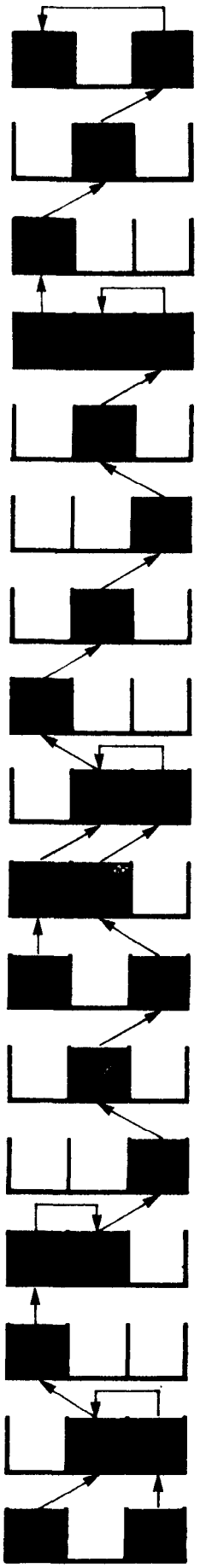

FORTH DIMENSIONS

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

Volume 1
Number 5
Price $2.00

INSIDE

FORTH DIMENSIONS

Published by Forth Interest Group

Volume 1 No. 5 Jan/Feb 1980

Publisher Roy C. Martens

Editorial Review Board

Bill Ragsdale
Dave Boulton
Kim Harris
John James
George Maverick

Subscription to FORTH DIMENSIONS is free with membership in the Forth Interest Group at $5.00 per year ($9.00 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 950 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.

PUBLISHER'S COLUMN

Forth Interest Group has come of age. FIG now has a publisher for FORTH DIMENSIONS. Roy Martens will handle all facets of putting together and getting out future issues of FORTH DIMENSIONS. He comes to FIG with a solid background dating back to Hughes Aircraft Company in the 50's to Singer Business Machines and American Micro Systems in the 70's. He has publication experience gained from Hayden Publishing Company (Electronic Design, Computer Decisions, etc.), CBS and EW Communications. Welcome aboard, Roy!

S. Figgie

Thanks. I look forward to working with FIG and hope that we can make FORTH DIMENSIONS a useful and timely tool for all members. Please send in your letters, cases, suggestions. Your input will make FORTH DIMENSIONS successful.

Roy C. Martens

$$$$ CASE STATEMENT $$$$ CONTEST

FIG is sponsoring a contest for the best CASE statement for FORTH.

Prize.....$100 ($50 from FIG and $50 from FORTH, Inc.)

Furthermore, entries will be considered as experimental proposals for possible inclusion in the future FORTH Standard.

Contest Rules.....

Submit a CASE Statement, as specified below, to

FIG CASE CONTEST
P.O. Box 1105
San Carlos, CA 94070

Postmarked on or before March 31, 1980.

All entries will be judged by selected, non-entering members of Forth Interest Group. All entries will become public domain and may be published by FIG.

Judging Criteria.....

A. Conformity to rules
B. Generality of statement
C. Simplicity of statement
D. Self-identifying function
E. "FORTH-like" style

Entry Requirements.....

Your entry should contain descriptions of a collection of FORTH words which allow the selection of one of several actions based on a selection criteria. The actions may be single words or groups of words. The selection criteria may be simple or complicated. A variety of situations should be accommodated. Included in your entry should be....

A. An overview of the statement

B. Source definitions in fig-FORTH words for the needed compiler and support words

C. An "English" explanation of how these words work

D. Glossary entries for each word

E. Examples of the use of this statement

F. A discussion on the statement, including advantages and disadvantages, limitations, applications, etc.

Contest Purpose.....

The selection of one of several procedures based on some criteria is a useful and common control structure. Standard FORTH provides two mechanisms for this structure: nested IF structures and execution vectors. It is desirable to have a standard structure to handle a variety of situations.

However these do not appropriately satisfy all situations in terms of source convenience and execution-time efficiency. A simple situation where the selection criteria is a single integer index with a limited, continuous range is adequately met by FORTRAN's computer GOTO or FORTH's execution vectors

At the other end of the complexity scale, if the selection criteria was an index with a non-contiguous range or a series of expressions, the simple statements would require manipulations at the source level and would appear less clear. Because of FORTH's hierarchial modularity, the range of complex situations can be met with a range of structures. The execution-time overhead need not be greater than what the situation requires. The purpose of this contest is to produce a "kit" of compiler words which will allow the optimum specification of case control by combining minimum execution-time overhead with a uniform source language.

;S

"TO" SOLUTION CONTINUED..... "EASTER"

As promised in the last issue, here is the example for the calculation of the dates of Easter from Paul Bartholdi, Observatoire De Geneve, Switzerland.

SCR #9

```
 0 (Dates of Easter, Clavius Algorithm    )
 1 (This algorithm and variable names are)
 2 (from "Fundamental Algorithms" by      )
 3 (D. Knuth.  The method was originated  )
 4 (by the sixteenth century astronomer   )
 5 (Clavius.                              )
 6
 7
 8 0 VARIABLE %VAR     ( if one, store   )
 9 : TO   1 %VAR !  ;  ( set to store    )
10 : FROM  0  %VAR ! ; ( set to fetch    )
13 : ((              (preserve %VAR flag )
14      R>  %VAR  @  >R   >R FROM  ;
15 : ))              ( restore %VAR flag )
14       R>   R>   %VAR  !  >R ;
15
16  —
```

SCR #10

```
 0 ( Simplified TO          DAB-79OCT29 )
 1
 2 : VARIABLE  ( defined to observe TO )
 3  <BUILDS   0  , DOES>  %VAR   @
 4    IF ( set, so store )  !  FROM
 5     ELSE ( clear, so fetch ) @ THEN ;
 6
 7
 8 VARIABLE C  VARIABLE D  VARIABLE E
 9 VARIABLE G  VARIABLE N  VARIABLE K
10 VARIABLE X  VARIABLE Z  VARIABLE YEAR
11
12
13  —>
14 Note that this demonstration does not
15 include the newer  +TO ,  AT  etc.
16
```

SCR #11

```
 0 ( Dates of Easter          DAB-79OCT29 )
 1 : (EASTER)        ( year —— day  day )
 2   (calculate date relative to March 1)
 3   DUP  DUP  TO  YEAR  19  MOD 1+ to  G
 4    100   /   1+   DUP   DUP   TO   C
 5   3   *  4  /  12  -  TO  X
 6    8   *   5   +   25   /   5   -   TO   Z
 7   YEAR  5  *  4  /  X  -  10  -  TO  D
 8   G  11  *  20  +  Z  +  X  -  30  MOD
 9   DUP  0<  IF  30  +  THEN
10   DUP  DUP  TO  E
11   25  =  G  1  >  AND  SWAP  24  =  OR
12   IF  E  1+  TO  E  THEN
13   44  E  -  DUP  TO  N
14    21  <  IF  N  30  +  TO  N  THEN
15   N  DUP  7  +  SWAP  D  +
16   7  MOD  -  DUP  DUP  TO  N  ;  —>
```

SCR #12

```
 0 ( Dates of Easter          DAB-79OCT29 )
 1 : EASTER  ( year ---print one Easter )
 2         (EASTER)  31  >
 3    IF   31   -   5   .R   ." APRIL  "
 4       ELSE   5  .R  ." MARCH  "  THEN
 5       YEAR   5  .R   ;
 6
 7 : EASTERS  (Begin year, end year —— )
 8  ( print Easter for a range of years.)
 9  PAGE 32 SPACES ." DATES OF EASTER" CR
10  1+  0  SWAP  ROT
11  DO  I  EASTER  1+
12     DUP   4  MOD   0=   IF   CR   THEN
13     DUP   240  MOD   0=   IF  PAGE  THEN
14  LOOP  CR  PAGE  DROP  ;
15
16
```

EXAMPLE

10	April	1955	1	April	1956	21	April	1957	6	April	1958
29	March	1959	17	April	1960	2	April	1961	22	April	1962
14	April	1963	29	March	1964	18	April	1965	10	April	1966
26	March	1967	14	April	1968	6	April	1969	29	March	1970

A MODEST PROPOSAL FOR DICTIONARY HEADERS

Robert L. Smith
Palo Alto, CA

The new fig-FORTH model has improved the utility of FORTH by allowing dictionary names to be of any length, up to the current maximum of 31 characters. Most previous implementations stored only the first three characters of the name, along with a count field. Confusion could easily arise, say between the words "LOOK" and "LOOP". The maximum word length stored in the fig-FORTH model can be changed dynamically by changing the value of WIDTH. If one wishes to return to the former restriction of 3 character names, the new model will still be an improvement because 1 and 2 character names will require less dictionary space.

One advantage of the old dictionary format has been lost, namely a fixed relationship between the link field and the CFA (Control Field Address) or the parameter field. The following proposal will restore the fixed relationship, while at the same time keep the new fig-FORTH advantages. In addition, the maximum allowable word size is increased to 63.

A model of the proposed dictionary entry is shown in Figure 1. The name of the word is stored in reverse order to simplify the dictionary searching. From the link field one can step backwards to obtain the name, or forwards to obtain the definition of the word. The leading bit of the machine representation of each character of the name is set to zero, except for the terminal character which has the leading bit set to one. Thus the effective end of the name can readily be determined. The name field is reversed to simplify the dictionary search procedure, but it is an implementation decision. Obviously one could put the characters in the forward direction, but the dictionary search would take longer. The suggested structure increases the maximum allowable word length to 63 (or possibly 64).

It would be possible to use the leading bit of the first character for the smudge bit, but it would somewhat complicate the dictionary search procedure. One character names would have to be a special case with no terminal bit specified, since that location would be designated for smudging purposes.

This proposal eliminates the need for a number of special words in the fig-FORTH model (TRAVERSE, PFA, NFA, CFA). The only obvious disadvantage is that the routine for printing the dictionary names would need to take characters in the opposite direction from other text. The advantages appear to outweigh the disadvantage.

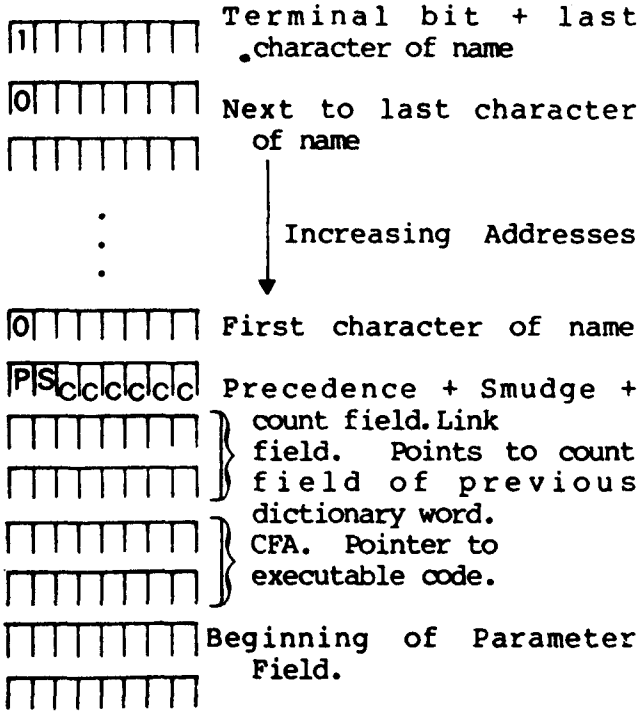

Figure 1. A proposed Dictionary Structure.

;s

FORTH-85 "CASE" STATEMENT

Richard B. Main
Zendex Corp.
Dublin, CA

NEPTUNE UES has recently extended its FORTH-85 with a "CASE" statement. The CASE statement allows an n-way branch based on a condition. Its use is very similar to the PASCAL CASE statement. the IF..(this)...ELSE... (that)... THEN structure is a two-way branch while the FORTH-85 DO-CASE END-CASES allows 65,000 different cases randomly arranged. (n+1 CASE may precede n CASE.) Each case can test on a 16-bit quantity. The CASE structure must begin with "DO-CASE" then "CASE...END-CASE" and finish with "END-CASES".

The test "CASE" structure screen gives an example of using CASE within FORTH-85. An unknown variable is passed to "MONITOR" for an n-way branch based on a match between the variable and one of the cases. "DO-CASE" places the variable passed on the stack into a location in memory. "41 CASE" will fetch the variable and compare it to 41. If it is 41, code between "41 CASE" and "END-CASE" will be executed (in this instance "ASSIGN" will be printed) and then a direct jump to "END-CASES". If the variable is not 41, then "41 CASE" will cause a jump to the next case, and so on.

The "n" for "CASE" need not be in-line code, nor absolute, nor known at compile time. During run-time "n" may be computed, fetched or otherwise placed on the stack just prior to executing case. The possibilities that exist for this, combined with 16-bit CASE testing and 65,000 possible cases AND (!) no restrictions on the amount or type of code between CASE... END-CASE, are immense. Some uses that occur to me immediately are monitor program executives, machine code disassemblers, text interpreters, and disk I/O drivers.

HOW IT WORKS (during compile)

The screen showing DO-CASE through END-CASES must be loaded into your system before they can be used. This screen will actually extend your FORTH compiler beyong having IF...ELSE... THEN, BEGIN...IF...ELSE...WHILE, DO... LOOP, AND BEGIN...END.

Line 1 extends the assembler to include JNC for use by the following code statements.

Line 2 contains code to set up the run-time variable "VCASE" for use by "DO-CASE" and "CASE".

Lines 3-9 code statements define the run-time behavior of "DO-CASE, CASE and END-CASE". While lines 11-15 define the compile-time behavior of the compiling words: "DO-CASE, CASE, END-CASE, and END-CASE".

During compile time "Do-CASE" assembles the code "DO-CASE", places "HERE" and Ø on the compile-time stack, and assembles (temporarily) a 16-bit Ø. "CASE" then compiles code "CASE" swaps the compile-time stack places "HERE" on the stack to locate "CASE" for "END-CASE" and temporarily assembles an 8-bit zero. "END-CASE" now has passed to it the location of "CASE" and "DO-CASE". "END-CASE" will pass "DO-CASE" location on to "END-CASES". "END-CASE" assembles code "END-CASE", places "HERE" on the compile-time stack for "END-CASES", computes "END-CASE" minus "CASE" and loads the result (assembles difference) at "CASE" +2. "END-CASE" further assembles a 16-bit zero temporarily for "END-CASES". "END-CASES" has passed to it, on the compile-time stack, all the locations of "END-CASE" and the beginning "DO-CASE". Since the number of cases is variable "DO-CASE" had put a zero on the compile-time stack to mark the bottom location the "BEGIN HERE SWAP! - DUP Ø = END" takes every "END-CASES" location and assembles "END-CASES" location there for direct forward jumps from "END-CASE" to "END-CASES".

The effect of all the compiling machination was to place 16-bit and 8-bit code for use by the address interpreter during execution. Reviewing the intrepreter during execution. Reviewing the interpreter: compiled colon definitions are simply strings of addresses of routines to execute. Therefore, the address string compiled for: "DO-CASE...CASE...END-CASE... END-CASES" is:

Fig 1

```
AB....CD....EF....(G)....
        where:

A = 16-bit address of code "DO-CASE"
B = 16-bit address (G)
C = 16-bit address of code "CASE"
D =  8-bit value of distance F+1
E = 16-bit address of code "END-CASE"
F = 16-bit address of (G) and
(G) is the location to continue execu-
    tion after case.
I = interpreter pointer
```

HOW IT WORKS (during execution)

The following will refer to A through (G) in Figure 1.

Before executing (address interpreting): A (DO-CASE) the case number to execute is placed on the run-time stack. Code DO-CASE is executed when A is encountered and it will pop the stack and store it at the memory location "VCASE". At this point there is no jump to make based on condition so I is incremented past B. B exists to provide a backward stub location for the compiling of the structure. Code between B and C is executed, and by the time C is encountered the case condition to test for is on the stack. C is the code "CASE" which will pop the stack and do a 16-bit compare to "VCASE". If there is no match code, "CASE" increments I (interp. pointer) by D (8-bit). Control would then pass to F+1. If there is a match, I is incremented by one and points to D+1. Code till E is executed. E will cause code "END-CASE" to execute and it will load I with F. Now I points to (G).

The code described here may be used for your personal use and experimentation only. Commercial users should write to the author.

EXAMPLE

```
 0 (  DO-CASE  CASE  END-CASE  END-CASES  )
 1 ASSEMBLER DEFINITIONS HEX : JNC D? END ;   FORTH DEFINITIONS
 2 0 VARIABLE VCASE
 3 CODE DO-CASE  H POP  ' VCASE SHLD  I INX  I INX  NEXT JMP
 4 CODE CASE  W POP  ' VCASE LHLD  L A MOV  W 1+ CMP
 5   0= NOT IF  I LDAX  I 1+ ADD  A I 1+ MOV  NEXT JNC
 6              I INR  NEXT JMP  THEN  H A MOV  W CMP
 7   0= NOT IF  I LDAX  I 1+ ADD  A I 1+ MOV  NEXT JNC
 8              I INR  NEXT JMP  THEN  I INX  NEXT JMP
 9 CODE END-CASE  I LDAX  A L MOV  I INX  I LDAX  A H MOV
10   H PUSH  I POP  NEXT JMP
11 : DO-CASE   \ DO-CASE  HERE  0  0 ,  ;  IMMEDIATE
12 : CASE   \ CASE  SWAP  HERE 0 C, !  IMMEDIATE
13 : END-CASE   \ END-CASE  HERE 0 , SWAP  HERE
14     OVER  - SWAP  C! ;  IMMEDIATE
15 : END-CASES   BEGIN HERE SWAP !  -DUP  0 = END ;  IMMEDIATE


 0 ( TEST ' CASE ' STRUCTURE )    BASE C@   HEX
 1
 2 : MONITOR   DO-CASE
 3    41 CASE  [ ASSIGN ]      END-CASE
 4    44 CASE  [ DISPLAY ]     END-CASE
 5    46 CASE  [ FILL ]        END-CASE
 6    47 CASE  [ GO ]          END-CASE
 7    49 CASE  [ INSERT ]      END-CASE
 8    53 CASE  [ SUBSTITUTE ]  END-CASE
 9             END-CASES  ;
10
11 : KEYBOARD  BEGIN  KEY 7F AND  DUP MONITOR 20 = END ;
12
13 BASE C!
14
15
```

Editors Note:

This article has been presented as a good example of the ease of addition of control structures to match application needs.

FORTH-85 is a UES software product derived from microforth (TM, FORTH, Inc.). Both these versions contain definition names at variance with fig-FORTH and FORTH-78. We present a reference table:

FORTH-85 microFORTH	fig-FORTH	FORTH-78
[	."	."
]	"	"
	COMPILE	None
I	IP	None
END	UNTIL or END	END
THEN	ENDIF or THEN	THEN

;S

ANOTHER GENERATION OF MISTAKES?

Roger L. Gulbranson
University of Illinois

Much has been said about how each generation of computers the large mainframes, the minis, and now the micros - has repeated the mistakes of the past generation. There have even been comments on upward-compatibility mistakes in going from one generation of microprocessor to its succeeding generation(s).[1] I would like to take this further by commenting on the latest generation of microprocessors, the 16-bit CPUs.

I was talking to an EE who tried to convince me that the yet-to-be released microprocessors are "so much better" than the existing ones because they have included all of the addressing modes in each instruction. Among other things, I am told, this reduces program size and makes the micro run faster, since its speed is directly related to the number of fetches it must do per instruction and the number of instructions used. As a concept, in toto, I can only reply, BUNK!

If one were to design a microprocessor stack computer, [2] it would be possible to incorporate an instruction set that has only one multi-addressing mode instruction, a "load effective address" instruction. Since this is perhaps a bit austere, it may be realistic to add appropriate load to stack, store from stack, conditional and unconditional branch, and subroutine call instructions to the group of "addressed" instructions. The remaining instruction set need not contain more than 128 (if even that many) zero-address instructions. This instruction set can be arranged so that all zero-address instructions are 8 bits long. This means that most of the time an instruction word will contain two instructions, decreasing the number of memory cycles per instruction. If, in addition, the stack is "cached" on chip, the number of memory cycles per instruction will drop considerably. And if this latter idea is properly extended to the instruction stream to create an "instruction stack," much like that on the CDC Cyber computer line or the Cray-1, the number of memory cycles can be reduced even further. This reduction of memory cycles should noticeably increase the speed of our hypothetical microprocessor.

Considering the impetus given to virtual stack machines by the Pascal P-code groups[3] and the Concurrent Pascal originators,[4] one wonders why these ideas have not been efficiently implemented in silicon. Must we wait for another generation?

[1] L. Armstrong, "16-bit Wave Gathering Speed," Electronics, Vol. 51, No. 4, Feb. 16, 1978, pp. 84-85.

[2] A good overview of stack machines can be found in the May 1977 issue of Computer.

[3] References to these groups can be found in Pascal News, a publication of the Pascal User's Group.

[4] Per Brinch Hansen, The Architecture of Concurrent Programs, Prentice-Hall Inc., Englewood Cliffs, N.J., 1977.

Reprinted from Computer, April, 1979

;s

INSTALLATION REPORTS

The distribution of fig-FORTH began on May 11, 1979 at the Fourth West Coast Computer Faire. The first installation to be brought up by a user occurred while the Faire was still running! Bob Steinhaus of Lawrence Livermore Lab got the 8080 listing on Saturday at the Faire. His wife read the hex code to him and he typed it in. By Sunday morning, he was running!

The next to run was Dwight Elvey of Santa Cruz. He organized five programmers on five Intel developmental systems. Each edited in 1/5 of the source code. They then merged the files, assembled at the listing origin and caught final editing errors by a byte-by-byte comparison. They then updated the I-O and re-origined. They were running in four evenings work (of five people).

The next to run was Dave Carlton of Ceres, CA. Dave brought up fig-FORTH on his TRS-80. He did fill his edit buffer and then crash, which cost re-typing 25 pages! Dave demonstrated his system to FIG on June 25. He had just interfaced to his floppy-disc and had the sample editor running.

John Forsberg of Maracaibo, Venezuela should be up and running with his Prolog 8080 with 32K RAM and casette, floppy and disk capability that he put together himself.

Frank D. Dougherty of Belvidere, IL has a IMD05 Ver 2.05 with 15 languages running. He says his AXion EX-801 printer works like a charm.

Many installations of the PDP-11 are operating. There is a short cut in this case. John James (the implementor) is also acting as a distributor. He provides a source diskette that will assemble and run under RT-11 and RSX-11M. Between 10 and 20 installations should be up by now. John has gone to great care to create a "portable" version for the various CPU variations. fig-FORTH is known to be running on Heathkit and DEC LSI-11's up through the DPD-11/60. Installations are known in California, Arizona, New York, and the Netherlands.

;s

MEETING NOTICES

LONDON, ENGLAND

FORMAL, a meeting to gather input for the future implementation of FORTH, will be held January 8, 9, and 10, 1980 at Imperial College, London, England. Attending from FIG will be: Kim Harris, Bill Ragsdale, Jon Spencer, Larry Forsley and probably 2 or 3 more. Look for a report in the next issue of FORTH DIMENSIONS.

NORTHERN CALIFORNIA

FIG monthly meetings will continue to be held the fourth Saturday of each month at the Special Events Room of the Liberty House department store in Hayward. Informal lunch at 12 noon at the store restaurant, followed by the 1 pm meeting. Directions: Southland Shopping Center, Highway 17 at Winton Avenue, Hayward, CA, Third floor, rear of the Liberty House. Dates: 1/26/80, 2/23/80, 3/22/80, etc. All welcome.

Send us notices of any meetings that you know about.

;s

LETTERS

Despite your warnings regarding Programma International's implementation of FORTH for the PET, I bought it just to have something to work with and get a feel for FORTH (or at least a FORTH-like system), and I have been having a ball with it (despite the limitations of this version), after adding 16K of RAM - I thought the thing would run in 8K but found out differently. The first VOCABULARY written for the PET was a DEFORTH routine to disassemble the dictionary. After DEFORTHing the latter dictionary entries and saving them on tape, I was able to truncate the dictionary to wipe out 10 unneeded words. This saved about 3 pages of memory.

Edward B. Beach
Arlington, VA

Editor...
People keep trying!

You will find enclosed a set of source listings for the 8080 nucleus of MSL. (Editor's note: Incorporated in fig-FORTH 8080 Assembly Listing, Version 1.1 see New Products.)

All my code routines are re-entrant. Not only that, but without exception, they use no more bytes or time than Cassady's routines. In fact, in many cases, you will note that my routines use fewer bytes and run faster. In some cases, such as multiply and divide, the improvement is enormous! viz

U* UNSIGNED MULTIPLY

4950 cycles	81 bytes	CASSADY
994 cycles	47 bytes	VILLWOCK

U/ UNSIGNED DIVIDE

30,600 cycles	203 bytes	CASSADY
2,495 cycles	76 bytes	VILLWOCK

I was well into the design and coding of MSL when I stumbled upon FIG and its efforts. Your documents (particularly the installation manual and James' work on the 11) have been most useful and have saved me considerable time in putting the finishing touches on MSL. I'd like to return the favor, so I hope that my 8080 routines will be useful to you.

Many thanks again to FIG for your excellent efforts.

R. D. Villwock
Pasadena, CA

MORE FROM GEORGE

Following are some observations on PASCAL and FORTH implementation details made upon reading the recent article in Dr. Dobb's Journal.

With the emergence of a low cost yet elaborate compiler for PASCAL at USCD consideration might be given to PASCAL as an alternative to FORTH. PASCAL is implemented with a self-compiling, virtual-machine language system offering transportability similar to FORTH, and is supplied by USCD with full source code. From the standpoint of sophisticated extensions to a system such as high-speed arithmetic processing hardware, converting from floppy-disks to hard disks, adding memory beyond directly addressable space, however, PASCAL may be much more difficult to work with. The system involved very large programs including a compiler, an interpreter, a run time

executive, and a debugger--all of which must be consistently modified in making extensions. The USCD authors may well turn out to be the only users able to efficiently make expansions themselves.

An important advantage seems present in PASCAL, however, which is dynamic storage allocation--the nested globals and locals environment, and FORTH does not seem to offer this. Several methods for accomplishing this in FORTH might therefore be noted. First, existing FORTH system words can be used to create a class of variables and constants whose code-address points to a new defining-word which returns, not the address or data in the variable or constant itself, but in a dynamically reserved area on the stack. The parameter field of the variable holds an address offset generated at compile time, relative to an environment pointer implemented as another variable whose parameter field is filled with the stack pointer value upon entry to a procedure, before shifting the SP to reserve the data area. The code addressed by the variable combines the offset with the environment pointer and returns either this address, or the data at that address on the stack.

The process of retrieving dynamically stored data would be slowest when implemented with regular FORTH language, so a second method is to provide machine language for the basic mechanisms involved just as FORTH does for its standard data handling. Assumed in both cases is a provision for compiling words (e.g. SVARIABLE—define a "stack" variable) which creates the related dictionary entries with the same ease as the regular VARIABLE and CONSTANT words do.

An interesting variant would be pointing the code-address of a variable at a modifiable jump-vector. Initially the vector points to code which reserves space for the variable on the stack, and is then switched to point the code retrieving the data. Thus storage is automatically allocated, not upon entry to a procedure, but precisely when a variable becomes used.

The principle of the above, creating new defining-words, could have other uses as well; for example, data for a variable could reside on disk and the code-address of its dictionary entry could point to a routine which retrieves it from disk. What is needed is an easy way to compile these structures. For instance, if we want to create a variable X, to reside on disk, we need another variable, Y, to hold the data, and an operator RETRIEVE; X is then compiled as the definition; Y RETRIEVE so every reference to X returns the value from disk. Having defined the "Y Retrieve" word we need an easy way to tell the compiler that X should be compiled as that kind of definition, without having to write it all out.

PASCAL implementation may be very well optimized, but FORTH code accomplishes even more in the way of code compaction because of its unique representation of operands as though they were operators in object code. This means that no separate operator for "load stack" to push an operand is needed in object code, saving space. This is a kind of software implementation of "tagged memory" on large scale machines. FORTH still requires 16 bits for each object code entry, but further compaction implies what must be a time consuming unpacking operation every time a virtual-machine instruction is executed. The same compaction in FORTH applies to procedure calls as well as to data; no separate "call" op code is needed as the procedure referencing operand is the call itself. Since each procedure

has within itself whatever housekeeping functions are involved with entry and exit, elaborate housekeeping need be performed only when necessary; e.g. locals can have fixed absolute addresses or be dynamically allocated as the case warrants. Have the hard-wired stack-machine designers missed something here?

PASCAL is also capable of making available at run time the power of the compiler for sophisticated interfacing of users to applications packages, by running an interpreter for PASCAL and calling precompiled object code procedures. The interpreter seems large, however, and the whole procedure cumbersome. A high quality APL interpreter will soon be available from MICROSOFT for this kind of appilcation, but APL has many problems running from its special keyboard to the difficulty of modifying it. FORTH thus offers unique advantages in this area.

George B. Lyons
Jersey City, NJ

"The 'TO' Solution" in issue #4 improves the handling of variables by increasing the amount of interpretation done at run-time in FORTH (a conditional branch on the value of the store/retrieve state variable). Perhaps this could also be accomplished instead at compile time. Let each variable contain two code addresses, one for retrieval, one for storing, and let the compiler select which to put into the object code generated on the basis of the state variable. "TO" as such would then not appear in the object code at all, saving space, but variables would become longer. But again, no assignment operators at all again, no assignment operators at all would appear in object code, either. In order for the compiler to work this way it would have to be expanded beyond the standard FORTH by adding the capability of distinguishing different types of words and using different procedures to compile variables and non-variables. This might be accomplished by adding to the code for each type of word, e.g. variables, a pointer to a compiler routine to be used when compiling words of each type. When encountering a word in the input stream, it would be looked up in the dictionary, the code address extracted, and then the pointer to the compiler code located within the code; compilation would then continue where pointed. The compile code for variables would check the state variable and enter the appropriate address from the word being compiled. Code for variables would have to make allowance for the varying distance of the parameter field from the code address.

This method might be a technique for implementing compilers of all sorts derived from the FORTH compiler.

In the case of ARRAYS, one might consider having several code addresses in each array in the dictionary, with associated operators in the source code which do not get put into the object code, but merely select which of the code addresses will be compiled for each array reference in source.

Of course, if you expand the compiler, it might get too big to fit in memory with all the application code the way FORTH normally does.

George G. Lyons
Jersey City, NJ

;s

NEW PRODUCTS

REVISED fig-FORTH LISTING

fig-FORTH 8080 Assembly Listing has been revised to Version 1.1. If you have an old version (1.0), send in the original cover and $4.00 for the new version (1.1). Otherwise available for $10.00 (overseas $13.00. Forth Interest Group, P.O. Box 1105, San Carlos, CA 94070.

PDP-11 fig-FORTH DISKETTE

In addition to the fig-FORTH system, the diskette includes an editor, Forth assembler, string package and an 80-page User's Guide with discussion and examples of Forth programming techniques.

Like all the fig-FORTH systems, this one has full length names to 31 characters, the "security package" of compile-time error checks, and alignment with the 1978 Forth International Standard.

This system runs under the RT-11 or RSX-11M operating systems, and can be modified for other environments or for stand-alone It can interface to database packages or other software, allowing interactive access and program development with systems not otherwise available interactively. The fig-FORTH model is distributed in Macro-11 source, for easy modifiability by programmers without a Forth background; the editor, assembler, and string package are in Forth source.

The complete system price is $130, including diskette and all documentation; the User's Guide separately is $20. (Ca. residents add 6% tax.) John S. James, P.O. Box 348, Berkeley, CA 94701.

FORTH FOR THE TRS-80

MMSFORTH offers TRS-80 users stack-oriented logic and structured programming, machine-code speed and compactness, virtual memory, major advantages of interpreter, compiler and assembler (all are co-resident), and your own commands in its extensible dictionary. MMSFORTH includes assembler/ editor, all necessary routines for disk and/or tape, and additional routines for BASIC-like handling of strings, arrays, etc. MMS supplies information and examples to learn this language or to start your own FORTH program for your specialized application.

SPECIAL - At no extra cost, THE GAME OF LIFE in MMSFORTH just 2 seconds per generation, an excellent demonstration of the techniques and speed of FORTH. MMSFORTH, without manual:

L.2 16K tape	$35.00
Disk, w/Disk I/O	$45.00
"The microFORTH PRIMER", manual for MMSFORTH	$15.00

Miller, Microcomputer Services, 61 Lake Shore Road. Natick, MA 01760

SBC-FORTH

SBC-FORTH is a complete firmware operating system designed to plug directly into the ROM sockets of the SBC-80 series of single board computers offered by Intel & National. Operating entirely within the resources of one SBC Card SBC-FORTH features resident compiling and assembling of user entered tasks. The addition of SBC Disk and RAM Cards to the system allows SBC-FORTH's resident disk I/O and Text Editor to edit, load, and run source programs from disk. SBC-FORTH is interactive with the user's CRT and produces tight object code capable of

execution speeds approaching assembler code. SBC-FORTH is presently available on four 2716 EPROMS for the SBC-80/20 and two 2732 EPROMS for the SBC-80/30 CPUs. Options include Single or Double Density SBC Disk I/O Drivers anad CRT Console Baud Rates. Priced from $750 depending on media. Zendex Corporation, 6398 Dougherty Road, Dublin, CA 94566.

STOIC-II

STOIC-II is an enhanced version of STOIC, a FORTH dialect which originated at MIT's Biomedical Engineering Center. STOIC offers significant advantages over other microprocessor FORTH implementations, notably a comprehensive disk file system and text editor in place of the "screen" ap proach used elsewhere. This encourages good program docu mentation by allowing arbitrarily long lines and pages, thus making it easier to include adequate comments and indenta tion. Other features include syntax checking and a separate stack for loop control, which together yield a system that recovers from most errors instead of crashing. STOIC also provides for character-string literals and offers a very extensive set of standard words, including 16 and 32 bit arithmetic.

The standard STOIC-II package includes the kernel and basic words, assembler, file system, editor, double-precision and floating-point arithmetic, target compiler and associated library of primities, and utility programs for paginated file listing and alpabetized listing of vocabulary branches. The version currently distributed runs on 8080, 8085, and Z-80 based computers with soft-sectored floppy disks, and is priced at $4000 with complete source code or $2000 for object code only. Avocet Systems, 804 South State Street, Dover, Delaware 19901.

FORTH, Inc. NEWS

A new on-line monitoring and record-keeping system developed by FORTH, Inc., using a PDP 11/60 minicomputer, is being used by the Department of Pulmonary Medicine of Cedars-Sinai Medical Center, Los Angeles. The FORTH system collects data from five on-line sources, maintains a data base including this information and performs calculations for and formats a wide variety of reports. Data is collected from patient admittance questionnaires and intensive care patient records through terminals; from the automated blood gas laboratory; the pulmonary function test equipment; and the exercise laboratory, all through direct links with the PDP 11/60. The speed of FORTH allows the Cedars-Sinai pulmonary specialists to examine all data on a particular patient, while the patient is still in the laboratory and connected to the equipment.

Current Openings

Project Manager - applications and special systems

Product Manager - processors and drivers

Programmer/Engineer - hardware and software projects

Technical Writer - software documentation

FORTH, Inc.
815D Manhattan Avenue
Manhattan Beach, CA 90266
