FORTH DIMENSIONS

AUGUST/SEPTEMBER 1978 VOLUME 1 NO. 2

CONTENTS

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

HISTORICAL PERSPECTIVE

FORTH was created by Mr. Charles H. Moore in about 1969 at the National Radio Astronomy Observatory, Charlottesville, VA. It was created out of his dissatisfaction with available programming tools, especially for automation. Distribution of his work to other observatories has made FORTH the de-facto standard language 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.

FORTH enjoys a synergism of its features. It has none of the elephantine characteristics of PL/1 or FORTRAN. It has a density and speed far surpassing BASIC, but retains an interactive nature during program development. Since it is extensible, special words are easily defined to give it the terseness of APL. Its clarity and consistency result from being the product of a single mind. (as were APL and PASCAL).

Although the language specification and many implementations are in the public domain, many other implementations and application packages are available as program products of commercial suppliers.

The FORTH Interest Group is centered in Northern California. It was formed in 1978 by local FORTH programmers to encourage use of the language by the interchange of ideas though seminars and publications. About 300 members are presently associated into a loose national organization. ('Loose' means that no budget exists to support any formal effort.) All effort is on a volunteer basis and the group is associated with no vendors.

;S W.F.R 8/20/78

FOR NEWCOMERS

FORTH listings consist of sequences of "words" that execute and/or compile. When you have studied a glossary and a few sample listings, you should develop the ability to understand the action of new words in terms of their definition components. For the time being, we present a simplified glossary of the undefined words in this issue of FORTH DIMENSTIONS. For a fuller listing send for the F.I.G Glossary.

: xxx ..... ;
':' creates a new word named 'xxx' and compiles the following words (represented at ....) until reaching ';'. When 'xxx' is later used, it executes the words right after its name until the ';'.

CONSTANT VARIABLE
Each creates a new word with the following name, which takes its value from the number just before.

IF ELSE THEN
A test is made at 'IF'. If true, the words execute until the ' ELSE' and skip until THEN. If false, skip until ELSE and execute until THEN.

BEGIN END
At END a test is made; if false, execution returns to BEGIN; otherwise continue ahead.

DO LOOP LEAVE
At DO a limit and first index are saved. At LOOP, the index is incremented; until the limit is reached, execution returns to DO. LEAVE forces execution to exit at LOOP.

DUP DROP OVER SWAP ROT + - * /
These words operate on numbers in a stack just as then do in a HP calculator. If you like HP, you'll love FORTH.

>R R>
>R moves the top stack number to another stack. R> retreives it back to the original stack.

@ C@
@ Fetches the 16 bit contents of an address. C@ does the same for a byte. C@ may be also called B@ or \@.

! C!
These words store the second stack number at the memory address on the top of the stack. C! stores only a byte; it may be named B! or \! on some systems.

TYPE types a string by memory address and character count.

SPACE types a space.

CR types a carriage return/line feed.

MOVE moves within memory by addresses and byte count.

. prints a number.

.R prints a number in a tabulated column.

2+ adds two to the stack top number.

STATE is a variable, true when compiling.

(skips over comments until finding a ')'.

EXECUTE executes the word whose address is on the stack.

' finds the address of the next input name.

AND is a bitwise logical and.

, places a number in memory as part of compiling.

= > < are logical comparisons of stack numbers.

20 WORD fetches the next input word string.

HERE is a temporary memory workspace.

IMMEDIATE <BUILDS DOES> are too involved to discuss here. They are described in some detail in the text.

EDITORIAL

FORTH DIMENSIONS is dedicated to the promotion of extensible, threaded languages, primarily FORTH. Currently we are seeing a proliferation of similar languages. We will review all such implementations, referring to sources and availability.

Our policy is to use the developing "FORTH 77" International Standard as our benchmark.

Variant languages, such as STOIC, URTH, and CONVERS, will be evaluated on their advantages and disadvantages relative to FORTH. However, in evaluating languages named FORTH, we will note their accuracy in implementing all FORTH features. We expect complete versions named FORTH to contain:

1. indirect threaded code
2. an inner and outer interpreter
3. standard names for the 40 major primitives
4. words such as ;CODE, BLOCK, DOES>, (or ;:), which allow increased performance.

We hope to enable prospective users/purchasers to correctly select the version and performance level they wish, to foster long-range growth in the application of FORTH.

W.F.R.

CONTRIBUTED MATERIAL

FORTH Interest Groups needs the following material :

1. Technical material for inclusion in FORTH DIMENSIONS. Both expositions on internal features of FORTH and application programs are appreciated.

2. Name and address of FORTH implementations for inclusion in our publications. Include computer requirements, documentation and cost.

3. Manuals available for distribution. We can purchase copies and distribute, or print from your authorized original.

4. Letters of general interest for publication in this newsletter.

5. Users who may be referenced for local demonstration to newcomers.

EXTENSIBILITY WITH FORTH

The purpose of any computer language (and its compiler or interpreter) is to bridge the gap between the "language" the machine understands (low level) and a language people understand (high level programming language). There are many choices for human-understandable languages: natural languages and artificial languages. The choice of language should allow convienient, terse, and unambiguous specification of the problem to be solved by the computer. Ordinarily only a few computer languages are available (e.g. BASIC, FORTRAN, APL). These were designed for certain classes of problems (such as mathematical equations) but are not suitable for others. The level of a language is a measure of suitability of that language for a particular application. The higher the level, the terser the program. By definition, [1] the highest level would allow a given problem to be solved with one operator (or command) and as many operands as there are input data required.

A natural language (e.g. English) might appear to be the best choice for a human-understandable computer language, and for some applications it may be. But natural languages suffer from three limitations: verbosity, ambiguity, and difficulty to decipher. This is partly because the meaning of a given word is dependent on its useage in one or more sentences (called "context sensitive") and because they require complex and nonuniform grammar roles with many exceptions. Specialized vocabularies and grammars permit terse and precise expression of concepts for restricted sets of problems. For example, [2] consider the following definition of a syllogism from propositional calculus:

$$((P1 \supset P2) \supset ((P2 \supset P3) \supset (P1 \supset P3)))$$

This sentence may be translated into English as "Given three statements which are true or false, if the truth of the first implies the truth of the second, this implies that if the truth of the second implies the truth of the third, then the truth of the first implies the truth of the third." Ambiguity is hard to avoid in most natural languages. The English phrase "pretty little girls school" (when unpunctuated) has 17 possible interpretations! (Try it.) [3]

As for the suitability of traditional programming languages (e.g. BASIC, FORTRAN, COBOL, PASCAL, APL) for "almost all technical problems", try coding the following "sentences" in your favorite computer language:

Quantum Mechanics:

$$H\psi = E\psi$$
where $H = -(\hbar^2/2m)\nabla^2 + V$
and E is the energy of the system

Electricity and Magnetism:

$$\nabla \cdot D = \rho$$
$$\nabla \cdot B = 0$$
$$\nabla \times E = -\frac{\partial B}{\partial t}$$
$$\nabla \times H = i + \frac{\partial D}{\partial t}$$

Matrix Algebra:

The trace of a matrix is equal to the sum of its eigenvalues.

Organic Chemistry:

$$3CH_3CH_2OH + Cr_2O_7^{--} + 8H^+ \rightarrow 3CH_3CHO(g) + 2Cr^{+++} + 7H_2O$$

Knitting:

K2 tog.3(5) times, *k1, p2, k1, k3, tog., k1,
p2, k1, p1, k1, p3 tog., k1, p1, p1* ; repeat
between *'s once more, k1, p2, k1, k3 tog.,
k1, p2, k1; k2 tog. 3(5) times; 47(51) sts.

Poetry:

Shakespearean sonnets are in iambic pentameter and
consist of three quatrains followed by a couplet.

FORTH is capable of being matched to each of the above relations at a high level. Furthermore, using the FORTH concept of vocabularies, several different applications can be resident simultaneously but the scope of reference of component words can be restricted (i.e., not global). This versatility is because the FORTH language is extensible. In fact the normal act of programming in FORTH (i.e., defining new words in terms of existing words) extends the language! For each problem programmed in FORTH, the language is extended as required by the special needs of that problem. The final word defined which solves the whole problem is both an operator within the FORTH language (which is also a "command") and the highest level operator for that problem. Further, the lower level words defined for this problem will frequently be useable for the programming of related problems.

It is true that popular computer languages allow new functions to be added using SUBROUTINES and FORTRAN-like FUNCTIONS. However these cannot be used syntactically the same as the operators in the language.

For example, let's assume a BASIC interpreter does not have the logical AND operator. To be consistent with similar, existing operators one would like to use "AND" in the following syntax:

A AND B

(A,B are any valid operands). Furthermore, one would want this new operator to have a priority higher than "OR" and lower than "NOT" so that

NOT A AND B OR C

would mean

(NOT A) AND (B OR C)

The only way to do this is to modify one's BASIC interpreter. Although not impossible, this is usually very difficult because of the following reasons:

(1) Most BASIC's are distributed without the interpreter source code (or not in machine-readable form). (2) One would have to learn this program and design a change. One modification might be "patched in", but many probably could not. Such a modified interpreter might not be compatible with future releases from the manufacturer. Errors might be introduced into other parts of the interpreter by this modification. (3) The process of changing is time consuming. Before one could try the new version, one would have to assemble, link, load, and possibly write a PROM. (How long would this take on your system?)

Another alternative would be to use a BASIC FUNCTION to add the AND operator, but it would have to be referenced as:

AND (A,B)

If NOT and OR were also FUNCTIONS, (NOT A) AND (B OR C) would have to be written as:

AND (NOT(A), OR(B,C))

This later form is lower level, less readable, and inconsistent with the intrinsic operators.

The addition of an AND operator in FORTH is as simple as any other programming addition; it would require one line of source code. The FORTH assembler could be used to take advantage of a particular processor's instruction, or the compiler could be used, resulting in machine transportability! Other differences from the example of modifying BASIC are: (1) Nothing existing in the FORTH system needs to be changed, so no learning is required, no errors are introduced to the existing system, and compatibility with future releases is preserved. The source of the FORTH kernel is not even necessary. (2) The new operator can be tried immediately after it is defined; if it is wrong it can be fixed before any further use is made of it. (3) This new operator is used exactly the same as any FORTH operator. So it may be mixed with existing operators in a totally consistant manner.

APL fans will point out that all the above is true for APL also. For something as simple as "AND" there is little difference. However, APL allows only monadic (single operand) and diadic (two operand) operators; FORTH operators can be written to accept as many operands as the programmer desires.

The previous discussion addressed the extension of the FORTH language, but it is almost as easy to extend the FORTH compiler! New compiler control structures (e.g., the CASE construct) can be added without changing any of the existing compiler. Or the existing compiler can be modified to do something different. To maintain compatibility with the existing compiler, the modifications could be part of a user- defined "vocabulary" so that both versions would be selectively available. Furthermore, one can write an entirely new compiler which accepts either the FORTH language or another language. (Complete BASIC interpreters have been written in FORTH.)

The choice of a computer language for a given application (including system development) should optimize the following attributes: (1) Be terse (i.e., the highest level for the application) (2) Be unambiguous (3) Be extensible (e.g., language, data types, compiler) (4) Be efficient (5) Be understandable (e.g., self documentation) (6) Be correct (e.g.,testing, proving assertions, consistency checks) (7) Be structured (e.g., structured programming, reentrant, recursive) (8) Be maintainable (e.g., modular, no side effects)

FORTH is a compromise among these goals, but comes closer than most existing programming languages.

;S KIM HARRIS

REFERENCES

1 Halstead, Maurice, Language Level: A Missing Concept in Information Theory, Performance Evaluation Review, ACM SIGMETRICS, Vol. 2 March '73.

2 McKeeman, Horning, and Wortman, A Compiler Generator, Prentice-Hall, 1970.

3 Brown, James Cooke, Loglan 1: A Logical Language, Loglan Institude, 2261 Soledad Rancho Road, San Diego, CA (714) 270-9773

GERMAN REVISITED

In the last issue of FORTH DIMENSIONS we showed how to create a bi-lingual (or multi-lingual) version of FORTH, and listed a simple program (set of FORTH definitions) for doing so. In respect to translation, there are three different classes of FORTH words:

(A) Those such as mathematical symbols which don't need to be translated.

(B) Words such as DO and IF which cannot be translated by a simple colon definition; the existing definitions must be re-copied and given German names. (all the definitions are short - one line - however)

(C) Other words, which could either be re-copied, or re-defined by a colon definition.

In any case, separate vocabularies can be used to prevent spelling clashes, no matter how many languages are spoken by one FORTH system. It can be possible to change languages as much as desired, even in the middle of a line.

The article stated that there was no run-time overhead. Such performance is possible, but the example given does have a run-time overhead of one extra level of nesting for each use of a word translated by a colon definition.

The following article by Bill Ragsdale is a more advanced treatment of language translation methods. It is written at the level of the FORTH systems programmer, and it uses a more standard FORTH version than the DECUS-supplied version which was used in the article which appeared in FORTH DIMENSIONS 1.

JOHN S. JAMES

FORTH LEARNS GERMAN

In the last issue of FORTH DIMENSIONS, we featured an article on natural language name *conversions for* FORTH. This article will add some additional ideas on the same topic.

First, the method shown (vectoring thru code) does have some run-time overhead. Also, some code definitions cannot execute properly when vectored in this manner, for example:

```
: R>   R>  ;
```

will pull the call of R> from the return stack and crash. We would ultimately like to translate names with:

1. Precisely correct operation during execution and compiling.

2. A minimum of memory cost.

3. A minimum of run-time cost.

4. A minimum of compile-time cost.

Let us now look at three specific examples to further clarify some of the trade-offs involved.

EXAMPLE 1 - COMPILING WORD

Let us see how UBER can be created to self-compile.

```
HEX
: D-E >R 2+ @ (2+ optional on some systems)
 STATE @ IF  (compiling) DUP 8 - C@ 80 <
  IF 2 - , ELSE (immediate ) EXECUTE THEN
 ELSE EXECUTE
  THEN ;
 : DO ENGLISH EMPLACE D-E ' , IMMEDIATE ;
   IMMEDIATE
 : UBER DO.ENGLISH OVER ;
 : LADEN DO.ENGLISH LOAD ; etc.
```

When building the translation vocabulary, the colon ':' creates the word UBER and then executes the immediate word DO.ENGLISH. DO.ENGLISH first emplaces the run-time procedure 'D-E' and then uses " ' , " to emplace the parameter field address of the next source word (OVER). Finally, the new word (UBER) is marked immediate, so that it will execute whenever later encountered.

Now we see how UBER executes. When it is interpreted from the terminal keyboard, 'D-E' will execute to fetch the emplaced PFA within the definitions of UBER (by R> 2+ @). After checking STATE the ELSE part will execute OVER from its parameter field address.

When UBER is encountered by the compiler in a colon definition, it will execute, as do all compiling words. Again R> 2+ @ will fetch the PFA of OVER to the stack. The check of STATE will be true and DUP 8 - C@ will fetch the byte containing the precedence bit. When compared to hex 80, a true will result for non-immediate, and 2 - , will compile the code field address.

However, if the word had been immediate (OVER isn't) the ELSE part will execute the word as in any compiling word.

The space cost of example 2 is 14 bytes per word (8 bytes per header and 6 bytes in the parameter field). The compile time cost is the execution of 'D-E.' There is no ultimate run-time cost in compiled definitions.

EXAMPLE TWO - <BUILDS - DOES>

Another way is to define a 'BUILDS-DOES' word E>G (English to German). It is then used to build a set of translation words similar to a FORTH mnemonic assembler.

```
: E>G   <BUILDS  ' , IMMEDIATE
        DOES>  @  STATE  @
        IF (compiling )  DUP 8 - C@ 80 <
            IF 2 - , ELSE EXECUTE THEN
          ELSE  EXECUTE  THEN  ;

E>G  UBER  OVER
E>G  LADEN  LOAD
E>G  BASIS  BASE     ..... etc.
```

E>G is a defining word that builds each German word (UBER) and emplaces the parameter field address of the English word (OVER) into the new parameter field (of UBER), and finally makes UBER immediate. When UBER is encountered by the outer interpreter, it does the DOES> part. The parameter of UBER, (the PFA of OVER) will be fetched and STATE tested. Since executing, the ELSE part will execute OVER from its parameter field address (as in the example 1).

When compiling, the DOES> part will be executed, again similarly to 'D-E' as in Example 1. The space cost of the BUILDS-DOES method is 10 bytes per word (8 in the header, 2 in the parameter field). The compile time is the same as in Example 1 and there is no run-time cost.

EXAMPLE THREE - RENAME

This last method is the most fool-proof of all. We will just re-label the name field of each resident word to the German equivalent.

```
: RENAME  ' 8 - DUP C@ 60
   AND (precedence bit  )
   20 WORD HERE C@ +
   HERE C! (store into length)
   HERE SWAP 4 MOVE ;

     ( overlay old name )

   RENAME OVER UBER

   RENAME LOAD LADED

   RENAME BASE BASIS
```

This method extracts the precedence bit of the old (English) definition and adds it to the length count of the new (German) name. The new name is then overwritten to the old name field. There is no space or time cost!! The dictionary is now truely translated.

A final caution is in order for Examples 1 and 2. Some FORTH methods may still give trouble. If you should try:

```
UBER
```

you will find the PFA of UBER which is a translating definition, and not the ultimate run-time procedure, (which is really in OVER). This would have disasterous results if you were attempting to alter what you thought was the executing procedure, and you were really altering the compiling word. For this reason, the method of Example 3 is the only truly 'fool-proof' method. The renaming method has the added use of allowing you to change names in your running system. For example, it is likely that the old <R will be renamed >R in the international standard FORTH-77. You can simply update your system by the use of the word RENAME.

;S W.F. RAGSDALE 8/27/78

THREADED CODE

Bell (1) and Dewar (2) have described the concepts of Threaded Code (also called Direct Threaded Code, or DTC), and an improvement called Indirect Threaded Code, or ITC. DTC was used to implement Fortran IV for the PDP-11, and ITC was used for a machine- independent version of Spitbol (a fast form of the string-processing language Snobol). Forth is a form of ITC, but different from the scheme presented in (2).

In DTC, a program consists of a list of addresses of routines. DTC is fast; in fact, only a single PDP-11 instruction execution is required to link from one routine to the next (the instruction is 'JMP @(R)+', where 'R' is one of the general registers). Overall, DTC was found to be about three percent slower than straight code using frequent subroutine jumps and returns, and to require 10-20 percent less memory. But one problem is that for each variable the compiler had to generate two short routines to push and pop that variable on the internal run-time stack.

In ITC, a program is a list of addresses of addresses of routines to be executed. As used in (2), each variable had pointers to push and pop routines, followed by its value. The major advantage over DTC is that the compiler does not have to generate separate push and pop routines for each variable; instead these were standard library routines. The compiler did not generate any routines, only addresses, so it was more machine independent. In practice, ITC was found to run faster than DTC despite the extra level of indirection. It also used less memory.

Forth is a form of ITC, with additional features.

Forth operations are lists of addresses pointing into dictionary entries. Each dictionary entry contains:

(A) Ascii operation name, length of the name, and precedence bit; these are used only at compile time and will not be discussed further.

(B) A link pointer to the previous dictionary entry. (This is used only at compile time.)

(C) A pointer called the code address, which always points to executable machine code.

(D) A parameter field, which can contain machine instructions, or Forth address lists, or variable values or pointers or other information depending on the variable type.

By the way, virtually everything in Forth is part of a dictionary entry: the compiler, the run-time routines, the operating system, and your programs. In most versions, only a few bytes of code are outside of the dictionary.

The code address is crucial; this is the 'indirect' part of ITC. Every dictionary entry contains exactly one code address. If the dictionary entry is for a "primitive" (one of the 40 or so operations defined in machine language), the code address points two bytes beyond itself, to the parameter field, which contains the machine-language routine.

If the dictionary entry is for a Forth higher-level operation (a colon definition), the code address points to a special "code routine" for colon definitions. This short routine (e.g. 3 PDP-11 instructions) nests one level of Forth execution, pushing the current 'I' register (the Forth "instruction counter") onto a return-address stack, then beginning Forth execution of the address-list in the new operation's parameter field.

If the dictionary entry is for a variable, then the code address points to a code routine unique to that variable's type. The parameter field of a variable may contain the variable's value - or pointers if re-entrant, pure-code Forth is desired.

Results of Forth-type ITC include:

(A) Execution is fast, e.g. two PDP-11 instruction executions to transfer between primitives, about ten to nest and un-nest a higher-level definition. (Because of the pyramidal tree-structure of execution, the higher-level nesting is done less often.) Yet the language is fully interactive.

(B) Forth operation names (addresses) are used exactly the same regardless of whether they represent primitives or higher-level definitions (nested to any depth). Not even the compiler knows the difference. In case run-speed optimization is desired, critical higher-level operations (such as inner loops) can be re-coded as primitives, running at full machine speed, and nothing else need by changed.

(C) Forth code is very compact. The language implements an entire operating system which can run stand-alone, including the Forth compiler, optional assembler, editor, and run-time system, in about 6k bytes. (Forth can also run as a task under a conventional operating system, which sees

it as an ordinary assembly-language program, and Forth can link to other languages this way.) Code is so compact that application-oriented utility routines can be left in the system permanently, where they are immediately available either as keyboard commands or instructions in programs, and they are used in exactly the same way in either case. No linkage editing is needed, and overlays are unusual.

REFERENCES

(1) Bell, James R. Threaded code. C. ACM 16, 6 (June 1973), 370-372.
(2) Dewar, Robert B. K. Indirect threaded code. C. ACM 18, 6 (June 1975), 330-331.

STAFF

The volunteer staffing of FORTH DIMENSIONS is a bit fluid. For this issue, our staff consisted of :

EDITOR	JOHN JAMES
CONTRIBUTORS	KIM HARRIS W.F.RAGSDALE
TYPESETTING	TOM OLSEN JOHN JAMES
ARTWORK	ANNE RAGSDALE
CIRCULATION	DAVE BENGEL
DATA PROCESSING	P D P -11

FORTH DEFINITION

FORTH is the combination of an extensible programming language and interactive operating system. It forms a consistent and complete programming environment which is then extended for each application situation.

FORTH is structured to be interpreted from indirect, threaded code. This code consists of sequences of machine independent compiled parameters, each headed by a pointer to executable machine code. The user creates his own application procedures (called 'words'), from any of the existing words and/or machine assembly language. New classes of data structures or procedures may be created; these have associated interpretive aids defined in either machine code or high level form.

The user has access to a computation stack with reverse Polish conventions. Another stack is available, usually for execution control. In an interactive environment, each word contains a symbolic identifier aiding text interpretation. The user may execute or compile source text from the terminal keyboard or mass storage device. Resident words are provided for editing and accessing the data stored on mass storage devices (disk, tape).

In applications that are to run 'stand-alone', a compact cross-compiled form is used. It consists of compiled words, interpretive aids, and machine code procedures. It is non-extensible, as the symbolic identifiers are deleted from each word, and little of the usual operating system need be included.

;S W.F.R. 8/26/78

NOTES

- The second meeting of the FORTH International Standards Team will occur in Los Angeles on October 16-19. Contact FORTH Inc. for additional information.

- A partially micro-coded FORTH-like language is described in "Threaded Code for Laboratory Computers" by J.B. Phillips, M.F. Burke, and G.S. Wilson, Dept. of Chemistry, University of Arizona, Tucson, AZ 85721. The article is published in Software - Practice and Experience, Volume 8, pages 257-263. Implementation is on a HP2100. The article also describes the advantages of threaded languages for laboratory applications.

- A "form" of FORTH for the Apple and PET 6502 based computers is available from Programma Consultants, 3400 Wilshire Blvd., Los Angeles, CA 90010. We have not used these enough to review them for this issue but they have been shipped and do work. For more information write to Programma Consultants or watch future issues of FORTH DIMENSIONS.

- FORTH Inc. is looking for a programmer with some systems-level experience using FORTH or similar languages. Interested persons should contact FORTH Inc., 815 Manhattan Avenue, Manhattan Beach, California 90266, (213) 372-8493 .

HELP

SCR # 6

```
 0     THE 'HELP' COMMAND IS PROBABLY THE MOST USEFUL OPTION FOR
 1 A FORTH SYSTEM.  IT ALLOWS YOU TO VIEW THE DICTIONARY WORDS
 2 AND LOCATE THEM IN MEMORY.  WHEN YOU ARE TESTING NEW
 3 DEFINITIONS, IT WILL SHOW RE-DEFINITIONS.  IT IS A WAY TO
 4 LOCATE WHERE A MISSING WORD SHOULD BE, BUT ISN'T.
 5
 6     IF YOU MAKE A COMPILE ERROR FROM DISC, 'HELP' WILL SHOW
 7 THE WORD IN WHICH THE ERROR OCCURED.
 8
 9     YOU SHOULD MODIFY THE FOLLOWING DEFINITIONS TO THE FORMAT
10 YOU WANT.  FOR OBJECT CODE EXAMINATION, I LIKE THE CODE FIELD
11 ADDRESSES AS SHOWN, SINCE THIS IS WHAT RESULTS IN THE COMPILED
12 CODE.   FOR A QUICK SNAP-SHOT OF THE DICTIONARY, I JUST PRINT
13 THE LENGTH AND NAMES.
14
15     JUST TYPE 'HELP' AND HIT THE 'BREAK' KEY TO STOP.
```

SCR # 7

```
 0 (  HELP  )                HEX
 1 00      CONSTANT  LAST.LINK    ( IS $8000 ON MICRO-FORTH )
 2  4      CONSTANT  #/LINE       ( WORDS PRINTED PER LINE  )
 3
 4 : .NAME              ( ENTER WITH ADDRESS OF LENGTH BYTE )
 5    DUP  C@  7F  AND  DECIMAL  3  .R  SPACE  1+  3  TYPE SPACE  ;
 6
 7 : .CODE-ADDRESS      ( ENTER WITH ADDRESS OF LENGTH BYTE )
 8     6  +  HEX  5  .R  SPACE  ;
 9
10 : .HEADER            ( ENTER WITH ADDRESS OF LENGTH BYTE )
11     DUP  .NAME  .CODE-ADDRESS  ;
12
13 : ?TERMINAL  0  ;  ( USER'S MACHINE DEPENDENT TERMINAL BREAK )
14              ( RETURN '00' FOR NO BREAK, AND '01' FOR A BREAK )
15 8 LOAD      ;S    8/27/78    WFR
```

SCR # 8

```
 0 (  HELP, CONT.  )
 1
 2 : .LINE           ( PRINT A LINE OF NAMES AND CODE ADDRESSES )
 3    #/LINE  0      ( ENTER WITH ADDRESS OF LENGTH BYTE        )
 4    DO  DUP  .HEADER  SPACE    4  +  @  DUP  LAST.LINK  =
 5        IF  LEAVE  THEN  LOOP  ;     ( EXIT WITH NEXT ADDRESS )
 6
 7 : HELP          ( PRINT DICTIONARY FROM TOP CURRENT WORD DOWN )
 8           ( TO BOTTOM.  FORMAT IS LENGTH COUNT, 3 LETTERS OF )
 9           ( NAME, AND CODE FIELD ADDRESS.  WILL TERMINATE    )
10           ( UPON LAST LINK VALUE OR A TERMINAL BREAK.        )
11    BASE  C@  >R    CURRENT  @  @
12    BEGIN  CR  .LINE  DUP  LAST.LINK  =  ?TERMINAL  +
13       END   DROP ( LAST LINK )  R>  BASE  C!  ;
14
15 DECIMAL     ;S     8/28/78     WFR
```

```
HELP
  4 HEL  1EBB    5 .LI  1E90    7 .HE  1E80   13 .CO  1E68
  5 .NA  1E43    6 #/L  1E39    9 LAS  1E2F    4 TAS  1E25
  4 BOO  1B8B    2 ->   1B6A    4 TUB  1B57    3 TTY  1B44  OK
```

ABOVE WE SEE AN EXAMPLE OF THE LOADING OF THE 'HELP' COMMAND FROM DISC. IT THEN IS TESTED, AND DUMPS THE DICTIONARY. WE SEE THE LISTING OF 'HELP' AND THE WORDS IS USES LISTING CONTINUES INTO THE RESIDENT DICTIONARY.

GOOD LUCK, WFR

MANUALS

DECUS PDP-11 FORTH
by Owens Valley Radio Observatory, California Institute of Technology, Martin S. Ewing. (alias The Caltech FORTH Manual) Available from DECUS, 129 Parker Street, PK3/E55, Maynard, Mass. 01754. Ordering information: Program No. 11-232, Write-up, $5.00 .

FORTH Systems Reference Manual
W. Richard Stevens, Sep 76. Kitt Peak National Observatory, Tucson, AZ 85726. (NOT FOR SALE)

LABFORTH
An Interactive Language for Laboratory Computing, Introductory Principles, Laboratory Software Systems, Inc., 3634 Mandeville Canyon, Los Angeles, CA 90049. $8.00 .

STOIC
(Stack Oriented Interactive Compiler) by MIT and Harvard Biomedical Engineering Center. Documentation and listings for 8080 from CP/M Users Group, 164 west 83rd Street, New York, N.Y. 10024. $4.00 membership, $8.00 per 8" floppy, 2 floppies needed.

CONVERS
The Digital Group, Box 6528, Denver, CO 80226 Manual: DOC-CONVERS $12.50 .

URTH
(University of Rochester FORTH), Tutorial Manual, Hardwick Forsley, Laboratory for Laser Energetics, 250 E. River Rd., Rochester, NY 14621 .

microFORTH Primer
FORTH, Inc. 815 Manhattan Ave., Manhattan Beach, CA 90266 (moving soon) $15.00 .

(Page 21, 22 Blank)