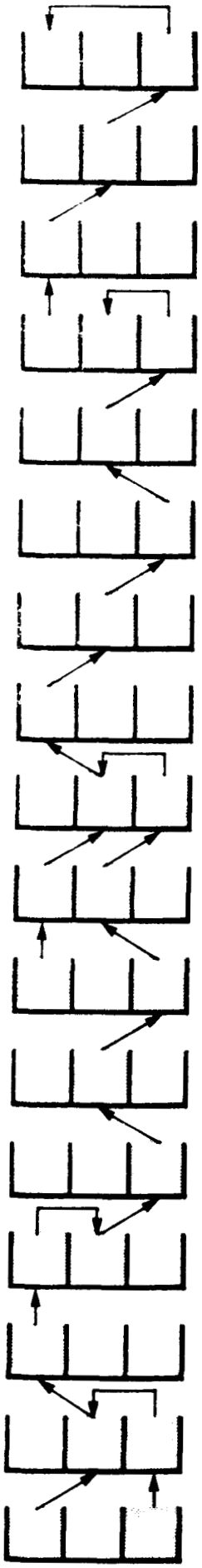

FORTH DIMENSIONS

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

Volume II
Number 5
Price $2.00

INSIDE

FORTH DIMENSIONS

Published by Forth Interest Group

Volume II No. 5 January/February 1981

Publisher Roy C. Martens

Guest Editor S. B. Bassett

Subscription to FORTH DIMENSIONS is free with membership in the Forth Interest Group at $12.00 per year ($24.00 overseas air). For membership, change of address and/or to submit material, the address is

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

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 2,400 is worldwide. It was formed in 1978 by FORTH programmers to encourage use of the language by the interchange of ideas through seminars and publications.

EDITOR'S COLUMN

Alan Taylor, in his speech at the FORTH Convention banquet, pointed out that FORTH is an incredibly powerful tool, and not precisely a language, in the traditional sense.

FORTH works close to the "naked machine" and yet is as general and powerful as many High Level Languages. This makes FORTH the perfect language for writing compilers or pseudo-compilers for other languages.

If we were to produce a compiler for ANSI Standard COBOL, for instance, COBOL would run on any machine which had FORTH running—which is easy!

A major software house is presently writing an Ada compiler, and expects to have it out in 1983.

I am willing to bet that a working FORTH compiler for Ada could be written in half that time—does anyone want to take me up on the bet?

S. B. Bassett

PUBLISHER'S COLUMN

1980 retropective: FORTH DIMENSIONS has completed a whole year with the new format and a number of people think it gets better with each issue. FORTH Interest Group has grown from 647 members on July 1st to over 2,400, thanks to *Byte*, *EE Times*, *InfoWorld*, *ComputerWorld* and other publications. The FORML Conference and FIG Convention were great successes. Many new chapters are being formed, how about you forming one? FORTH vendors are increasing almost as fast as the membership. FORTH is being implemented on more and more machines and applications are beginning to roll out. It's been a great year for us and we hope for you.

1981 perspective: FORTH DIMENSIONS will get better, we will have paid guest editors for each issue. A number of new publications and other FIG items will be introduced (see order form). A number of mailings will be done to FIG members about products available (if you *don't* want to receive these mailings, please drop us a note). We'll be doing more publicity to trade magazines and at computer shows. We need more aticles for FORTH DIMENSIONS, send yours in. Happy New Year!

Roy Martens

A NEW SYNTAX FOR DEFINING DEFINING WORDS

William F. Ragsdale

ABSTRACT

The computer language FORTH utilizes a syntax that is generally context free (i.e., postfix, or reverse Polish). However, deviations from this principle are noted in the syntax for words that themselves define words. This paper presents an altered form, which improves clarity of expression, and generalizes the construction for compilers which generate FORTH systems (meta-FORTH).

BACKGROUND

Compilation of FORTH programs consists of adding to memory a sequence of numerical values (addresses) corresponding to source text (words). This period is called compile-time. These values, called compilation addresses, are later interpreted by the address interpreter (at their run-time). They specify actual machine code which is ultimately executed, under control of the address interpreter.

1. FORTH source syntax is close to FORTH object code.

2. Traditional computer languages require significant processing to convert their syntax to object code. The syntax conversion is specified using Backus-Naur statements or "rail-road-track" diagrams of N. Wirth.

3. The traditional compiler's conversion adds complexity to the compiler, increases complexity and compiler size. It also masks the results from the user so that the use can't see or control the object code FORTH reduces complexity by requiring the user to write in a direct, simple syntax.

The espoused benefits are:

1. The programmer directly controls program flow. Inefficiency should be more apparent to the programmer.

2. The compiler is simpler, smaller, and more time efficient.

3. Compiler functions may be added by the programmer consistent with those previously in the system.

The arguments against having to write in this form include:

1. God created infix notation. If not so, why did we learn it as children? God doesn't lie to little children.

2. Languages are created by compiler writers, not compiler users. Therefore, let these brilliant sources have a larger part of the pie (read headaches for pie).

For completeness, it should be noted that program branching requires reference to addresses not at the point of compilation. The compiling words of FORTH (DO, UNTIL, IF, etc.) use the compile time stack to hold interim addresses which specify such branching. The nesting of conditionals keeps this process simple and efficient, and obiviates the need for backtracking or looking ahead in the source text.

A PROBLEM

Three exceptions to a context free expression exist in FORTH as generally used and formalized in FORTH-79:

1. The word IMMEDIATE sets the precedence flag of the most recent definition in the CURRENT vocabulary. Location of this bit is done via a variable/vocabulary pointer pointing backward in memory an unknown amount. If selection of the CURRENT vocabulary has been altered, the wrong definition is made immediate.

2. Defining words create a dictionary word header, but some other word backtracks in the object code to change the execution procedure assigned to each word created. E.g.,

```
: C-ARRAY  CREATE  ALLOT  DOES> + ;

10  C-ARRAY  DEMO
```

   The word DEMO is created by CREATE as a variable and is proximately altered by DOES> to execute with a much different role in making a run-time address calculation.

3. ; and END-CODE make available for use each correctly compiled definition. This is often determined from an alterable pointer, sensitive to the vocabulary specified as CURRENT.

To display these cases together:

```
: WWW . . . ;
: XXX . . . ;  IMMEDIATE
: YYY . . . ;CODE . . . END-CODE
: ZZZ . CREATE . . . DOES> . . . ;
```

Each of the above words is quite different in function and execution, yet they were all defined by : . The user must analyze the contents of each definition to determine what type of word it is (i.e., colon-definition, compiling-word, code-definer, or high-level-definer). Because of these varied forms, the glossary definition of : is only partly complete. The other variations on : must be discovered as they occur.

Creation and use of the above types is complicated in that the resulting functions are dependent on words following within and outside (!) each definition. As new words are defined by CREATE and assigned execution code by DOES> and ;CODE , the compiling function must backtrack by implicit pointers to alter previously generated word headers.

Added commentary is appropriate for item 2, above. It is a general characteristic of FORTH that a word's function may be uniquely determined by the contents of its code field. This field points to the actual machine code which executes for this word. Common classes of words which are consistent include: variables, constants, vocabularies.

This is not the case with defining words. These words all have the same code field contents as any other colon-definition which indicates that they execute interpretively until the concluding ; . But actually the intervening DOES> or ;CODE terminates execution and alters the specification for the execution of the word being defined. Philosophically, it appears that this is the grossest form of context sensitivity of any language, due to the generality and power of the construct.

But this power and generality contains its own downfall. It increases the complexity of comprehension and complexity of compilation. When a novice competently begins to use DOES> and ;CODE he has come of age in FORTH.

THE GOAL

The goal of the proposed new technique is a uniform expression of source text that may be compiled for resident RAM execution, resident ROM execution or target execution (from a binary image compiled to disk for later execution). To enable this uniformity, a context free expression is used.

THE PROPOSAL

The proposed syntax for defining words uses only the compile time stack (or dedicated pointers), generating object code and word headers linearly ahead. Each word type has a unique defining word so that no later modification of a word definition need be made. A meta-defining word is proposed which makes all defining words. Each defining word is obvious because each, itself, is created by this "meta-definer".

This meta-definer is BUILDS> . This name is an old friend to some, since it was the name of the word previously used where CREATE is specified by FORTH-79. This word still has its old role of building words which themselves build words, but is used in a more obvious fashion.

Here is an example, written in FORTH-79 for a word which creates singly dimensioned byte arrays:

```
: C-ARRAY   CREATE  ALLOT
            DOES>   +   ;
```

It would be used in the form:

```
10  C-ARRAY  DEMO
```

to make an array named DEMO with space for 10 bytes. When DEMO executes it takes an offset from the stack and returns the sum of the allotted storage base address plus the offset.

Using the proposed new meta-definer BUILDS> the creation of C-ARRAY is:

```
DOES> + ;
  (the run-time part)
BUILDS> C-ARRAY  ALLOT ;
  (the compile-time part)
```

And is used:

```
10  C-ARRAY  DEMO
  just as above.
```

It should be noted that the impact of the use of BUILDS> is only in defining defining words. Later use of such defined words would be as presently conventional.

THE NEW SYNTAX

Here is a summary of the defining word syntax that appears at the application level. Note that these examples are very close to what we commonly use in FORTH-79.

: <name> . . . . ;

> Define a non-immediate word which executes by the interpretation of sequential compilation addresses.

NOW <name> . . . . ;

Define an immediate word which executes by the interpretation of sequential compilation addresses, and will execute when encountered during compilation.

CREATE <name>

As in FORTH-79.

n CONSTANT <name>

As in FORTH-79.

VARIABLE <name>

As in FORTH-79.

VOCABULARY <name>

As in FORTH-79, but each defined vocabulary is immediate.

When the programmer creates new word types, a significantly different syntax is used, as compared to FORTH-79.

DOES> . . . . ;

Begin the nameless run-time high-level code for words to be defined by <name>.

BUILDS> <name> . . . . ;

<name> <namex>

Define <name> which, when later executed will itself create a word definition. The code after <name> executes after creating the new dictionary header for <namex> to aid parameter storage. The previous run-time code is assigned to each word <namex> created by <name>.

When new classes of words are created with their run-time execution expressed by machine code, their defining word is created thusly:

CODE> . . . . END-CODE

Begin the nameless run-time machine code for words to be defined by <name>.

BUILDS> <name> . . . . ;

<name> <namex>

Define <name> which, when later executed will itself create a word definition. The code after <name> executes after creating the new dictionary header for <namex> to aid parameter storage. The previous run-time code is assigned to each word <namex> created by <name>.

THE METHOD

We will follow the method of the honey bee. To propagate the colony the bees need a queen bee. An ordinary bee is fed special hormones to become a queen bee. By regulating this process the colony regulates its growth.

Our queen bee will be BUILDS> . It is originally created as a colon definition. Then it is converted into a new type of word that creates words which always create. This form uses parameters to create a dictionary entry and then passes control to the users code which specifies completion of the entry.

We will break the CREATE DOES> construct into two parts. The creating part will be called BUILDS> with the right pointer emphasizing that the following word 'builds' other

words. BUILDS> is the meta-defining word since it is the source of all defining words. It must be emphasized that the word creating function is inherent in any word created by BUILDS> , and need not be additionally specified.

The execution procedure is begun by CODE> (for words with a machine code execution) or by DOES> (for words with a high-level execution). Coupling from these two words is accomplished by passing an address and bit mask from DOES> or CODE> to BUILDS> .

The precedence of a word traditionally is set by declaring each such word as IMMEDIATE . In the new form, this is declared for the defining word, not for each word as defined. By executing IMMEDIATE after the CODE> or DOES> part, but before the BUILDS> part, the bit mask on the stack is altered to the immediate form. This mask is applied to all words as later defined, so all will be immediate.

Usually colon-definitions and code definitions are created 'smudged' so that they will not be found during a dictionary search. When successfully compiled, the smudge bit is reset, making the word available for use. Other words are much less susceptible to errors of compilation, and so are created un-smudged. The smudge function is not generally manipulated by the user but completed by ; or END-CODE. The smudge bit is contained in the header count byte.

By executing SMUDGE after the CODE> or DOES> part, but before the BUILDS> part, every word later created will be created smudged. It is a system choice how the un-smudging is performed. It is suggested that a pointer uniquely specify the current smudged bit address.

Some systems achieve the same result by selectively linking words into the dictionary. In this case the selective linking is done by the defining part of BUILDS> as selected by the bit mask associated with each defining word.

A major problem exists for meta-compilation (target-compilation) of new defining words. The compile-time portion must know the run-time compilation address corresponding to each word type. Several methods are currently used. In all cases the syntax is a deviation from the usual version suitable for testing on a resident system. Part of the art of target compilation is knowing how to alter resident defining words to operate in the target compilation situation.

The programmer may declare byte counts to allocate memory space and later re-origin compilation to fill in code fragments. Other techniques consist of compiling the full structure and then passing address locators to previously defined words. In polyFORTH, dual definitions are used. The target compilation definition of our C-ARRAY example is:

```
: C-ARRAY CREATE ALLOT ;CODE FORTH
: C-ARRAY CREATE ALLOT DOES> + ;
```

It is an exercise in ingenuity to determine which parts of the above code end up in the target system, and which are added to the host compiler.

Here is a summary of the meta-compiling of our example:

```
DOES> + ;
BUILDS> C-ARRAY ALLOT ;
10 C-ARRAY DEMO
```

First the DOES> compiles <does> + ; into the target system and passes

the locating parameters to BUILDS> . <does> is an in-line code vector to machine code.

Then the BUILDS> compiles C-ARRAY ALLOT ; into the target system with the proper object locators for the DOES> part and then places another copy of C-ARRAY ALLOT ; into the resident compiler so that C-ARRAYs may be immediately defined for the target system.

Finally, the C-ARRAY in the host system executes to place a definition for DEMO into the target system, locate the address of DEMO for later compilation, and finally ALLOT ; makes the target memory allocation and concludes the target definition.

The only source changes anticipated are the occasional explicit change of vocabulary to correctly select (during target compilation) words which affect the application memory. Again, this is only done for selected defining words.

The key to this method is that the run-time portion is known before the compile-time portion, and the creation of defining words is done uniformly, linearly ahead.

CONCLUSION

A complete implementation of these concepts follows. A six word glossary expands the explanations given above. This implementation is written in FORTH-79, with system dependent words taken from fig-FORTH. The source of each word is identified in the Appendix.

This construction for BUILDS> is offered as a method to regularize the structure of FORTH at the defining word level. Its success will be judged by either usage or the stimulation of other methods for this purpose.

GLOSSARY

BUILDS> addr mask ---

A defining word used in the form:

BUILDS> <name>. . . . ;

to define a defining word <name> . The address and mask (left by either DOES> or CODE>) are placed into the definition of <name> to specify the header structure for all words created by <name> and locate the execution procedure assigned by <name> . The text between <name> and ; is compiled to complete the definition of <name> .

When <name> executes in the form:

<name> <namex>

it generates a dictionary entry for <namex> and then executes the code following <name> to finish compilation of <namex> .

When <namex> executes, it executes the code in the DOES> or CODE> part preceding <name> . Refer to DOES> or CODE> .

CODE> --- addr mask

Used in the form:

CODE>..(assembly text)..END-CODE

to begin the nameless compilation of a sequence of assembler code text. The address and mask left locate this sequence for BUILDS> . The mask contains the precedence and smudge bits and may be

altered by IMMEDIATE and/or SMUDGE while still on the stack, before being compiled by BUILDS> .

When a word with a CODE> part ultimately executes, it executes the code between CODE> and END-CODE, at a machine code level. Execution must ultimately be returned to the address interpreter NEXT .

DOES> --- addr mask

Used in the form:

DOES> . . . . ;

to begin the nameless compilation of a sequence of high-level code. The address and mask left locate this sequence for BUILDS> . The mask contains the precedence and smudge bits and may be altered by IMMEDIATE and/or SMUDGE while still on the stack, before being compiled by BUILDS> .

When a word with a DOES> part ultimately executes, it executes the code following DOES> with its own parameter field address automatically placed on the stack.

IMMEDIATE addr mask --- addr mask

Set the precedence bit in the mask to indicate that all words later defined by the defining word being defined will always execute when encountered.

Immediate words are aids to compilation, such as:

IF BEGIN DO ." etc.

NOW

A defining word used in the form:

NOW <name> . . . . ;

to define <name> in the fashion of : , but in the immediate form. That is, <name> will execute even when encountered during compilation.

SMUDGE addr mask --- addr mask

Set the smudge bit in the mask to indicate that all words defined by the defining word being defined will begin in the 'smudged' condition. This condition prevents a word from being found in a dictionary search until un-smudged at the completion of correct compilation.

APPENDIX

The example implementation of the new BUILDS> is written in FORTH-79 running on a 6502 processor. When system dependencies occur, the fig-FORTH methods were used regarding error control and dictionary header structure. Here is a tabulation of the pedigree of each word (its origin) used in this application.

Numbers indicate a standard definition from FORTH-79, fig indicates the definition from fig-FORTH. Assembler words are from a 6502 assembler.

!	122	AGAIN	fig	LOOP	124
!CSP	fig	ALLOT	154	MIN	127
'	171	ASSEMBLER	fig	OR	223
(	122	BL	fig	OVER	170
-	121	C@	156	QUIT	211
+!	157	C,	fig	ROT	212
.	143	CFA	fig	SMUDGE	fig
=	134	COMPILE	146	SWAP	230
1+	107	CONTEXT	151	TOGGLE	fig
1-	105	CCUNT	159	VARIABLE	fig
2+	135	CSP	fig	VOC-LINK	fig
:	116	CURRENT	137	WORD	fig
;	196	DO	142	[	125
=	123	DP	fig	[COMPILE]	179
>R	200	DUP	205	]	126
?CSP	fig	HERE	188		
@	199	I	136		
		LOAD	202		

```
SCR # 6
  0 (  Adopt form of FORTH-79                              WFR-30NOV08 )
  1 : CREATE   0  VARIABLE  -2  ALLOT  ;
  2 : 1-    1   -  ;                 : 2-  2  -  ;
  3 : WORD   WORD  HERE  ;
  4 : 'SMUDGE   SMUDGE  ;   ( rename for access to old version )
  5 : END-CODE    ASSEMBLER  [COMPILE]  C;  ;
  6
  7 ( ********* BEGINNING OF THE NEW  BUILDS>  PACKAGE ***********)
  8 79-STANDARD  ( but this is not a standard program )
  9 HEX
 10 : 2DUP    OVER  OVER  ;
 11 : THRU   1+  SWAP  DO  I  LOAD  LOOP  ;
 12 : IMMEDIATE    40  OR  ;        ( next word defines immediates )
 13 : SMUDGE       20  OR  ;          ( next word defines smudged )
 14
 15 DECIMAL   7 11 THRU    DECIMAL

SCR # 7
  0 ( META-definitions of DOES> and CODE>                  WFR-30NOV08 )
  1 CREATE  <DOES>            ASSEMBLER  HEX
  2    DEX,  DEX,                          ( make room for pfa value )
  3    CLC,  2 # LDA,  W     ADC,  BOT     STA,
  4             TYA,  W 1+ ADC,  BOT 1+ STA,       ( copy the pfa )
  5    SEC,  PLA,  1 # SBC,  W STA,
  6          PLA,  0 # SBC,  W 1+ STA,  ( ready for hi-level call ,
  7    ' QUIT CFA @ JMP,    ( make hi-level call for DOES> )
  8
  9 : DOES>     ( run time of META word  --- cfa, count byte mask )
 10    HERE  80  ( leave locators )  !CSP  'SMUDGE ( compensate ; )
 11    !  ( begin compiling )  20  C,  COMPILE  [  <DOES> , ]  ;
 12
 13 : CODE>     ( run time of META word  --- cfa, count byte mask )
 14    HERE  80  ( leave locators )  !CSP  'SMUDGE
 15    [COMPILE]  ASSEMBLER  ;

SCR # 8
  0 ( META-definition of BUILDS>                           WFR-30NOV08 )
  1 DOES>         ( run-time for meta-BUILDS> which makes headers )
  2    COUNT  ( pfa+1, count mask )
  3    BL  WORD  DUP  C@  1+  ALLOT  ( the name )
  4    DP  C@  OFD  =  ALLOT  ( for 6502 only )
  5    DUP  ROT  TOGGLE  HERE  1-  80  TOGGLE  ( name marker bits )
  6    CURRENT  @  @  ,  CURRENT  @  !  ( link into vocabulary )
  7    DUP  @  ,  ( lay down code field )
  8    2+  >R  ( resume the BUILDS> word  )  ;
  9
 10 SMUDGE
 11 : BUILDS>                                ( begins defining words )
 12    [  -2 ALLOT OVER  ,  ( change cfa to above DOES> )  -4 CSP +!
 13          C,  ,  ( lay down count mask, then cfa )  ]
 14       ?CSP  C,  ,  !CSP  CURRENT  @  CONTEXT  !  ]  ;
 15

SCR # 9
  0 (  CODE  ;  and  NOW                                   WFR-30NOV08 )
  1 CODE>     END-CODE       SMUDGE       ( no execution procedure )
  2 BUILDS> CODE                    ( create a smudged code definition )
  3         HERE  DUP  2-  !  [COMPILE]  ASSEMBLER  !CSP  ;
  4
  5 CODE>                                      ( for colon-definitions )
  6    IP 1+ LDA,  PHA,  IP LDA,  PHA,  CLC,  W LDA,  2 # ADC,
  7    IP STA,  TYA,  W 1+ ADC,  IP 1+ STA,  NEXT JMP,   END-CODE
  8
  9 2DUP  SMUDGE           -4  CSP +!   ( use these params twice  )
 10 BUILDS>  :            ( create new colon-definitions till ';' )
 11          !CSP  CURRENT  @  CONTEXT  !  ]  ;
 12
 13 IMMEDIATE  SMUDGE
 14 BUILDS>  NOW                ( creates immediate colon-definitions )
 15          [ ' : 3 + (  share code within ':' )  1 ]  AGAIN  ;

SCR # 10
  0 (  VARIABLE  CREATE  and  CONSTANT                     WFR-30NOV08 )
  1 CODE>                                              ( for variables )
  2    CLC,  W LDA,  2 #  ADC,   PHA,
  3             TYA,  W 1+ ADC,  PUSH JMP,    END-CODE
  4
  5 2DUP                -4   CSP +!          ( share run-time code )
  6 BUILDS> CREATE  ;                     ( general purpose creator )
  7
  8 BUILDS> VARIABLE  0 , ;  ( create a variable, not initialized  )
  9
 10
 11 CODE>                                              ( for constants )
 12    2 # LDY,  W )Y LDA,  PHA,
 13       INY,  W )Y LDA,  PUSH JMP,   END-CODE
 14
 15 BUILDS>  CONSTANT  , ;  ( create a constant, value from stack )

SCR # 11
  0 (  VOCABULARY,  ARRAY                                  WFR-30NOV08 )
  1 DOES>   2+  CONTEXT  !  ;
  2
  3 IMMEDIATE       ( note that all vocabularies will be immediate )
  4 BUILDS>  VOCABULARY
  5       A081  ,  CURRENT  @  CFA  ,
  6       HERE  VOC-LINK  @  ,  VOC-LINK  !  ;
  7
  8 VOCABULARY A-TRIAL
  9
 10
 11       ( one dimensional byte array, confined within allocation )
 12 DOES>    COUNT  ROT  MIN  +  ;
 13 BUILDS>   C-ARRAY  DUP  1-  C,  ALLOT  ;
 14
 15 10   C-ARRAY FOR-TEST
```

FIG GROUPS

Standards--Bill Ragsdale, c/o fig, P.O. Box 1105, San Carlos, CA 94070.

FORML--Kim Harris, P.O. Box 51351, Palo Alto, CA 94303.

8080 Renovation Project--cleaning up the figFORTH 8080 implementation --Terry Holmes, c/o fig, P.O. Box 1105, San Carlos.

figGRAPH--Howard Pearlmutter, c/o fig P.O. Box.

figSLICE--The FORTH Machine, to be built with bit slice technology--Martin Schaaf, 202 Palisades Dr., Daly City, CA 94015.

figTUTOR--how to teach FORTH to new people--forming--Sam Bassett, c/o fig P.O. Box.

HELP!! MAYDAY!!

The Editors, not being "old FORTH hands", need experienced LOCAL help in testing submitted programs.

Diversity of systems (fig or not) and terminals much appreciated.

Reply to fig P.O. Box, please!

INPUT NUMBER WORD SET

Robert E. Patten

Purpose

The FORTH primitives <# # #S $ SIGN #> allow generalized numeric output. This paper presents a generalized method for numeric input.

Method

This word set, as implemented, will convert a word placed at HERE, terminated with a trailing blank, to a double integer on the data stack. The type of input converted is available in the variable TRAIT.

This word set will allow extentions to include other number and data types (i.e., floating point, triple precision numbers, and simple string parsers).

Most words in this word set expect a flag on the data stack and leave a flag indicating success or failure of the conversion or test performed. A true flag indicates success. The word CHR is an exception. CHR replaces the flag with a character from the word at HERE. If the last conversion or test was a failure, CHR leaves the same character. If the last operation was a success, CHR leaves the next character on the data stack.

The defining word N: may be used to create a word to convert the word at HERE to a double integer. A successful conversion will leave a double integer and a true flag. A failure will leave only a false flag. If words defined by N: are used to define a word created by UNTIL: then this new word will, when executed, try each N: created word on the word at HERE until one is successful, leaving only a double integer on the stack. If none of the words are successful then nothing is left on the data stack. This outcome is not acceptable because no number was put on the data stack. Because of this, the last word in the UNTIL: defined word should cause a "Word not defined" error.

INPUT NUMBER WORD SET

(<N) --- flag d flag

Leave a plus sign, a double number zero, and a true flag on the data stack in preparation for number conversion.

(N>) sign d flag --- d true
--- false

If flag is true apply sign to double number and leave number and true flag else leave only false flag.

.N d1 flag --- d2 flag

Substituting zero for blank, convert CHR into double number beneath leaving true flag if ok, else leave false flag.

>CHR --- addr

Leave the address of a variable which contains a pointer to the last character fetched by CHR.

?,NNNS d1 flag --- d2 flag

Allow groups of comma and three digits to be converted to double number beneath. If no comma return true flag. If no three digits following the comma then return false flag.

?. flag --- flag

If CHR is a period then set DPL to zero and leave true flag else leave false flag.

?ASCII flag --- flag

If CHR equals character following ?ASCII then leave true flag else leave false flat.

?BOTH flag --- flag

If flag is true and CHR is a blank then leave a true flag else leave a false flag.

?END flag --- flag

If CHR is a blank then leave a true flag else leave a false flag.

?SIGN false d flag --- sign d flag

If CHR equals - change false flag to a true flag and leave true flag on top else leave false flag on top.

?SKIP flag --- flag

Make flag true. Used to skip past character if flag was false.

ASCII --- char

Place following character on data stack as a number.

CHR flag --- char

Add flag to >CHR and fetch character at >CHR to data stack.

N: A definining word used in the form:

```
N: <name> . . . ;
      --- d true
      --- false
```

Convert word at here leaving double number and a true flag on the data stack. If word does not convert leave only a false flag on the data stack.

N d1 flag --- d2 flag

Convert digit at CHR into double number beneath. If successful leave true flag else a false flag.

NNN d1 flag --- d2 flag

Do three N leaving true flag if successful else false flag.

NS d1 flag --- d2 flag

Do N until failure, leaving a true flag on top of data stack with >CHR pointing to last character accepted

(N) d1 digit --- d2

Convert binary digit into number beneath.

REQUIRED flag --- flag

If flag is false exit this word leaving false flag. If flag is true leave true flag and continue.

TRAIT --- addr

A variable containing the word count from the last UNTIL: defined word.

UNTIL:

A defining word used in the form:

```
UNTIL: <name> . . . ;
```

Words created by UNTIL: are like colon-definitions except the run time function is to execute words in the definition until there is a true flag on the data stack, then exit the word leaving the word count of the words executed in the variable TRAIT.

```
( ASCII TO BINARY WORD SET REP )  BASE @ DECIMAL
0 VARIABLE >CHR
: ((N) ( -- df d cf )   0 0.0 1   -1 DPL ! HERE        >CHR ! ;

: CHR   ( f -- character )  >CHR +! >CHR @  C@;

: (N>) ( f d1 f -- d2 tf   good number
                -- ff      bad number  )
   'R ROT IF DMINUS THEN R>  IF 1 ELSE DROP DROP 0 THEN ;

: COLON  ( Build : def. with security.)
 'EXEC !CSP  CURRENT @ CONTEXT !  <BUILDS ]  SMUDGE ;

: N:  ( -- d tf / -- ff )
    COLON  DOES>            ' (N>) >R ( EXECUTE AFTER )
    >R  ( EXECUTING PARAMETER FIELD )  ((<a) ( SET UP STACK ) ;

0 VARIABLE TRAIT
: UNTIL:     COLON DOES> >R  -1 TRAIT !
  BEGIN   1 TRAIT +!   R @ EXECUTE  R> 2+ >R UNTIL
  ( EXECUTE WORDS UNTIL TRUE)    R> DROP  ( THEN EXIT. ) ;

: (N) ( d1 digit -- d2) SWAP BASE @ U* DROP  ROT BASE @ U* D+
     DPL @ 1+  IF 1 DPL +! THEN ;

: N ( d1 f -- d2 f ) CHR BASE @ DIGIT  IF (N) 1 ELSE 0 THEN ;

: NNN ( d1 f -- d2 f )  N N N ;

: NS ( d1 f -- d2 tf ) N BEGIN  DUP WHILE N REPEAT
     0= -:  >CHR +! ;

: REQUIRED ( f -- f : if false exit else continue )
      DUP 0=  IF R> DROP THEN ;
: ?ASCII      BL WORD HERE 1+ C@   [COMPILE] LITERAL ; IMMEDIATE
: ?ASCII ( f -- f )   COMPILE CHR [COMPILE] ASCII
   COMPILE = ;  IMMEDIATE
: ?SIGN ( ff d f -- f d f )  ?ASCII - DUP
  IF >R ROT 0= ROT ROT R> THEN ;
: ?,NNNS   ( d1 f -- d2 f )    BEGIN ?ASCII ,  DUP
    WHILE  NNN REQUIRED REPEAT  0=   -1 >CHR +!  ;
: ?. ( f -- f ) ?ASCII .   DUP  IF 0 DPL ! THEN ;
: ?SKIP  ( f -- tf ) DROP 1 ;
: ?END ( f -- f )  CHR BL = ;
: ?BOTH ( f -- f ) DUP IF ?END   THEN ;
: .N ( d1 f -- d2 f    AUTOSCALE)
   N DUP 0= IF ?END IF 2 (N) 1  ELSE 0 THEN THEN ;

N: INTEGER ?SIGN NS ?,NNS REQUIRED  ?END ;
N: REAL ?SIGN NNN ?,NNNS REQUIRED ?. NS ?END   ;
  : :NN  ?ASCII : REQUIRED   6 BASE ! N REQUIRED  DECIMAL N ;
N:  TIME  ( HH:MM:SS ) DECIMAL   N N DROP OVER  24 < 0= 0=
    REQUIRED   :NN REQUIRED :NN  ?BOTH REQUIRED ) DPL ! ;

N: SSN DECIMAL NNN REQUIRED ?ASCII - N N REQUIRED ?ASCII - NNN N
 ?BOTH REQUIRED 0 DPL ! ;          N: ZERO ;
N: AREA-CODE  ?ASCII ( REQUIRED NNN REQUIRED ?ASCII ) ?BOTH ;
N: PHONE NNN REQUIRED ?ASCII - REQUIRED NNN N ?BOTH REQUIRED
   0 DPL ! ;
N: DOLLAR DECIMAL  ?ASCII $ REQUIRED ?SIGN NNN ?,NNNS ?. .N .N
   0 DPL ! ;

: (BAD) 0 ERROR ;

UNTIL: (NUMBER) INTEGER REAL DOLLAR TIME SSN PHONE AREA-CODE
   (BAD)  ;

: NUMBER DROP BASE @ >R  (NUMBER) R> BASE ! ;

' NUMBER CFA ' INTERPRET 36 + !

BASE ! ;S


( TEST (NUMBER)                  14 LOAD
: GET-NUMBER     QUERY BL WORD   HERE  NUMBER D. TRAIT ? DPL ? ;

: NUMBER-TEST    BEGIN CR GET-NUMBER ?TERMINAL  UNTIL ;

: X IF D. DPL @ 1- ELSE ." BAD " THEN ;

: TEST-N: [COMPILE] ' QUERY BL WORD CFA EXECUTE X ;
: <N ((N) QUERY BL WORD ;
: N> (N>) X .S ;
                              ;S
```

ATARI DISKETTE

Diskette and documentation for fig-FORTH on ATARI computers. Runs on one disk drive and 16K RAM. Has full screen editor and extensions. $50.00

Bob Gonsalves
c/o Pink Noise Studios
1411 Center Street
Oakland, CA 94607

NEW PRODUCTS

FORTH for OSI

by Forth-Gear

Forth-Gear is pleased to announce the release of a complete FORTH software package for several models of Ohio Scientific Instruments computers. The Forth Interest Group model language runs under OSI's Disk Operating System OS65D-3.2, but high level FORTH DOS words are implemeted in FORTH for full compatibility with fig-standard extensions. A line editor is included for the creation and disk storage of FORTH programs. A 6502 assembler permits the use of machine code routines as FORTH definitions. The editor and assembler may both be extended by the creation of new definitions in high level FORTH.

Included with the package are several utility programs in FORTH, including a RAM Dump, video graphics, data disk initializer (may use all tracks except track zero), a sample machine code routine (screen clear), and a system disk optimizer.

Minimal system requirements are 24 Kilobytes of RAM and one disk drive. System attributes beyond the minimal requirements may be fully utilized by regenerating the system disk with the optimizer program. Two systems are currently available: The 5 1/4" disk version works on all C2-4P and C4 models. The 8" disk version works on all C2-8P, C8P, C2-OEM, and C3 models with either the polled keyboard or a serial terminal. Superboard, C1P, and C2 versions will be available very soon.

A single-user system consisting of a disk (specify size) and fifty page user manual is available from Consumer Computers, 8907 La Mesa Blvd., La Mesa, California 92041, for the introductory price of $69.95 prepaid. Telephone (714) 698-8088 9 to 5 PST.

STRUCTURED PROGRAMMING BY ADDING MODULES TO FORTH

Dewey Val Schorre

Structured programming is a strong point of FORTH, yet there is one language feature important for structured programming which is currently absent in FORTH. This feature is called a module in the programming language MODULA, and appears under other names in other languages, such as procedure in PASCAL. It can, however, be easily added by defining three one-line routines.

The names of these routines are: INTERNAL, EXTERNAL and MODULE. A module is a portion of a program between the words INTERNAL and MODULE. Definitions of constants, variables and routines which are local to the module are written between the words INTERNAL and EXTERNAL. Definitions which are to be used outside the module are written between the words EXTERNAL and MODULE.

One of the most common uses of modules is to create local variables for a routine. These variables are defined between INTERNAL and EXTERNAL. The routine which references them is defined between EXTERNAL and MODULE. Notice that this module feature is more general than the local variable feature of other programming languages, in that several routines can share local variables. Such sharing is important, not so much from the standpoint of saving space, but because it provides a means of communication between the routines.

If you have written any local routines between the words INTERNAL and EXTERNAL, then in order to debug them, you will have to delete the word INTERNAL and put a ;S before the word EXTERNAL. Since debugging in FORTH proceeds from the bottom up, once you have debugged these local routines, you will have no further need to refer to them from the console. They will only be referenced from the external part of the module. Modules can be nested to arbitrary depth. In other words, one module can be made local with respect to another by defining it between the words INTERNAL and EXTERNAL.

Now let's consider matters of style. The matching words INTERNAL, EXTERNAL and MODULE should all appear on the same screen. When modules are to be nested, one should not actually write the lower level module between the words INTERNAL and EXTERNAL, but should write a LOAD command that refers to the screen containing the lower level module. The screens of a FORTH program should be organized in a tree structure. The starting screen which you LOAD to compile the program is a module which LOAD's the next level modules.

Screens are much better for structured programming than the conventional character string file because they can be chained together in this tree structured manner. You will write a module for one program, and when you want to use it in another program, you don't have to edit it into the new program or add it to a library. All you have to do is to reference it with a LOAD command.

There is an efficiency advantage to the use of modules. One minor advantage is that compilation speed is improved because the dictionary that has to be searched is shorter. The more important advantage of saving dictionary space is not realized with this simple implementation, which changes a link in the dictionary. To save space, one would have to implement a dictionary that

was separate from the compiled code. Moreover, this dictionary would not be a simple push-down stack, because the storage freed by the word MODULE is not the last information entered into the dictionary.

The words needed to define modules are as follows:

```
: INTERNAL  ( --> ADDR)  CURRENT @ @ ;
: EXTERNAL  ( --> ADDR)  HERE ;
: MODULE( ADDR1 ADDR2 --> )PFA LFA ! ;
```

FORML CONFERENCE

A Report on the
Second FORML Conference

The Second Conference of the Forth Modification Laboratory (FORML) was held over Thanksgiving, November 26 to 28, 1980, at the Asilomar Conference Center, Pacific Grove, California (some 120 miles south of San Francisco).

The weather was unseasonably beautiful, as the rainy season, normally starting in November, was late. Most conference attendees managed to find some free time to enjoy the beach and wooded areas.

With the way smoothed by a core crew who showed up Tuesday, the majority of participants arrived for lunch Wednesday, and launched right into a full schedule of technical sessions.

There were 65 conference attendees, with enough of them bringing family to raise the count to 96 people at Asilomar in connection with FORML.

The rooms were in scattered well-landscaped buildings. Meals were provided in a central dining building, and were generally praised. Thanksgiving noon dinner, a deluxe buffet meal, was a special treat.

The evening meetings, both Wednesday and Thursday, had formal technical sessions which evolved into quite open, informal, and productive discussions. The participants had to be persuaded to break up to move to the scheduled social gatherings over wine and cheese.

SUMMARY OF SESSIONS

The number of people presenting papers was so great (almost 40) that sessions were scheduled from Wednesday afternoon all the way to Friday afternoon. Topics of sessions, together with their chairmen, were:

FORTH-79 Standard
    Bill Ragsdale

Implementation Generalities
    Don Colburn

Implementation Specifics
    Dave Boulton

Concurrency
    Terry Holmes

FORTH Language Topics
    George Lyons

Other Languages
    Jon Spencer

MetaFORTH
    Armand Gamberra

Programming Methodology
    Eric Welch

Applications
    Hans Niewenhuijzen

In addition, Kim Harris, the Conference Chairman, opened the Conference with a welcome and a review of FORML-1, London, January 1980. Kim also closed the final session.

As one example of a conference paper, "Adding Modules to FORTH" by Dewey Val Schorre, gave a mechanism for setting up words which are local to a "module"--a sequence of FORTH code. His mechanism involves only three FORTH words, two of which already exist in FIG-FORTH. His novel but straightforward way of using these three simple words provides many of the benefits of VOCABULARY with less overhead, and by focusing on modularity, it can lead to clearer programs.

Another item of particular interest was George Lyons's paper on Entity Sets. His proposal is very economically implemented, and allows, at compile time, selection from lists of identically-named operators, such as @ ! + , based on data type.

These and other wonders will be published in the Proceedings of the Conference. This should be ready by the end of February, and will be sold by FIG.

LESS FORMAL OBSERVATIONS

At the Wednesday evening technical session an informal discussion on various topics included "Notes on the Evolution of a FORTH Programmer" by Charles Moore, in which he described how his own programming style had matured.

On the final day the question was brought up of whether FORTH was a programming language or a religion. The consensus was: Yes! In the same discussion the expression "born-again programmer" appeared. (It is in competition for catch-phrase of the year with "black-belt programmer", which was heard at the FIG Convention in San Mateo the following day.)

;s G. Maverick

LETTERS

J. E. Rickenbacker pointed out that the JMP ($xxFF) of the fig-FORTH inner interpreter does not work on a 6502.

That is right, but the fig-FORTH compiler automatically tests for this condition and avoids ending a CFA in FF.

The only problem occurs during initial installation when a hand assembly is required. Since 6502 assemblers, unlike FORTH, are inflexible you just have to sit there helpless watching them make the same dumb mistake at each new assembly and then add a correction when the assembler is finished. Since fig-FORTH has about 210 definitions, the chances are pretty good (about 210 out of 256) that a CFA will end in FF.

My advice would be to leave the patch in until the system is pretty well debugged and then install the jump indirect scheme of the fig-FORTH model. It would be a shame to permanently slow down the system unnecessarily because of an initial installation inconvenience which is primarily the fault of the inflexibility of the 6502 assembler.

As to Mr. Rickenbacker's query on a FORTH assembler vocabulary, he may find Programma International's version of APPLE-FORTH helpful. The system isn't FORTH, it is something like FORTH. However they have a FORTH-like assembler in their system which may be helpful. The op-codes have been analyzed for postfix operation, etc.

FORTH is beautiful.

Edgar H. Fey Jr.
La Grange, IL

LYONS' DEN

In the course of implementing the FIG model on my computer I have noticed that the word NOT is in the assembler vocabulary but not in the high level glossary. Instead 0= is used for logical negation in high level code. Defining NOT as a synonym for 0= in the main kernel glossary might be useful. Code would be a little more readable by distinguishing between the operations of testing whether a number on the stack from a mathematical formula is zero, and logically negating a boolean flag left on the stack by a relational operator, even though the code used to perform these two operations is the same. But a stronger need for a high level NOT occurs when floating point or other data types in addition to the standard integer type is implemented by a vocabulary containing redefinitions of the mathematical operators. In that case a new 0= would be defined to test, say, whether a floating point number were zero, and this new 0= could not be used for logical negation. Of course, the existing practice seems to be to define new operators with unique names such as F0= instead of redefining the kernel names, avoiding this problem. Also, a user can always add a synonymous NOT to the FORTH vocabulary before redefining 0= and the other operators in the vocabulary for a new data type. Once using NOT in code written in the terminology of the new vocabulary, however, one might as well use it for code in the kernel terminology as well, and then such could not be compiled by the standard kernel. So, why not add a NOT?

George B. Lyons
Jersey, City, NJ

EMPLOYMENT WANTED

Chairman of the FORTH Bit Slice Implementation Team (4th BIT) desires a junior programmer position working in a FORTH environment. (Also know COBOL & BASIC.)

Contact: Martin Schaaf
202 Palasades Dr.
Daly City, CA 94015
(415) 992-4784 (eves.)

HELP WANTED

HELP 4TH BIT

With the implementation of a FORTH machine in AMD bit slice technology. If you're a hardware or microcode expert we can use your help. (This is a volunteer FORML project.)

Contact: Martin Schaaf
Chairman, 4th Bit
202 Palasades Dr.
Daly City, CA 94015

MEETINGS

LA fig User's Group
October 1980

The LA group continued to experiment with format on its second meeting. It will continue to meet on the fourth Saturday each month at the Allstate Savings and Loan located at 8800 S. Sepulveda Blvd., 1/2 mile north of the LA airport.

The agenda this month called for a FIG meeting at 11, lunch at noon, and a FORML session at 1 patterned after our northern neighbors.

ARE YOU A — — — ·-·- FIGGER?
YOU CAN BE!
RENEW TODAY!

At 11, a 20 minute random access was followed by an introduction by each of the 40 people present. The remaining half hour before lunch was evenly divided between a summary of the FORTH '79 document given by Jon Spencer, and a series of short announcements. These included a reminder about the Asilomar happening, a query about target compilers for the figFORTH environment, a suggestion that the LA and northern CA group exchange copies of notes or handouts from the meetings, a brief interchange of thoughts on program exchange leading to the idea of a uniform digital cassette standard, requests for assemblers and model corrections, and finally a parallel was drawn between the science fiction group's use of an "Amateur press association" as a potentially useful distribution channel.

From 1:15 to 4, Jon Spencer presented a FORML section which covered 3 topics:

1. Language processing.
2. Address binding and examples of a FORTH linker.
3. A continuation of his talk of last month on an algebraic expression evaulator for FORTH.

We all offer our thanks to Phillip Wasson who has organized the LA group. He is available at 213-649-1428 for details of the coming meeting. To get things rolling as far as program and information exchange, I volunteered as the LAFIG librarian. In 2 sessions, this has already expanded to writing a review for FORTH Dimensions and keeping track of spare copies of the handouts. I can be reached evenings from 7 to midnight at 213-390-3851.

;s Barry A. Cole

L.A. fig Meeting
November 1980

The November meeting was slightly smaller and less formal than the preceding meetings. After a short round of introductions, we were treated to a demo of a new set of FORTH system/application tools by the author, Louis Barnett of Decision Resources Corp. He has an Advanced Directory, File, and Screen Editor system which fits on top of fig-FORTH. I have looked at implementing a similar system in the past. He has thought out the tradeoffs of flexibility, speed, and keeping compatible with existing FORTH block formats. He allows the blocks to be interpreted in the traditional manner (by block #), as well as by file name and relative block number. He uses buffer pools and bitmaps to use all available disk space. It keeps a list of block numbers within a named file. Best of all, it allows editing, printing, and compiling by named file. I was sufficiently impressed to buy a copy on the spot.

After lunch, I presented an introduction to a tool I have been working on. It is used to build stack diagrams interactively for screens or colon definitions from the source screen coupled with symbolic element names entered from the console. I will write it up for a future issue of F.D.

;s Barry A. Cole

LA MEETING

The next meeting of the "L.A. FORTH Users Group" will be

at: Allstate Savings & Loan
Community Room
8800 S. Sepulveda Blvd.
Los Angeles, CA
(1/2 mile north of LAX)

January 24, 1980 ("FORTH" Saturday)
11-12 AM General session
12- 1 PM Lunch break
1- 3 PM FORML Workshop

Info: Philip Wasson (213) 649-1428

FORML
October 1980

Henry Laxen opened with a discussion session on the problems of teaching FORTH. This produced a number of ideas ranging from subglossaries and reorganizations of glossaries, to comments on style and the categorization of tools. An anecdote by Kim Harris described a class of experienced FORTH programmers all FORTHing a traffic intersection problem only to be startled to discover that Charles Moore's solution used no IFs (the dictionary already is a link of IFs !)

Northern California
November 1980

FORTH-79 STANDARD: Bill Ragsdale summarized details of the just-published standard which had been worked out last year at Catalina Island. Handed out was a FORTH-79 Standard HANDY REFERENCE card and a two-page FORTH-79 Standard Required Word Set and requirements sheet with system errors and errors of usage specified. About vocabulary chaining, Bill mentioned the European approach--dynamic and oneway. In contrast, FORTH, INC. has a 4 level chain and the FORTH-79 Standard uses explicit chaining by vocabulary-name invocation.

Handouts included a FORTH machine proposal by Martin Schaaf, Ragsdale's CASE statement, a workshop announcement (for December) and Product Reviews of Laboratory Microsystems' Z-80 fig-FORTH and SBC-FORTH from Zendex Corp., by C.H. Ting. Introductions included:

- Sam Bassett is writing a text on FORTH For Beginners.
- Kim Harris' Humbolt State Univ. class will be held the week of 23-27 March.
- Ron Gremban offered a 4th programming job.
- FORTH will be mentioned in the next WHOLE EARTH CATALOG.
- Bill Ragsdale had been elected to the Board of Directors of FORTH, INC.
- Future fig meetings will be held underneath Penneys just East of Liberty House, Hayward.

;s Jay Melvin

FORML
November 1980

FORML - Klaus Schleisiek spoke about his FORTH implementation of an audio synthesizer which we heard on a cassette recording. The input device has a lightpen and output was by 64 speakers. Digital counters were organized in a linked list of registers comprising a table of sounds searched by NEXT. The structure of Klaus' program was depicted in discussion and on a half dozen xeroxed screens.

Northern California
October 1980

Bok Lee described STOIC, "A baroque elaboration of FORTH". This dialect differs from figFORTH by virtue of its third stack (Loopstack for I parameters) and its 4th stack which handles up to four vocabularies used, a compile buffer (which can be simulated by :: definitions in FORTH) and by its file system which is not screen-dependent but of indefinite length. The STOIC presentation was followed by a panel debate consisting of Kim Harris, Bob Fleming, Dave Bolton, Bill Ragsdale and B.W. Lee where it was unanimously decided "to each his own". General agreement was made about STOIC or FORTH's ability to simulate features of each other. The following differences seemed noteworthy:

- STOIC has some old style (FORTRAN?) mechanisms reflecting author Sack's incomprehension of some of author Moore's concepts.
- STOIC is conceptually not verbal, as is FORTH.
- STOIC is very well documented!
- STOIC is not supported by a group (like fig) and, consequently,
- STOIC is not portable.

Mr. Bok's handouts included a (sample) DUMP program, NORTHSTAR and CP/M memory maps for STOIC and a decompiler. Other meeting handouts included a structured (FIND) by Mike Perry (which appears to be 8080 coded), a 6502 assembler with heavy commenting by Tom Zimmer as well as Zimmer's ad for tiny PASCAL, ROM and disk based OSI FORTH and Asilomar FORML details. C.H. Ting introduced his just published FORTH SYSTEMS GUIDE, which is enlightening. Sam Daniel volunteered to take on my scribeship abandoned due to marriage and relocation in L.A.

;s Jay Melvin

FORTH COURSE

PEOPLE, COMPUTERS, AND
FORTH PROGRAMMING

DATE

March 23-27, 1981

COURSE

The course is an intensive five day program on the use of FORTH. Topics are to incluse 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 used for demonstrations and class exercises. Due to class size limitations only twenty participants will be permitted. Please register as soon as possible but no later than March 1, 1981. The cost will be $100, or $140 with 3 units of credit. The manual "Using FORTH" will be available for an additional $25.

Send payment to:

Barbara Yanosko
Office of Continuing Education
Humboldt State University
Arcata, CA 95521

LOCATION

Humboldt State is located in Arcata, California, six miles north of Eureka and about 300 miles north of San Francisco. Arcata has bus and plane service from San Francisco and Portland. Motels are available for lodging. Transportation will be available from the local "Motel 6". Other motels are within walking distance. For reservations, contact:

Motel 6
4755 Valley West Blvd.
Arcata, CA 95501

INFORMATION

For other information contact:

Professor Ronald Zammit
Physics Department
Humboldt State University
Arcata, CA 95521

(707)826-3275
(707)826-3276

FIG CONVENTION

The second annual FIG Convention was a big success with 250 FORTH users, dealers, and enthusiasts attending a full day of sessions on FORTH and FORTH-related subjects. The Villa Hotel in San Mateo, CA provided the setting this year.

In the annual report, Bill Ragsdale mentioned some of the milestones passes by FIG in 1980:

1. Total membership is now 2,044. About 1200 new members joined this year, primarily due to the BYTE issue devoted to the FORTH language.

2. Roy Martens was hired this year as the full-time publisher of FORTH DIMENSIONS, and is also taking over responsibility for all mail-order and telephone inquiries.

3. The first college-level course in FORTH was taught by Kim Harris in 1980. Another course, to be offered in 1981, will give college credit for completion.

4. The FORTH-79 Standard was approved just prior to the convention, and copies are available through FIG mail order.

5. Regional groups are springing up all over the U.S. New groups are now meeting in Los Angeles, Boston, Dallas, San Diego, San Francisco, and approximately 20 other cities across the country.

Following a panel session on the FORML conference at Asilomar, Charles Moore of FORTH, Inc., closed the morning session with a reminder that it is the very flexibility and versatility of FORTH which will cause more problems as more people become acquainted with it. In particular, we must be able to demonstrate to large mainframe users that FORTH is also applicable in their environment.

The afternoon session was highlighted by two very interesting presentations. The first was on software marketing, pointing out very clearly the differences in professional and amateur approaches to selling of software. The second presentation was by Dr. Hans Nieuwenhuizen, of the University of Utretch in Holland, regarding the implementation of High Level Languages in FORTH. Dr. Nieuwenhuizen reported running BASIC, PASCAL, and LISP systems, written entirely in FORTH, at the University of Utretch. (Please do not write Dr. Nieuwenhuizen concerning availability of this software. When it is ready for distribution, an announcement will be made through FORTH DIMENSIONS.)

The formal part of the convention concluded with presentations from some of the many vendors of FORTH systems and software.

After a short interlude for informal discussion and attitude adjustment, Mr. Allen Taylor, author of the Taylor Report in ComputerWorld, was the guest speaker at the now-traditional evening banquet.

;s S. Daniel

MORE LETTERS

Since I seem to be the first OSI user to have the FIG model installed and fully operational, I thought that you might add my company name to your list of vendors. I have been extremely faithful to the model, changing only the I/O and -DISC. Everything works just fine, and by that I mean a lot better than OSI's standard system software. I did find a miscalculated branch (forward instead of backward) and the address of ;S was left off of the end of UPDATE (with lethal result) in case you are interested. Unfortunately, I couldn't use the ROM monitor for MON since it blows out the OSI DOS. Instead MON jumps to the DOS command interpreter, which is more useful than the OSI ROM monitor, anyway.

I feel that I am in a position to fully support the system, since I know OSI's hardware and DOS inside-out, and also it appears that I may have their (OSI's) cooperation and even mention in future advertisements.

I have enclosed a press release which describes system requirements, ordering information, and price.

Guy T. Grotke
San Diego, CA

We are soliciting comments, suggestions and bug reports concerning the fig-FORTH 8080 source listing. Work on converting this to the 1979 Standard will begin in early Februry, 1981, so please make submissions as soon as possible to:

8080 Renovation Project
c/o FORTH Interest Group
P.O. Box 1105
San Carlos, CA 94070

Terry Holmes

Dear FIG (Whoever you are),

Just a little note to let you know that I received all the FIG material that I ordered. I would like to know if the 8080 listing is available on IBM formatted single density 8" diskettes and if the fig-FORTH model listed in the Installation Manual, i.e., Screen Nos. 3-8, 12-80, 87-97, is also available on an IBM format 8" single density diskette? I don't relish having to type in all that material, to get fig-FORTH up and running.

I have taken the liberty to spread the word about fig-FORTH in my computer club and have attached copies of two of our newsletters, in which reference to it has been made, see VCC NL Issue 109- bottom p. 3 and Issue 112- middle p.2.

S. Lieberman
Valley Computer Club
P.O. Box 6545
Burbank, CA 91510

(An 8080 figFORTH system on 8" diskette for CP/M systems is available from Forthright Enterprises - P.O. Box 50911, Palo Alto, CA 94303 -- Ed.)

Here is a program you are welcome to publish in FORTH Dimensions.

Lyall Morrill
San Francisco, CA

```
SCR # 222
 0 ( ENIUQ A 'Self-Rep' after Douglas R. Hofstadter 17 AUG 80 )
 1
 2      FORTH DEFINITIONS  DECIMAL
 3
 4 ( Self-reproducing source code in two lines of fig-FORTH.  When
 5 loaded, types a copy of itself.  Note the 128th character. )
 6
 7
 8 : ENIUQ CR 34 WORD HERE COUNT 2DUP TYPE CR TYPE 34 EMIT : ENIUQ
 9 : ENIUQ CR 34 WORD HERE COUNT 2DUP TYPE CR TYPE 34 EMIT : ENIUQ"
10
11
12 ( See "Godel, Escher, Bach:  An Eternal Golden Braid,"
13 by Douglas R. Hofstadter, Basic Books, Inc., 1979, page 498. )
14
15      ;S   LLM
```

Just a line to let you know of a couple of FORTH activities at this end of the country. Here at Temple U we have a lab equipped with 25 AIM systems. Microprocessor Systems is a 56-hour lecture / 28-hour hands-on course of which about 12/6 hours are allotted to AIM Assembler.

I am now testing both Rehnke's V 1.0 FORTH cassette and Rockwell's V 1.3 FORTH ROM chips. I expect to teach one or the other in place of the AIM Assembler this term.

On March 21st the IEEE UPDATE Committee is running an all-day tutorial on FORTH. At that time I hope to demonstrate FORTH transportability between, say, AIM and PET or Apple. I wonder whether anything has been published on this sort of demonstration.

Karl V. Amatneek
Director of Education
Committee for Professional
UPDATE
Wyndmoor, PA

(No, but if you'd like to write one... Ed.)

I get a great deal of your mail. I work for GTE LENKURT, 1105 Old County Road, San Carlos. Those idiots in the post office can't distinguish that from P.O. Box 1105 and our names are not that dissimilar, I guess.

Please get another box number.

M. Mohler
San Carlos, CA

(Guess we're TOO popular -- Ed.)

The video editor presented as an example of CASE use by Major Robert Selzer in FORTH DIMENSIONS v. II/3, p. 83 is super.

Enclosed is a direct extension of Major Selzer's work to edit ASCII files over several consecutive screens. It is used in the form:

```
n1 n2 FEDIT
```

where n1 is the first screen in the file and n2 is the last.

FEDIT contains all the commands of Major Selzer's VEDIT and works in the same manner. ESC exits the editor and the cursor position is controlled by the single keystrokes LEFT, RIGHT, UP, DOWN AND RETURN. When the top or bottom boundary of the display is reached a new display of either the next or the previous 24 lines in the file is presented for editing.

The added commands are RUB which deletes characters and two double keystroke commands HOME and TAB.

HOME followed by DOWN or UP produces a display of the next or previous 24 lines respectively independent of the position of the cursor. Two successive strokes of HOME produce a new display with the line containing the cursor in the old display at the center of the new display. These commands provide rather rapid traversal of a file and positioning of the file on the display.

At the end of a file, additional numbered but blank lines may be displayed. Text written into this area will not be put into the buffer. Similarly if the first line of the file ends up in the middle of a display, the area above the first line is protected.

The TAB key is used to erase, delete and replace lines from PAD. TAB followed by E erases the line containing the cursor. TAB followed by D erases the line and holds the line in the text output buffer PAD. The cursor may then be moved to any position in the file, including other screens, and the contents of PAD may be put on the new line by the key-strokes TAB then P. TAB followed by H places the cursor's line in PAD without deleting the line. These commands use the fig-FORTH line editor definitions E, D, R (REPL in the listing) and H.

Major Selzer's definition of CASE does not work in fig-FORTH with its compiler security features. An appropriate definition of his CASE word for fig-FORTH is shown on line 10. The word OFF on line 68 controls a switch in my EMIT to stop output to my printer. All other words should be standard fig-FORTH. The terminal dependent cursor position sequence used by Selzer for his ADM-3A terminal (YXCUR, line 3) also works on my SOROC IQ 120 terminal.

I have found FEDIT to be a convenient editing tool which I use along with the fig-FORTH editor. Eventually, I suppose, my entire fig-FORTH editor will find its way into FEDIT. I hope your readers will also find it convenient. I also hope FEDIT lays to rest some of the recent criticism of FORTH (in BYTE) concerning its rudimentary editing facilities. My thanks to all of you in FIG for your efforts in promotion FORTH.

Edgar H. Fey
LaGrange, IL

```
SCR # 64
 0 ( ASCII FILE EDITOR  SCR 64 TO 68     E H FEY Corr 11/2/80)
 1   0 VARIABLE CUR  0 VARIABLE COFF  0 VARIABLE N2  0 VARIABLE N1
 2
 3 : YXCUR ( x y ... ) 27 EMIT 61 32 + EMIT 32 + EMIT ;
 4 : .CUR  ( ... ) ( Print cur ) CUR @ 64 /MOD SWAP 4 + SWAP YXCUR ;
 5 : !CUR  (n ... ) ( Store n in cur ) 0 MAX 1535 MIN CUR ! ;
 6 : +CUR  (n ... ) ( Add n to cur ) CUR @ + !CUR ;
 7 : +.CUR (n ... ) (Add n & print cur ) +CUR .CUR ;
 8 : HOM   ( ... ) ( Reset cur ) 0 CUR ! ;
 9 : (CASE) OVER = IF DROP 1 ELSE 0 THEN ;
10 : CASE COMPILE (CASE) [compile] IF ; IMMEDIATE
11
12 : .LB ( l blk ... ) ( Print line l of block blk if blk in file )
13   DUP N1 ! < OVER N2 @ > OR ( ...l blk blk>n20 Rblk<n1 )
14   IF ( Not in file ) DROP DROP 68 0 DO 32 EMIT LOOP
15   ELSE ( In file ) DUP >R (LINE) TYPE R> 124 EMIT . THEN ;        -->

SCR # 65
16
17 : .NL (n l b ... ) ( Print n lines from line l relative to l in )
18   ( 0 of block b ) OVER DUP >R 16 /MOD   ( ...n l b rem1 l/16 )
19   R> 0< IF ( l<0 ) ROT + -1 + SCR ! 16 + ( ...n 1 rem1+16 )
20   ELSE ( l>=0 ) ROT + SCR !              ( ...n 1 rem1 )
21   THEN  SWAP ROT OVER >R + >R            ( ...rel l+n ! )
22   DO ( n lines )                         ( ...rel )
23   CR I 3 .R SPACE DUP SCR @ .LB 1+ /MOD SCR +! LOOP DROP ;
24
25 : ACUR ( ... acur ) ( Abs cursor addr in file ) CUR @ COFF @ + ;
26 : (+LIN) ( n ... cur ) ( Computer CR+nLF ) CUR @ 64 / + 64 * ;
27 : ACMX ( ...acurmax ) N2 @ N1 @ - 1+ B/BUF * ;
28
29 : .HOME ( n ... ) ( New display, line of old cursor at line n )
30   ACUR 64 / OVER - 24 OVER N1 @ .NL
31   64 *    COFF !  HOM 64 * +CUR ;                                 -->

SCR # 66
32 : .TOP ( n... ) COFF @ 0= IF DROP ELSE +CUR 23 .HOME THEN ;
33
34 : +.ACUR ( n ... ) ( ADD n to abs cursor.  Display cursor )
35   DUP CUR @ + DUP 0<                     ( ... n n+cur f )
36   IF ( OFF top ) DROP .TOP               ( ... )
37   ELSE ( Not off top ) 1535 >            ( ...n n+cur>1535 )
38      IF ( Off bottom ) +CUR ACUR ACMX < IF 0 .HOME THEN
39      ELSE ( In display ) +.CUR  THEN THEN ;
40
41 : +LIN ( n ... ) ( Add n lines to cur, CR+nLF ) (+LIN) !CUR ;
42 : +ALIN ( n ... ) ( Add n lines to curs ) (+LIN) CUR @ - +.ACUR ;
43 : BOX ( ...f ) ( True if in file ) ACMX ACUR > ACUR -1 > AND ;
44 : !ABLK ( c... ) ( Store ascii char in buffer ) BOX IF ( In file )
45   ACUR B/BUF /MOD N1 @ + BLOCK + C! UPDATE !  +.ACUR THEN ;
46 : RUB 32 DUP EMIT !ABLK -2 +.ACUR ; ( Del char, retreat curs )
47 : ALIN ( ...l ) ( Get cursors scr and line                       -->

SCR # 67
48   ACUR B/BUF /MOD N1 @ + SCR ! 64 * ;
49 : E ( ... ) ( Display blank line )
50   0 +ALIN 64 0 DO 32 EMIT LOOP 0 +ALIN ;
51 : .P ( ... ) ( Replace line from PAD at cur line & display )
52   ALIN REPL 0 +ALIN ALIN SCR @ .LB 0  +.ACUR ;
53
54 : TAB2 ( ... ) ( 2nd key stroke for choice of TAB )
55   KEY 69 CASE ALIN E .E          ELSE ( E=Erase curs line )
56       72 CASE ALIN H             ELSE ( H=Hold curs line at PAD )
57       80 CASE  .P                ELSE ( P=Replace line from PAD )
58       68 CASE ALIN H ALIN E .E ELSE ( D=Del line, hole in PAD )
59   DROP 0 +ALIN THEN THEN THEN THEN     ; ( Default CR no LF )
60
61 : HOME2 ( ... ) (2nd key stroke choice of HOME ) KEY
62    10 CASE 1535 +CUR 0 .HOME  ELSE ( Down=scroll next )
63    11 CASE -1535 +CUR 23 .HOME ELSE ( Up=scroll prev )             -->

SCR # 68
64    30 CASE       12 .HOME     ELSE ( Home=center cursor )
65   DROP 0 +ALIN THEN THEN THEN :    ( Default= CR no LF )
66
67 : FEDIT ( n1 n2... ) ( Edit file in blocks n1 to n2 incl. )
68   N2 ! N1 ! OFF 0 COFF ! HOM 0 .HOME BEGIN
69   KEY 27 CASE    0 23 YXCUR QUIT     ELSE ( ESC )
70        8 CASE     -1  +.ACUR        ELSE ( LEFT )
71       10 CASE     64  +.ACUR        ELSE ( DOWN )
72       11 CASE    -64  +.ACUR        ELSE ( UP )
73       12 CASE      1  +.ACUR        ELSE ( RIGHT )
74       13 CASE      1  +ALIN         ELSE ( RETURN )
75      127 CASE        RUB            ELSE ( RUB )
76        9 CASE     7 EMIT TAB2       ELSE ( TAB, Next key picks )
77       30 CASE     7 EMIT HOME2      ELSE ( HOME, Next key picks )
78   DUP EMIT !ABLK THEN THEN THEN THEN THEN THEN THEN THEN THEN
79   AGAIN ;                                                        ;S
```

THE FORTH SOURCE

A wide variety of FORTH printed material, both public domain and copyrighted, is available. Send for list:

Mountain View Press
PO Box 4656
Mt. View, CA 94040

NEW PRODUCTS

6800 & 6809 FORTH

† FORTH	
FORTH System	$100
† FORTH+	
plus Assembler, CRT Editor	$250
firmFORTH	
produces compacted ROMmable code	$350

Kenyon Microsystems
3350 Walnut Bend
Houston, Texas 77042
Phone (713)978-6933

CRT EDITOR AND
FILE MANAGEMENT SYSTEM

The Decision Resources File Management System (FMS-4) for the FORTH language has extensive vocabulary for creating, maintaining and accessing name files.

Disk space is dynamically allocated and deallocated so there is never any need to reorganize a disk. From the user viewpoint, access is to logical records; FMS-4 performs the mapping to physical screens.

Files may be referenced by name without concern for the physical location of the file on disk. FMS-4 supports sequential and direct access while preserving FORTH's facilities for addressing screens by number.

FMS-4 maintains a file directory of up to 47 entries. Each file may consist of from one to 246 records (1024 bytes per screen) in a single volume (single density diskette). It is also possible to extend FMS to control multiple volume files and to support larger directories.

In addition to an extensive command set, there are many lower level primitives which may be combined to define a virtually unlimited set of commands.

Computer system hardware should include:

One or more 8" IBM compatible floppy disk drives

Enough memory to support 6K bytes (on an 8 bit processor) for FMS-4 in addition to the FORTH nucleus and any other concurrently resident applications.

An 8080/8085 or Z80 cpu.

A CRT or printing terminal which supports upper and lower case.

System software should include:

fig-FORTH compatible nucleus or equivalent.

An assembler for the target cpu. DRC can supply an 8080 assembler at additional cost.

FMS-4 source code is delivered ready to run (on compatible systems) on a single density 8" soft sectored diskette (IBM 3740).

A complete user manual describing all facets of FMS-4 operation is provided. The manual includes an extensive glossary which defines and documents the usage of each word in the FMS vocabulary.

Wordsmith is a CRT screen editor which is integrated with Decision Resources' File Management System - FMS-4. The combination is an especially powerful file oriented editor which combines the extensive disk space management facilities of FMS-4 with the flexibility and immediacy of on-screen editing.

The full record being edited is continuously displayed on the CRT and all changes are immediately visible. There are 41 editing commands including: multidirectional cursor movement, record to record scrolling, record insert and delete, string search and replace, text block movement and many more.

The FMS-4 and Wordsmith Packages

Wordsmith and FMS-4 source code is delivered ready to run (on compatible systems) on a single density 8" soft sectored diskette (IBM 3740).

Complete user manuals for each system are provided.

Pricing

Single noncommercial user license:	
FMS-4	$50
Wordsmith (with FMS-4)	$95
Manual only:	
Wordsmith	$15
FMS-4	$15
Both	$25

(credited toward purchase of full package)

California residents add 6% sales tax
Shipping and handling: $2.50

Commercial Purchasers should contact Decision Resources.

Decision Resources Corporation
28203 Ridgefern Court
Rancho Palos Verdes, CA 90274
(213) 377-3533.

TRS-80 DISKETTES

Advanced Technology Corp. of Knoxville, TN, is presently distributing its fig-STANDARD FORTH version (TFORTH) customized for the Radio Shack TRS-80. Included in this package are: assemblers, 'TRACE' function for generating minimum system /CMD files, POINT, SET, CLS commands for graphics use, Floating point package, I/O package (LPT Output) and variable number base to base 32.

The language is supplied on either 80 or 40 track 5-1/4" diskette for $129.95 and the manual is also included.

This product may be purchased from:

Sirius Systems
7528 Oak Ridge Highway
Knoxville, TN 37921

or

QC Microsystems
P.O. Box 401326
Garland, TX 75040

or directly from us,

Advanced Technology Corp.
1617 Euclid Avenue
Knoxville, TN 37924
(615) 525-1632

PRODUCT REVIEWS

by
C.H. Ting

Z-80 fig-FORTH by Ray Duncan of Laboratory Microsystems, 4147 Beethoven St., Los Angeles, CA 90066 (213) 390-9292.

Two 8" single density diskettes, $25.00.

The first disc is a CP/M disc containing Z-80 assembly source codes, hex object codes, user instructions, fig-FORTH Installation Manual, and fig-FORTH Glossary. The second disk is in FORTH block format containing system configurations, a line editor, a poem 'The Theory That Jack Built' by F. Winsor, Eight Queens Problem by J. Levan, Towers of Hanoi by P. Midnight, Breakforth by A. Schaeffer, and some utilities.

I do not have a system that can run the Z-80 codes. However, the source codes seem to be carefully done and follow faithfully the fig-FORTH 8080 model. Lots of typing was put in to have the entire Installation Manual and Glossary entered on disc. The games were published in FORTH Dimensions. The amount of information offered at this price is unbelievable. I just wish that I had a machine that could run it.

SBC-FORTH from Zendex Corp., 6398 Dougherty Rd., Dublin, CA 94566 (415) 829-1284.

Four 2716 EPROM's to run in an SBC-80/20 board with SBC-201 single density disk. $450.00.

I had the PROM's installed in a System 80/204. It ran only after I jumpered the CTS/ and RTS/ pins of the 8251 serial I/O chip. Obviously the chip uses some interrupt scheme to drive the terminal. I was not able to get the detailed information on how the interrupts were supposed to go from Zendex. I do not have a disc drive in the system to test out the disc interface. Other things ran satisfactorily. I was able to talk to the parallel I/O ports using the assembler.

This type of ROM based FORTH machine can be very powerful for programmable controllers and low cost development systems if some non-volatile memories like core or battery-backed CMOS were added.

A very nice thing they did in the manual was to include the code or colon definitions in the Glossary, making it infinitely more useful as a reference.

NEW PRODUCTS

APPLE figFORTH

Including an Assembler, Screen Editor, Source Code and associated compiler, with some documentation on disk. No other documentation, support or instruction. Source listing will be available from fig in mid-81. Apple format disk - $30.00.

George Lyons, 280 Henderson St., Jersey City, NJ 07302.

CROMEMCO DISKETTE

A fig-FORTH 5-1/2" disk with Z80 assembler for Cromemco machines. $42.00

Nautilus Systems
PO Box 1098
Santa Cruz, CA 95061

NEW PRODUCTS

"Systems Guide to fig-FORTH"

Author: C.H. Ting
156 14th Ave.
San Mateo, CA 94402

132 pages, $20.00

This book is meant to be a bridge between "Using FORTH" and the "fig-FORTH Installation Manual", and to serve as a road map to the latter. It might also be used as a collection of programming examples for those studying "Using FORTH".

In it, I have tried to arrange the fig-FORTH source codes into logical groups: Text Interpreter, Address Interpreter, Error Handler, Terminal I/O, Numeric Conversions, Dictionary, Virtual Memory, Defining Words, Control Structures, and Editor. Extensive comments are thrown in between source codes at the risk of offending the reader's intelligence. Occasionally flow charts (horror of horrors!) are used to give graphic illustrations to some complicated words or procedures.

There is a very wide gap between the front page and the back page of the FORTH Handy Reference Card. It is relatively easy to manipulate the stacks and to write colon definitions to solve programming problems. The concepts behind words of system functions, like INTERPRET, [,] , COMPILE, VOCABULARY, DEFINITIONS are very difficult to comprehend, not to mention <BUILDS and DOES> . One cannot understand the FORTH system and how it does all these wonderful things by reading the source codes or by searching the glossary. These documents are vehicles to define the FORTH system, not to promote understanding of them.

OSI DISC

Tiny PASCAL written in fig-FORTH. Machine Readable for OSI-C2-8P. Single or Dual Floppy System 8" disc.
Cost: $60.00
This includes fig-FORTH with fig editor and assembler for FREE!

OSI C2 or C3 fig-FORTH on 8" disc.
Cost: $45.00
Includes assembler and fig editor

Tom Zimmer
292 Falcato Dr.
Milpitas, CA 95035
(408) 245-7522 ext. 3161 or
(408) 263-8859.

---

tinyPASCAL

Printed listing of tinyPASCAL in fig-FORTH.

$10.00 US/Canada, $14.00 Overseas. Check (US bank), VISA or Master Charge.

Mountain View Press
PO Box 4656
Mt. View, CA 94040

FORTH Version 1.7

Cap'n Software FORTH Ver. 1.7 for Apple II (TM) or Apple II+ computers is the FORTH Interest Group (FIG) language, plus extensive program development tools and special Apple options. $175.00.

Cap'n Software
P.O. Box 575
San Francisco, CA 94101.

SEPARATED HEADS

Klaus Schleisiek

Memory in RAM-based systems can be used more efficiently by means of a "Symbol Dictionary Area," which allows words and/or name and link fields which are needed only at compile time to be thrown away after compilation. Incremental use of these techniques will result in more efficient memory usage and will also encourage the use of more and shorter definitions because there is no longer the need to pay the penalty of taking up memory space with numerous name and link fields.

In the course of a two-year project I developed some tools which allow a significant compression of code in RAM-based systems. I also feel these methods will have a significant impact on programming style, particularly because they will encourage the use of more and shorter definitions. The following is an explanation of these various functions in a somewhat historical order.

My programming task was to develop a lightpen-operated sound system, which would allow control of a number of small sound synthesizers by pointing a lightpen to various dots, light potentiometers, and the like on a video display. There was to be no keyboard intervention. A "dot" was put together by compiling a word which associated the following information:

A) The shape of the dot itself as an address of some programmable character.

B) The dot's location on the screen as an address relative to the upper left hand corner of the screen.

C) As an option, either a text string or a string of programmable graphics characters to be displayed above, below, or to either side of the dot.

Thus, every "dot" served a double purpose. On the one hand, it described a portion of the display itself which had to flash on the screen. Secondly, it supplied the key to a large keyed CASE statement which associated the dot with the function to be performed when the lightpen was pointed to it. In other words, the definitions of the dots themselves were only needed at compile time.

The dot definitions were used to create a densely packed "image" definition to flash the picture on the screen, while the addresses of the dot locations were used as keys in the CASE statement. So, to be memory efficient, I wanted to set up some mechanism which would allow the presence of "symbols" at compile time that could then be thrown away after compilation to free memory. By "symbol" I mean any legal FORTH definition that is only needed at compile time. This led to the idea of dividing the dictionary into "main dictionary" and " symbol dictionary."

Figure 1 shows the arrangement of this scheme based on the 6502's unique memory mapping.

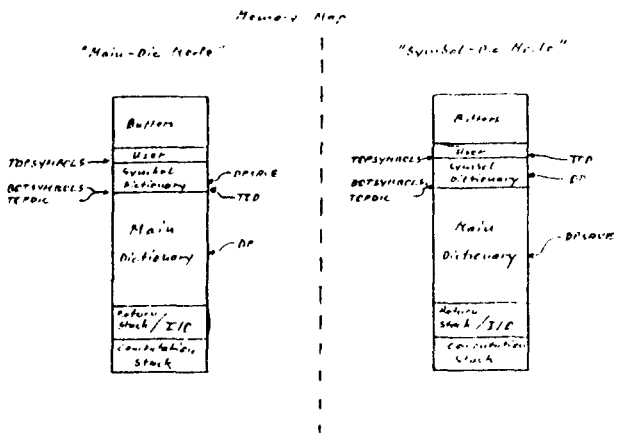

Figure 1

I soon realized that most of the words defined in my programs would never be used again after compilation and started thinking about putting the name and linkfield (head) of a definition into the symbol dictionary, and compiling the code field and parameter field only into the main dictionary. I wanted to do it in a fashion similar to SYMBOL DIC and MAIN DIC .

This would mean switching back and forth between one state which compiles the heads into the symbol dictionary and another state which compiles the heads as usual. This switching is done by the variable HEADFLG (SCR #23) which is respectively set and reset by DROP-HEADS and COMPILE-HEADS (SCR #23). The state of HEADFLG in turn changes the behavior of CREATE (SCR #24).

One complication is that the use of HEADFLG interferes with the symbol dictionary mechanism: If you are compiling into the main dictionary, you want the dropped heads to be compiled into the symbol dictionary, but if you are compiling into the symbol dictionary anyway, you want the heads to go there too.

In other words, in the first case the body of a definition would be separated from the head, while in the second case, body and head would not be separated. This requires the redefinition of CREATE (SCR #24) and the use of three values for HEADFLG. The first two states are set explicitly by COMPILE-HEADS and DROP-HEADS, but the third state is recognized and handled by CREATE .

When a word is compiled, its name field and link field are compiled into the symbol dictionary and the word is made immediate and (CFA) is compiled as its code field, followed by the address of the next memory location in the main dictionary. The remainder of the current definition (the body) will then be compiled into the main dictionary. When references are made to the word, its CFA is contained in the memory location next to the code field address of (CFA) .

The function of (CFA) (SCR #23) is either to compile the execution address of code into the dictionary (when the word is subsequently used in a definition), or to execute the definition, depending on STATE, before forgetting the symbols. The implementation described here deals with the 6502 and has to deal with the idiosyncrasy that no CFA may be located at XXFF, which in turn makes the definition of (CFA) and CREATE somewhat mysterious!

FORGET-SYMBOLS (SCR #22) is the word which "rolls" through every dictionary and "unlinks" every definition which was placed in the symbol dictionary, thereby freening it (Figure 2). It is somewhat slow and it is assumed that no symbol exists below FENCE @ . Before forgetting anything in the main dictionary, however, you must FORGET-SYMBOLS . Otherwise links may be broken and the interpreter won't work anymore.

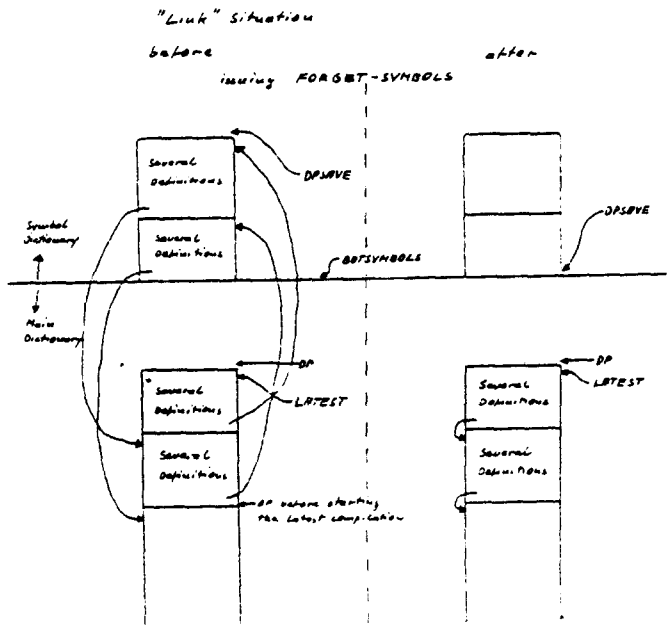

Figure 2

The next step was to make the defining words work as well in the DROP-HEADS mode, which meant that (;CODE) had to be redefined (SCR #25). It now uses the subdefinition (;COD) and depending on the state of HEADFLG, determines the location of the code field to be rewritten and rewrites it.

A problem might arise in the rare case where a definition whose head is to be thrown away is supposed to be immediate by itself. The "solution" to this problem was to simply declare such a case illegal. There is a reason, however. The only situation where one might want an immediate definition to be placed in the main dictionary would be in coincidence with [COMPILE] within some definition. Otherwise, one would want to compile it entirely into the symbol dictionary anyway. Such a case is so rare that it did not seem worth the effort to redefine IMMEDIATE and [COMPILE] . To use a word whose head has been compiled in to the symbol dictionary immediately within a definition, one has to use .... [XXXX] .... !

Finally, I observed that I was generating <BUILDS ... DOES> and ;CODE constructs with big compile time definitions, which do nothing but take memory space at execution time. But dropping the heads of <BUILDS ... DOES> means that the compile time parts of these definitions won't ever be used at compile time either. Thus, if the heads of <BUILDS ... DOES> are dropped, everything prior to DOES> may be dropped as well. It will, however, be necessary to redefine DOES> and ;CODE to do this (SCR #26 and SCR #27).

At compile time, the situation of a <BUILDS ... DOES> construct is as follows: While in the DROP-HEADS state, the name has been put into the symbol dictionary and subsequently <BUILDS ... has been compiled into the main dictionary. When we come to DOES> everything which had been compiled into the main dictionary, including the code field, must be moved into the symbol dictionary.

This is done by MOVE-DEF? (SCR #25), which is used in DOES> and ;CODE . Depending on the state of HEADFLG , MOVE-DEF? either compiles (;CODE) or moves the previous definition into the symbol dictionary and compiles ((;CODE)) . ((;CODE)) has to be one step more indirect than (;CODE) and resembles the function of (CFA) in ordinary definitions.

A final note: The definitions for GOTO and LABEL, which allow multiple forward references (e.g., several GOTO's) may precede as well as follow "their" label. Even though I implemented this because it seemed more convenient than restructuring, there is some question as to its true value because it takes 318 bytes!

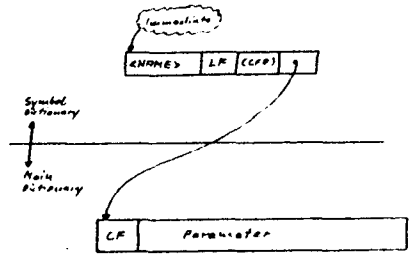

Figure 3

GLOSSARY

```
TOPDIC
        A CONSTANT THAT LEAVES THE NEXT BUT LAST ADDRESS
        TO BE USED AS MAIN DICTIONARY ON THE STACK.
BOTSYMBOLS
        A CONSTANT THAT LEAVES THE FIRST ADDRESS TO BE USED
        AS THE SYMBOL DICTIONARY ON THE STACK.
TOPSYMBOLS
        A CONSTANT THAT LEAVES THE NEXT BUT LAST ADDRESS
        TO BE USED AS SYMBOL DICTIONARY ON THE STACK.
DPSAVE
        A VARIABLE THAT CONTAINS THE DICTIONARY POINTER OF THE
        CURRENTLY INACTIVE DICTIONARY PARTITION.
TOD     ( TOP-OF-DICTIONARY )
        A VARIABLE THAT CONTAINS THE CURRENT NEXT BUT LAST
        MEMORY LOCATION TO BE USED FOR COMPILING DEFINITIONS.
MAIN-DIC
        RESETS DP TO POINT TO THE NEXT FREE MEMORY-LOCATION
        IN THE MAIN DICTIONARY.
        I.E. THE FOLLOWING DEFINITIONS ARE PERMANENTLY
        COMPILED INTO THE MAIN DICTIONARY.
        IF DP WAS ALREADY POINTING INTO THE MAIN DICTIONARY,
        IT DOESN'T DO ANYTHING.
        COUNTERPART: "SYMBOL-DIC"
```

```
SYMBOL-DIC
        SETS DP TO POINT TO THE NEXT FREE MEMORY LOCATION
        IN THE SYMBOL DICTIONARY.
        I.E. THE FOLLOWING DEFINITIONS WILL BE COMPILED INTO
        THE SYMBOL DICTIONARY AND MAY BE FORGOTTEN USING
        "FORGET-SYMBOLS" WITHOUT AFFECTING THE MAIN DICTIONARY.
        IF DP WAS ALREADY POINTING INTO THE SYMBOL DICTIONARY,
        IT DOESN'T DO ANYTHING.
        COUNTERPART: "MAIN-DIC"

FORGET-SYMBOLS
        IS USED FOR UNLINKING THOSE DEFINITIONS WHICH HAD
        BEEN COMPILED INTO THE SYMBOL DICTIONARY FROM
        THE MAIN DICTIONARY DEFINITIONS.
        RESETS THE SYMBOL DICTIONARY POINTER TO "BOTSYMBOLS".
     WARNING: IF ANYTHING HAS BEEN COMPILED INTO THE
        SYMBOL DICTIONARY, YOU HAVE TO 'FORGET-SYMBOLS"
        BEFORE FORGETTING ANYTHING IN THE MAIN DICTIONARY.
HEADFLG
        A VARIABLE THAT CONTAINS THE "HEAD-STATE" I.E.
        HEADFLG = 0  -> COMPILE-HEADS HAD BEEN ISSUED
        HEADFLG = 1  -> DROP-HEADS AND MAIN-DIC HAD BEEN ISSUED
        HEADFLG = 2  -> DROP-HEADS AND SYMBOL-DIC HAD BEEN
                        ISSUED.

COMPILE-HEADS                                           E
        COMPILE THE HEADS ( NAME & LINKFIELD ) OF THE
        FOLLOWING DEFINITIONS INTO THE MAIN DICTIONARY.
        COUNTERPART: DROP-HEADS

DROP-HEADS                                              E
        THE HEADS ( NAME & LINKFIELD ) OF THE FOLLOWING
        DEFINITIONS WILL BE COMPILED INTO THE SYMBOL DICTIONARY,
        THE BODY ( CODE- & PARAMETERFIELD ) INTO THE MAIN
        DICTIONARY. FURTHERMORE, THE COMPILE TIME PARTS
        OF DEFINITIONS IN TERMS OF <BUILDS ... DOES> AND
        ... ;CODE WILL BE COMPILED INTO THE SYMBOL DICTIONARY
        TOO. ( I.E. EVERYTHING PRECEEDING ... DOES> OR
        ... ;CODE RESPECTIVELY WITHIN THE CURRENT DEFINITION )
        THE PARTS WHICH ARE LOCATED IN THE SYMBOL DICTIONARY
        MAY BE FORGOTTEN BY ISSUING "FORGET-SYMBOLS" WHICH
        EFFECTIVELY DISCARDS THE HEADS / COMPILE TIME CODE.
        BEFORE ISSUING "FORGET-SYMBOLS" THESE WORDS MAY BE USED
        IN THEIR USUAL MANNER FOR EITHER COMPILATION INTO
        HIGHER LEVEL DEFINITIONS OR EXECUTION.
     WARNING: DROP HEADS MAY NOT BE USED FOR IMMEDIATE
        DEFINITIONS. NO ERROR CHECKING IS PERFORMED !
```

```
SCR # 20
  0 ( SYMBOLDICTIONARY          KS  10-5-80 )
  1 FORTH DEFINITIONS HEX
  2
  3 3800 CONSTANT TOPSYMBOLS
  4 3000 CONSTANT BOTSYMBOLS
  5 3000 CONSTANT TOPDIC
  6
  7 3000 VARIABLE TOD
  8 3000 VARIABLE OPSAVE
  9
  A : SWITCH-DIC
  B    HERE OPSAVE DUP @ DP ! ! ;
  C
  D : ?MAIN-DIC  (  ---  F-1  )
  E    HERE BOTSYMBOLS U< TOPSYMBOLS HERE U< OR ;
  F -->

SCR # 21
  0 ( SYMBOLDICTIONARY          KS  10-5-80 )
  1
  2 : MAIN-DIC
  3    TOPDIC TOD ! ?MAIN-DIC 0= IF SWITCH-DIC THEN ;
  4
  5 : SYMBOL-DIC
  6    TOPSYMBOLS TOD ! ?MAIN-DIC IF SWITCH-DIC THEN ;
  7
  8 : ?SYMBOL   ( N-1  ---  N-2,FLAG-1  )
  9    BOTSYMBOLS OVER I+ U< OVER TOPSYMBOLS U< AND ;
  A
  B : ?FENCE   ( N-1  ---  N-2,FLAG-1  )
  C    DUP FENCE @ U< ;
  D
  E
  F -->

SCR # 22
  0 ( SYMBOLDICTIONARY          KS  10-5-80 )
  1 : FORGET-SYMBOLS
  2    VOC-LINK @
  3    BEGIN DUP @ >R 2 - DUP >R @
  4          BEGIN BEGIN ?SYMBOL
  5                WHILE PFA LFA @
  6                REPEAT DUP R> !
  7                BEGIN PFA LFA DUP @
  8                      ?SYMBOL SWAP ?FENCE ROT OR 0=
  9                WHILE SWAP DROP
  A                REPEAT SWAP >R ?FENCE
  B          UNTIL
  C          DROP R> DROP R> -DUP 0=
  D    UNTIL
  E    MAIN-DIC BOTSYMBOLS OPSAVE ! ;
  F -->

SCR # 23
  0 ( SYMBOLDICTIONARY          KS  10-5-80 )
  1
  2 0 VARIABLE HEADFLG
  3
  4 : COMPILE-HEADS   ?EXEC 0 HEADFLG ! ;
  5
  6 : DROP-HEADS   ?EXEC 1 HEADFLG ! ;
  7
  8 : (CFA)
  9    0 , HERE 0 , IMMEDIATE
  A    DOES>   @ STATE @
  B    IF , ELSE EXECUTE THEN ;
  C
  D -->
  E
  F
```

```
SCR # 24
  0 ( NEW CREATE          KS  10-5-80  )
  1 : CREATE   HEADFLG @
  2    IF ?MAIN-DIC
  3       IF 2 ELSE SYMBOL-DIC 1 THEN
  4       HEADFLG !
  5    THEN
  6    TOD @ HERE 080 + U< 2 ?ERROR
  7    -FIND IF DROP NFA ID. 4 MESSAGE CR THEN
  8    HERE DUP C@ WIDTH C@ MIN 1+ ALLOT
  9    DP C@ OFD = ALLOT
  A    DUP 080 TOGGLE HERE 1 - 080 TOGGLE
  B    LATEST , CURRENT @ ! HEADFLG @ 1 =
  C    IF 0 HEADFLG ! (;CFA) 1 HEADFLG !
  D       MAIN-DIC DP C@ OFF = ALLOT HERE SWAP !
  E       TOD @ HERE 080 + U< 2 ?ERROR
  F    THEN HERE 2+ , ;  -->

SCR # 25
  0 ( MOVE-DEF?  I.E. THROW AWAY THE <BUILDS-PART    KS  10-5-80  )
  1 : (;CODE)
  2    LATEST PFA HEADFLG @ 1 = IF 4 ELSE CFA THEN ! ;
  3
  4 : (;CODE)    R> (;CODE) ;
  5 : ((;CODE))  R> @ (;CODE) ;
  6
  7 : MOVE-DEF?
  8    HEADFLG @ 1 =
  9    IF SYMBOL-DIC LATEST PFA DUP CFA DP !
  A       @ OPSAVE @ >R DUP OPSAVE ! R> OVER - DUP >R
  B       HERE SWAP CMOVE R> ALLOT COMPILE ((;CODE))
  C       HERE 2 ALLOT MAIN-DIC HERE SWAP !
  D    ELSE COMPILE (;CODE)
  E    THEN ;
  F -->

SCR # 26
  0 ( REDEFINITION OF <BUILDS   DOES>    GRL,KS  10-5-80 )
  1
  2 : <BUILDS
  3    CREATE SMUDGE ;
  4
  5 : DOES>
  6    MOVE-DEF? 020 C, ['] HERE 8 + @ LITERAL , ; IMMEDIATE
  7    ASSEMBLER
  8    PLA, TAY, PLA, N STA, INY, 0=
  9    IF, N INC, THEN,
  A    IP 1+ LDA, PHA, IP LDA, PHA,
  B    IP STY, N LDA, IP 1+ STA,
  C    2 # LDA, CLC, W ADC, PHA,
  D    0 # LDA, W 1+ ADC, PUSH JMP,
  E
  F -->

SCR # 27
  0 ( REDEFINITION OF  ;CODE    KS  10-5-80 )
  1
  2 ;CODE
  3    ?CSP MOVE-DEF? [COMPILE] [ SMUDGE
  4    ?CSP [COMPILE] ASSEMBLER ; IMMEDIATE
  5
  6 ;S
  7
  8
  9
  0

SCR # 28
  0 ( GOTO          KS  10-7-80 )
  1 FORTH DEFINITIONS HEX
  2 DROP-HEADS
  3 : (GOTO)
  4    DOES> DUP @ BEGIN -DUP
  5                  WHILE DUP @ SWAP
  6                        HERE OVER - SWAP !
  7                  REPEAT HERE OVER !
  8                  CFA ['] 0 CFA @ ] LITERAL SWAP ! ;
  9
  A : MOVE-HEAD   (  ---  HERE IN MAIN-DIC-1  )
  B    HERE SWITCH-DIC DUP HERE
  C    OVER C@ WIDTH C@ MIN 1+ DUP >R CMOVE
  D    HERE DUP 080 TOGGLE R> ALLOT DP C@ OFD = ALLOT
  E    HERE 1 - 080 TOGGLE LATEST PFA LFA DUP @ , ! ;
  F -->

SCR # 29
  0 ( GOTO          KS  10-7-80 )
  1 COMPILE-HEADS
  2 : GOTO
  3    COMPILE BRANCH -FIND
  4    IF DROP DUP CFA @ ['] 0 CFA @ ] LITERAL =
  5       IF @ HERE -
  6       ELSE ['] (GOTO) 2+ @ ] LITERAL OVER CFA @ =
  7            IF BEGIN DUP @ WHILE @ REPEAT
  8               HERE SWAP ! 0 ,
  9            ELSE 4 ERROR
  A       THEN THEN
  B    ELSE MOVE-HEAD ['] (GOTO) 2+ @ ] LITERAL , , SWITCH-DIC 0 ,
  C    THEN ; IMMEDIATE
  D -->
  E
  F

SCR # 2A
  0 ( GOTO          KS  10-7-80 )
  1
  2 : LABEL  -FIND
  3    IF DROP CFA DUP @ ['] (GOTO) 2+ @ ] LITERAL =
  4       IF EXECUTE ELSE 4 ERROR THEN
  5    ELSE MOVE-HEAD ['] 0 CFA @ ] LITERAL , , SWITCH-DIC
  6    THEN ; IMMEDIATE
  7
  8 FORGET-SYMBOLS
  9 ;S
  A
  B ( 28 - 2A TAKES 318 BYTES )
  C
  D
  E
  F
```

FORTH IN PRINT

THE TAYLOR REPORT/Alan Taylor

Alternative Software Making Great Strides

Imagine having your own private Cobol compiler -- with special security features and your user application statements -- that you could develop and keep running on your future as well as current hardware. *That* would be a change indeed for any user, and as yet it is still just a dream. But there appear to be no technical reasons and few practical reasons to expect that such a compiler won't be generally available in a year or two.

The Forth Interest Group's (FIG) recent conference showed continued breakthroughs in really opening up software capabilities to users on at least six distinct fronts -- hardware, languages, environments, cross-compiling, research targets and user training. This, only a year after the publication of the first FIG models of the Forth language, showed how some basic knowledge can bear fruit.

The power behind these and other developments has been a growing international group of people and firms. Headed by Chuck Moore's own Forth, Inc., independent user groups in America, Europe and Japan who appreciated the power of Forth have resulted in small commercial ventures with Forth compilers on micros. (The leader here, with more than 100 user groups of its own, is Miller Microsystems, located just a mile from *Computerworld's* headquarters!)

From this base of people, FIG is able to produce a technical journal, *Forth Dimensions*, which is improving all the time.

Since Forth is extendable -- that is any user can add new statements (either because the language is becoming more appreciated or else because the particular application or installation wants a different vocabulary) the journal's emphasis is on comparing different methods of implementing language elements. This focus allows the community to see how to keep the language efficient.

The journal also promotes the continued development of the Forth standard, annual conferences, and general communication among the many groups.

All this, however, is only as important as what is actually made with the FIG Forths. And that was why the 1980 conference was particularly important.

Outside Language

Forth, before now, had an outside language which, while somewhat Pascal-like, was distinctly forbidding and, because of the rareness of Forth programmers, something that users hated to use.

However, other more popular and conventional languages including Pascal, Lisp, Basic and (potentially) Cobol can be written in Forth, thus releasing the employment problem, while adding for their users the extending [illegible]

(Continued on Page 31)

FIGFORTH, TOO

If you have had a long wait for delivery of an order from the **FORTH Interest Group** (again in October 13's "Data Files"), it may be the post office's fault. I, too, ordered copies of the figFORTH manuals and source code for FORTH. It took 22 days for our super-efficient postal service to deliver my copies. Also received from the FORTH Interest Group were copies of *FORTH Dimensions*, its bimonthly publication. The September/October 1980 issue, larger than normal with over 90 pages, was professionally prepared and made good reading. The reason for this extra-size issue (regular issues seem to run about 35 pages) was publication of the source code for entries in a "CASE" statement contest. *FORTH Dimensions* is sent as part of membership in the FORTH Interest Group. The current membership cost is $12 per year in the U.S. and Canada, and $15 per year overseas.

Forth for Alpha Micro's AMOS

PALO ALTO, CA — Professional Management Services' (PMS) Version 3.2 of μA/Forth, a fig-Forth (Forth Interest Group) product, is aligned with the 1978 standard of the Forth International Standards Team and allows complete access to Alpha Microsystems' multitasking operating sytems, AMOS.

Forth was developed for control applications, data bases, and general business. μA/Forth implements full-length names up to 31 characters, extensively checks code at compile-time with error reporting, contains string-handling routines and a string-search editor, and permits sealed vocabularies to control user access. Included is a Forth assembler, permitting structured, interactive development of device handlers, speed-critical routines, and linkage to operating systems or to packages written in other languages.

As an extensible, threaded language Forth words (commands) may be created from previously defined words, and even the original words supplied with the system (about 100) can be redefined if desired, adapting the language for special circumstances.

The distribution disk is in single density, AMS format, and includes all source code. The diskette includes an editor, a Forth assembler, and string package in Forth source code. This complete system is available for $130.

For additional information, contact Professional Management Services, 724 Arastradero Rd., Suite 109, 94306, (408) 252-2218. Circle 202.

MEETINGS

How to form a FIG Chapter:

1. You decide on a time and place for the first meeting in your area. (Allow about 8 weeks for steps 2 and 3.)

2. Send to FIG in San Carlos, CA a meeting announcement on one side of 8-1/2 x 11 paper (one copy is enough). Also send list of ZIP numbers that you want mailed to (use first three digits if it works for you).

3. FIG will print, address and mail to members with the ZIP's you want from San Carlos, CA.

4. When you've had your first meeting with 5 or more attendees then FIG will provide you with names in your area. You have to tell us when you have 5 or more.

Northern California
4th Saturday — FIG Monthly Meeting, 1:00 p.m., at Southland Shopping Ctr., Hayward, CA. FORML Workshop at 10:00 a.m.

Southern California
4th Saturday — FIG Meeting, 11:00 a.m. Allstate Savings, 8800 So. Sepulveda, L.A. Call Phillip Wass, (213) 649-1428.

FIGGRAPH
2/14/81
3/14/81 — FORTH for computer graphics. 1:00 p.m. at Stanford Medical School, #M-112 at Palo Alto, CA. Need Info? L. Pearlmutter 415/856-1234

Massachusetts
3rd Wednesday — MMSFORTH Users Group, 7:00 p.m., Cochituate, MA. Call Dick Miller at (617) 653-6136 for site.

San Diego
Thursdays — FIG Meeting, 12:00 noon. Call Guy Kelly at (714) 268-3100 x 4784 for site.

Seattle
Various times — Contact Chuck Pliske or Dwight Vandenburg at (206) 542-8370.

Potomac
Various times — Contact Paul van der Eijk at (703) 354-7443 or Joel Shprentz at (703) 437-9218.

Texas
Various times — Contact Jeff Lewis at (713) 729-3320 or John Earls at (214) 661-2928 or Dwayne Gustaus at (817) 387-6976. John Hastings (512) 835-1918

Arizona
Various times — Contact Dick Wilson at (602) 277-6611 x 3257.

Oregon
Various times — Contact Ed Krammerer at (503) 644-2688.

New York
Various times — Contact Tom Jung at (212) 746-4062.

Detroit
Various times — Contact Dean Vieau at (313) 493-5105.

Japan
Various times — Contact Mr. Okada, President, ASR Corp. Int'l, 3-15-8, Nishi-Shimbashi Manato-ku, Tokyo, Japan.

Quebec, Canada
Various times — Contact Gilles Paillard (418) 871-1960.

Publishers Note:

Please send notes (and reports) about your meetings.