical source file. The Symbol Table and hex dump were especially useful in tracking down the last small typos. (I used the check sums for the 'S1' dump to locate typos such as INS instead of INX.) To get FORTH running on my system, all I did was to modify the ACIA address and delete the coding for Trace.

I notice a peculiar behavior regarding the stack. If I type . when the stack is empty, I get an error message, as expected. But after the error message, there are two numbers on the stack. Is this normal?

Gordon Stallings
Bartlesville, OK

Editor...

The numbers left on the stack after an error are the block number and character offset. See ERROR. This allows WHERE (Scr88) to display the offending text.

Thanks to John James and FIG, I've upgraded my sub-FORTH to what I now call 2650-FOURTH. To date, except for the disk I/O verbs, my FORTH more or less matches Mr. James' FORTH with the exception that I've incorporated an assembler, it's fully ROM based and it has a few more primitives. I do support a cassette I/F but can't use the full power of the fast virtual storage. I will release a copy of my 2650-FORTH to FIG as well as any application work that I've done.

Myself being broadly classified as a computer architect or computer designer, I have a very keen interest in turning out a FORTH engine (to borrow a phrase from Western Digital), and will attempt the implementation. I will probably use the 2900 series bit slices since I have all the development tools. Is there someone in this vein that I could contact?

Edward J. Murray
Pretoria, Union of South Africa

Editor...

Look forward to receiving your 2650-FORTH. Address for John S. James, P.O. Box 348, Berkeley, CA 94701.

I was somewhat disconcerted when I read the article by Mr. David J. Sirag, "DTC Versus ITC for FORTH on the the PDP-11", FORTH Dimensions, Volume 1, No. 3. The author has, I believe, misunderstood the intent of the article by Mr. Dewar.

In Mr. Dewar's ariticle, the definitions of direct threaded code (DTC) and indirect threaded code (ITC) are:

> "DTC involves the generation of code consisting of a linear list of address of a linear list of address of routines to be excuted."
>
> "ITC..." (involves the generation of code consisting)"... of a linear list of addresses of words which contain addresses of routines to be executed."

As applied to the FORTH type of hierarchial structure (hierarchial indirect threaded code?), I would extend Mr. Dewar's definition to be:

> "ITC involves the generation of code consisting of a linear list of addresses of words which contain addresses of routines to be executed. These routines may themselves be ITC structures."

However, Mr. Sirag based his conclusions on the following loose definition:

> "The distinction between DTC and ITC as applied to FORTH is that in DTC executable machine code is expected as the first word after the definition name; while, in ITC the address of the machine code is expected."

Obviously, the two men are not referring to the same things. Mr. Dewar is referring to the list of addresses which define the FORTH word, while Mr. Sirag is referring to the implementation of the FORTH interpreter. If indeed Mr. Sirag's statement were true (which it is not) that their "analysis contradicts the findings of Dewar", then they should have implemented a DTC language rather than the ITC language of FORTH! Indeed, a careful examination of what is actually occuring in LABFORTH reveals that their techniques are logically identical to Dewar's ITC. They have simply, through clever programming, taken advantage of a particular instruction set and architecture. It is beyond the scope of this letter to prove this equivalence, or to suport the FIG desire to have a common implementation structure for all versions of FIG FORTH.

Please note that I am not quibling over semantics with Mr. Sirag. All definitions are arbituary. (However, the value of a definition lies in its consistency, precision, and useability. I find Mr. Sirag's definition of DTC and ITC to be inconsistent with the environment in which he operates, FORTH, and thus quite useless.) My intent is two fold: (1) I am a self-appointed defender of the excellent work of Mr. Dewar, and (2) I want to correct any misconceptions concerning this issue for readers of this newsletter who did not have access to Dewar's (better) definition of DTC and ITC.

Jon F. Spencer
Sherman Oaks, CA

Many thanks to John Cassady for writing an excellent 8080 FORTH and to Kim Harris for implementing the necessary mods. I received 8080 fig-FORTH Ver. 1.1 on 2 October 1979 and within a few days had the assembly language source typed in and assembled. A day or two later the editor with a suitable patch for the MATCH code was up and running along with the disk based long error messages. I have been learning and gaining experience with fig-FORTH ever since.

After more than a year of using STOIC from volume #23 of the CP/M Users Group it is really nice to be using a true FORTH that is consistant with the examples in the FORTH Inc. manuals and

the several articles that have appeared on FORTH. I cannot over-emphasize how well documented the fig-FORTH system is and how easy the system was to bring up. No bugs or errors have been uncovered in nearly six months of use.

The only thing missing from this otherwise nearly perfect package is the assembler vocabulary. Is an 8080/Z-80 assembler vocabulary available from the FORTH Interest Group or if not is any planned? If an 8080 assembler is available or is planned a short note or a word about future plans in the next issue of FORTH DIMENSIONS would be sufficient.

In any case I hope I get to see some of you at NCC in May so that I can personally thank you for making FORTH available to me...

Sincerely,

M. Paul Farr
2250 Ninth Street
Olivenhain, CA 92024

Editor --

Yes! An 8080 assembler is now available in source code to complement 8080 fig-FORTH. Send $3.25 (includes postage) to John Cassady, 11 Miramonte Road, Orinda, CA 94563.

Many thanks for the fig-FORTH installation manual glossary and FORTH Model, which have been difficult but enjoyable items of study since they arrived a couple of weeks ago.

Like many of your members I became interested in FORTH without having access to a FORTH system, and gained my first practical familiarity by using the FORTH low level interpreter style of linkage on machine code programs.

With help from the Model I have now got to grips with the outer interpreter and virtual memory system, and will be getting together with Bill Powell and other FORTH fanciers over here on an cooperative implementation effort.

Many thanks for your effort and creativity, which are not unappreciated!

Bill Stoddart
15 Croftdown Road
London NW5, England

Editor's note -- Bill had a marginal note to this letter: "certainly grows on you. This really is 'Computer Liberation.' BASIC was just a red herring."

Do you have a Z80 version of fig-FORTH? It is not listed on your order sheet but reading the text I got the impression that you do.

By the way, I have a tutorial paper discussing assembly programming in FORTH environment for both 8080 and Z80. It is available, including source listing written in fig-FORTH, from KALTH microsystems. The price is $5.-US for 8080/85 version, $7.-US for Z80 version or $10.-for both (add 15% in Canadian funds).

Also, I am working on the assembler for the Intel 8086/88. If I knew that there are also other people interested in it, that would motivate me getting it complete sooner. (It is a cross-assembler that can be run on any FORTH based system.)

Yours truly,

Kalman Fejes
KALTH microsystems
P.O. Box 5457, Station "F"
Ottawa, Ont., Canada

Editor --

Fig doesn't have a plan a Z-80 version of fig-FORTH. We would be pleased to publish a contributed version, if as complete as the 8080 Version 1.1

As a participant in the Forth International Standards Team, I cast a yeah vote for the inclusion of "TO" and its requisite definition of VARIABLE (though I prefer the name FIELD). Although I was first exposed to this definition on Catalina Island, it has many similarities to my own implementation of FIELD and RECORD.

In its simplest form, as outlined by Paul Bartholdi, FORTH DIMENSIONS 1/4, integer variables of predetermined precision are defined to behave as bidirectional constants. Normal behavior is to push their stored value onto the stack. A momentary, alternate behavior is to pop the stack value into their confines. This temporary behavior occurs only when referenced after the word "TO", which sets a direction flip-flop. Thus

```
VARIABLE A          VARIABLE B
10 TO A             A TO B
```

will place 10 into A and B without using the @ (fetch) and ! (store) operators.

Each of us, who has implemented a version of "TO", encounters some exasperation in dealing with the addresses appearing on the stack. Since, in the prior illustration, neither A nor B supplied its address for TO's execution we ponder the shortcomings of this newly offered definition and reluctantly sprinkle our procedures with @ and !.

FORTH is an elaboration on the indirect threaded list program architecture. As programmers we are free to add indirection to our methods of accessing and manipulating data. Indirection, however, is only a navagation technique for constructing the address required by the hardware to implement our desired operation. When at the end of our circuitous logic, are we then to complain "What can I do about this address".

Let's face it, @ and ! are perfect operators.

I value TO and its implications in system structure. The procedures written using "TO" are more readable than standard Forth, and result in fewer visits to NEXT as they are executed. "TO" will be included in any system I generate, together with other essential words, which include @ and !.

As a Forth fanatic and a FORTH DIMENSIONS fan I sincerely hope that the newsletter will continue. If there is some assistance I can render please advise.

Williams S. Emery
2700 Peterson Place, #53D
Costa Mesa, CA 92626

Editor --

You're doing it! By thoughtful correspondence and participation in group events people such as Bill are multiplying our efforts.

STRUCTURED VARIABLES

From time to time at the Fig meetings the question of structured variables arises. This is a proposal for how they might be handled.

The December 1978 issue of communications of the ACM contained a paper by John Backus on "Functional

Programming" (also called variable free programming). I believe a variation of his ideas could be implemented in FORTH. Suppose we are given a pair of queues with bases at opposite ends of available memory pointing toward each other. Then enter an array into one of them and begin processing it. Let the results go to the other queue as they are developed. Multiple steps would alternate between the queues until a final result is obtained. These alternating queues can give some of the effects of functional programming (1) large state changes, (2) limited memory of past states, (3) no concern with garbage collection, 4) variables not named or declared.

Backus placed operators within the data. This could be done or not, as experience dictates. These queues are not to replace the stack which FORTH already has. The stack could be used to hold what I would call operator variables or modifiers.

Let us look at a couple of simple examples. Suppose we wanted to transpose an array. 1 2 3
4 5 6
Enter it into one queue. [1 2 3 4 5 6]. Type in the transpose command. 2 1 TP. The 2 and the 1 go on the stack so the transpose function knows what kind of a transpose is desired. The result will come out on the other queue: 1 4 2 5 3 6]

Should we wanted to sum a vector. [1 2 3 4 5 6]. Type in a reduction command. ' + RD. The 'Tick' put the address of 'Plus' on the stack so the reduction function knows what kind of reduction to perform. The other queue receives the result: 21]

W. H. Dailey
47436 Mantes Street
Fremont, CA 94538

FORTH IN THE PUBLIC VIEW

After the survey article in March 15, 1979 Electronics, Mr. Robert Gaebler wrote the usual letter to the editor critiquing FORTH's postfix notation. We are reprinting a well stated rebuttal to this letter which also appeared in Electronics.

To the Editor:

I want to reply to Robert Gaebler's letter on expression format in the FORTH language [Electronics, July 5, 1979, p. 6].

Gaebler notes, and I agree, that compilers can do the translation from infix to postfix notation and thus save the programmer both work and the risk of errors. Unfortunately, these advantages are not available without some penalty for extensible languages such as FORTH. If the compiler is to translate, it must know how to parse expressions. The parsing rules for primitive operators are supplied with the compiler, but those for the added operators must be supplied by the programmer at compile time, which makes the parser much more complicated.

Examination of almost any program will reveal that the majority of program statements are nonalgebraic or can easily be converted to a non-algebraic form. Thus the advantages of infix notation, when present, apply only to a fraction of the program statements. For most function definitions, the prefix notation of subroutine or macro calls is required, and this can be replaced by postfix notation with little or no loss of clarity.

Use of postfix notation leaves the parsing of all expressions in the hands of the programmer. It means that arguments for an operator may be

prepared using the full power of the programming language, without any restrictions being imposed by the compiler. With this freedom comes the possibility of error, and argument preparation is one of the most error-prone portions of programming in a language such as FORTH. If effort is to be spent on improving the ease of programming, it should be spent on simplifying argument preparation and stack manipulation. Postfix notation, with the applicative style of programming that it produces, has so many advantages that it should not be sacrificed to an algebraic notation that is not "natural," but only something we all learned in school.

THIS IS THE END!

THE END OF VOLUME II #1!

THE END OF YOUR MEMBERSHIP?

DON'T LET IT HAPPEN!

RENEW TODAY!

CHECK THE LABEL FOR RENEWAL DATE!

SEND A CHECK TO F!G TODAY!

MAKE THIS YOUR BEGINNING!

RENEW NOW!

MAKE A COPY FOR A FRIEND!

POST COPY ON YOUR BULLETIN BOARD!