



INTERACTIVE COMPUTER SYSTEMS, INC.  
6403 DIMARCO ROAD • TAMPA, FLORIDA 33614  
(813) 884-5270

COURTESY COPY

fig-FORTH

INSTALLATION MANUAL

GLOSSARY

MODEL

RELEASE 1

WITH COMPILER SECURITY

AND

VARIABLE LENGTH NAMES

272

MAY 1979

Provided through the courtesy of the Forth Interest Group, P.O. Box 1105,  
San Carlos, CA 94070.

Further distribution of this public domain publication must include this notice.

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

**fig-FORTH**

**INSTALLATION MANUAL**

**GLOSSARY**

**MODEL**

**RELEASE 1**

**WITH COMPILER SECURITY**

**AND**

**VARIABLE LENGTH NAMES**

**272**

**MAY 1979**

**Provided through the courtesy of the Forth Interest Group, P.O. Box 1105,  
San Carlos, CA 94070.**

**Further distribution of this public domain publication must include this notice.**

## fig-FORTH INSTALLATION MANUAL

1.0	INTRODUCTION
2.0	DISTRIBUTION
3.0	MODEL ORGANIZATION
4.0	INSTALLATION
5.0	MEMORY MAP
6.0	DOCUMENTATION SUMMARY

### 1.0 INTRODUCTION

The fig-FORTH implementation project occurred because a key group of Forth fanciers wished to make this valuable tool available on a personal computing level. In June of 1978, we gathered a team of nine systems level programmers, each with a particular target computer. The charter of the group was to translate a common model of Forth into assembly language listings for each computer. It was agreed that the group's work would be distributed in the public domain by FIG. This publication series is the conclusion of the work.

### 2.0 DISTRIBUTION

All publications of the Forth Interest Group are public domain. They may be further reproduced and distributed by inclusion of this credit notice:

This publication has been made available by the Forth Interest Group,  
P. O. Box 1105, San Carlos, Ca 94070

We intend that our primary recipients of the Implementation Project be computer users groups, libraries, and commercial vendors. We expect that each will further customize for particular computers and redistribute. No restrictions are placed on cost, but we

expect faithfulness to the model. FIG does not intend to distribute machine readable versions, as that entails customization, revision, and customer support better reserved for commercial vendors.

Of course, another broad group of recipients of the work is the community of personal computer users. We hope that our publications will aid in the use of Forth and increase the user expectation of the performance of high level computer languages.

### 3.0 MODEL ORGINIZATION

The fig-FORTH model deviates a bit from the usual loading method of Forth. Existing systems load about 2k bytes in object form and then self-compile the resident system (6 to 8 k bytes). This technique allows customization within the high level portion, but is impractical for new implementors.

Our model has 4 to 5 k bytes written as assembler listings. The remainder may be compiled typing in the Forth high-level source, by more assembly source, or by disc compilation. This method enhances transportability, although the larger portion in assembly code entails more effort. About 8k bytes of memory is used plus 2 to 8k for workspace.

#### 3.1 MODEL OVER-VIEW

The model consists of 7 distinct areas. They occur sequentially from low memory to high.

Boot-up parameters  
Machine code definitions  
High level utility definitions  
Installation dependent code  
High level definitions  
System tools (optional)  
RAM memory workspace

### 3.2 MODEL DETAILS

#### Boot-up Parameters

This area consists of 34 bytes containing a jump to the cold start, jump to the warm re-start and initial values for user variables and registers. These values are altered as you make permanent extensions to your installation.

#### Machine Code Definitions

This area consists of about 600 to 800 bytes of machine executable code in the form of Forth word definitions. Its purpose is to convert your computer into a standard Forth stack computer. Above this code, the balance of Forth contains a pseudo-code compiled of "execution-addresses" which are sequences of the machine address of the "code-fields" of other Forth definitions. All execution ultimately refers to the machine code definitions.

#### High-level Utility Definitions

These are colon-definitions, user variables, constants, and variables that allow you to control the "Forth stack computer". They comprise the bulk of the system, enabling you to execute and compile from the terminal. If disc storage (or a RAM simulation of disc) is available, you may also execute and compile from this facility. Changes in the high-level area are infrequent. They may be made thru the assembler source listings.

#### Installation Dependent Code

This area is the only portion that need change between different installations of the same computer cpu. There are four code fragments:

(KEY) Push the next ascii value (7 bits) from the terminal keystroke to the computation stack and execute NEXT. High 9 bits are zero. Do not echo this character, especially a control character.

(EMIT) Pop the computation stack (16 bit value). Display the low 7 bits on the terminal device, then execute NEXT. Control characters have their natural functions.

(?TERMINAL) For terminals with a break key, wait till released and push to the computation stack 0001 if it was found depressed; otherwise 0000. Execute NEXT. If no break key is available, sense any key depression as a break (sense but don't wait for a key). If both the above are unavailable, simply push 0000 and execute NEXT.

(CR) Execute a terminal carriage return and line feed. Execute NEXT.

When each of these words is executed, the interpreter vectors from the definition header to these code sequences. On specific implementations it may be necessary to preseve certain registers and observe operating system protocols. Understand the implementors methods in the listing before proceeding!

R/W This colon-definition is the standard linkage to your disc. It requests the read or write of a disc sector. It usually requires supporting code definitions. It may consist of self-contained code or call ROM monitor code. When R/W is assembled, its code field address is inserted once in BLOCK and once in BUFFER.

An alternate version of R/W is included that simulates disc storage in RAM. If you have over 16 k bytes this is practical for startup and limited operation with cassette.

#### High-level Definitions

The next section contains about 30 definitions involving user interaction: compiling aids, finding, forgetting, listing, and number formating. These definitions are placed above the installation dependent code to facilitate modification. That is, once your full system is up, you may FORGET part of the high-level and re-compile altered definitions from disc.

#### Sytsem Tools

A text editor and machine code assembler are normally resident. We are including a sample editor, and hope to provide Forth assemblers. The editor is compiled from the terminal the first time, and then used to place the editor and assembler source code on disc.

It is essential that you regard the assembly listing as just a way to get Forth installed on your system. Additions and changes must be planned and tested at the usual Forth high level and then the assmbly routines updated. Forth work planned and executed only at an assembly level tends to be non-portable, and confusing.

#### RAM Workspace

For a single user system, at least 2k bytes must be available above the compiled system (the dictionary). A 16k byte total system is most typical.

The RAM workspace contains the computation and return stacks, user area, terminal input buffer, disc buffer and compilation space for the dictionary.

## 4.0 INSTALLATION

We see the following methods of getting a functioning fig-FORTH system:

1. Buy loadable object code from a vendor who has customized.
2. Obtain an assembly listing with the installation dependent code supplied by the vendor.  
Assemble and execute.
3. Edit the FIG assembly listing on your system, re-write the I-O routines, and assemble.
4. Load someone else's object code up to the installation dependent code. Hand assemble equivalents for your system and poke in with your monitor. Begin execution and type in (self-compile) the rest of the system. This takes

about two hours once you understand the structure of Forth (but that will take much more time!).

Let us examine Step 3, above, in fuller detail. If you wish to bring up Forth only from this model, here are the sequential steps:

4.1 Familiarize yourself with the model written in Forth, the glossary, and specific assembly listings.

4.2 Edit the assembly listings into your system. Set the boot-up parameters at origin offset 0A, 0B (bytes) to 0000 (warning=00).

4.3 Alter the terminal support code (KEY, EMIT, etc,) to match your system. Observe register protocol specific to your implementation!

4.4 Place a break to your monitor at the end of NEXT, just before indirectly jumping via register W to execution. W is the Forth name for the register holding a code field address, and may be differently referenced in your listings.

4.5 Enter the cold start at the origin. Upon the break, check that the interpretive pointer IP points within ABORT and W points to SP!. If COLD is a colon-definition, then the IP has been initialized on the way to NEXT and your testing will begin in COLD. The purpose of COLD is to initialize IP, SP, RP, UP, and some user variables from the start-up parameters at the origin.

4.6 Continue execution one word at a time. Clever individuals could write a simple trace routine to print IP, W, SP, RP and the top of the stacks. Run in this single step mode until the greeting message is printed. Note that the interpretation is several hundred cycles to this stage!

4.7 Execution errors may be localized by observing the above pointers when a crash occurs.

4.8 After the word QUIT is executed (incrementally), and you can input a "return" key and get OK printed, remove the break. You may have some remaining errors, but a reset and examination of the above registers will again localize problems.

4.9 When the system is interpreting from the keyboard, execute EMPTY-BUFFERS to clear the disc buffer area. You may test the disc access by typing: 0 BLOCK 64 TYPE This should bring sector zero from the disc to a buffer and type the first 64 characters. This sector usually contains ascii text of the disc directory. If BLOCK (and R/W) doesn't function--happy hunting!

5.0 If your disc driver differs from the assembly version, you must create your own R/W. This word does a range check (with error message), modulo math to derive sector, track, and drive and passes values to a sector-read and sector-write routine.

### RAM DISC SIMULATION

If disc is not available, a simulation of BLOCK and BUFFER may be made in RAM. The following definitions setup high memory as mass storage. Referenced 'screens' are then brought to the 'disc buffer' area. This is a good method to test the start-up program even if disc may be available.

#### HEX

```
4000 CONSTANT LO  ( START OF BUFFER AREA )
6800 CONSTANT HI  ( 10 SCREEN EQUIVALENT )
: R/W >R ( save boolean )
  B/BUF * LO + DUP
  HI > 6 ?ERROR ( range check )
  R> IF ( read ) SWAP ENDIF
  B/BUF CMOVE ;
```

Insert the code field address of R/W into BLOCK and BUFFER and proceed as if testing disc. R/W simulates screens 0 thru 9 when B/BUF is 128, in the memory area \$4000 thru \$6BFF.

#### fig-FORTH VARIABLE NAME FIELD

A major FIG innovation in this model, is the introduction of variable length definition names in compiled dictionary entries. Previous methods only saved three letters and the character count.

The user may select the letter count saved, up to the full natural length. See the glossary definition for WIDTH.

In this model, the following conventions have been established.

1. The first byte of the name field has the natural character count in the low 5 bits.
2. The sixth bit = 1 when smudged, and will prevent a match by (FIND).
3. The seventh bit = 1 for IMMEDIATE definitions; it is called the precedence bit.
4. The eighth or sign bit is always = 1.
5. The following bytes contain the names' letters, up to the value in WIDTH.
6. In the byte containing the last letter saved, the sign bit = 1.
7. In word addressing computer, a name may be padded with a blank to a word boundary.

The above methods are implemented in CREATE. Remember that -FIND uses BL WORD to bring the next text to HERE with the count preceding. All that is necessary, is to limit by WIDTH and toggle the proper delimiting bits.

#### 5.0 MEMORY MAP

The following memory map is broadly used. Specific installations may require alterations but you may forfeit functions in future FIG offerings.

The disc buffer area is at the upper bound of RAM memory. It is comprised of an integral number of buffers, each B/BUF+4 bytes. B/BUF is the number of bytes read from the disc, usually one sector. B/BUF must be a power of two (64, 128, 256, 512 or 1024). The constant FIRST has the value of the address of the start of the first buffer. LIMIT has the value of the first address beyond the top buffer. The distance between FIRST and LIMIT must be N\*(B/BUF+4) bytes. This N must be two or more.

Constant B/SCR has the value of the number of buffers per screen; i.e. 1024 / B/BUF.

The user area must be at least 34 bytes; 48 is more appropriate. In a multi-user system, each user has his own user area, for his copy of system variables. This method allows re-entrant use of the Forth vocabulary.

The terminal input buffer is decimal 80 bytes (the hex 50 in QUERY) plus 2 at the end. If a different value is desired, change the limit in QUERY. A parameter in the boot-up literals locates the address of this area for TIB. The backspace character is also in the boot-up origin parameters. It is universally expected that "rubout" is the backspace.

The return stack grows downward from the user area toward the terminal buffer. Forty-eight bytes are sufficient. The origin is in R0 (R-zero) and is loaded from a boot-up literal.

The computation stack grows downward from the terminal buffer toward the dictionary, which grows upward. The origin of the stack is in variable S0 (S-zero) and is loaded from a boot-up literal.

After a cold start, the user variables contain the addresses of the above memory assignments. An advanced user may relocate while the system is running. A newcomer should alter the startup literals and execute COLD. The word +ORIGIN is provided for this purpose. +ORIGIN gives the address byte or word relative to the origin depending on the computer addressing method. To change the backspace to control H type:

HEX 08 0E +ORIGIN ! (byte addresses)

#### 6.0 DOCUMENTATION SUMMARY

The following manuals are in print:

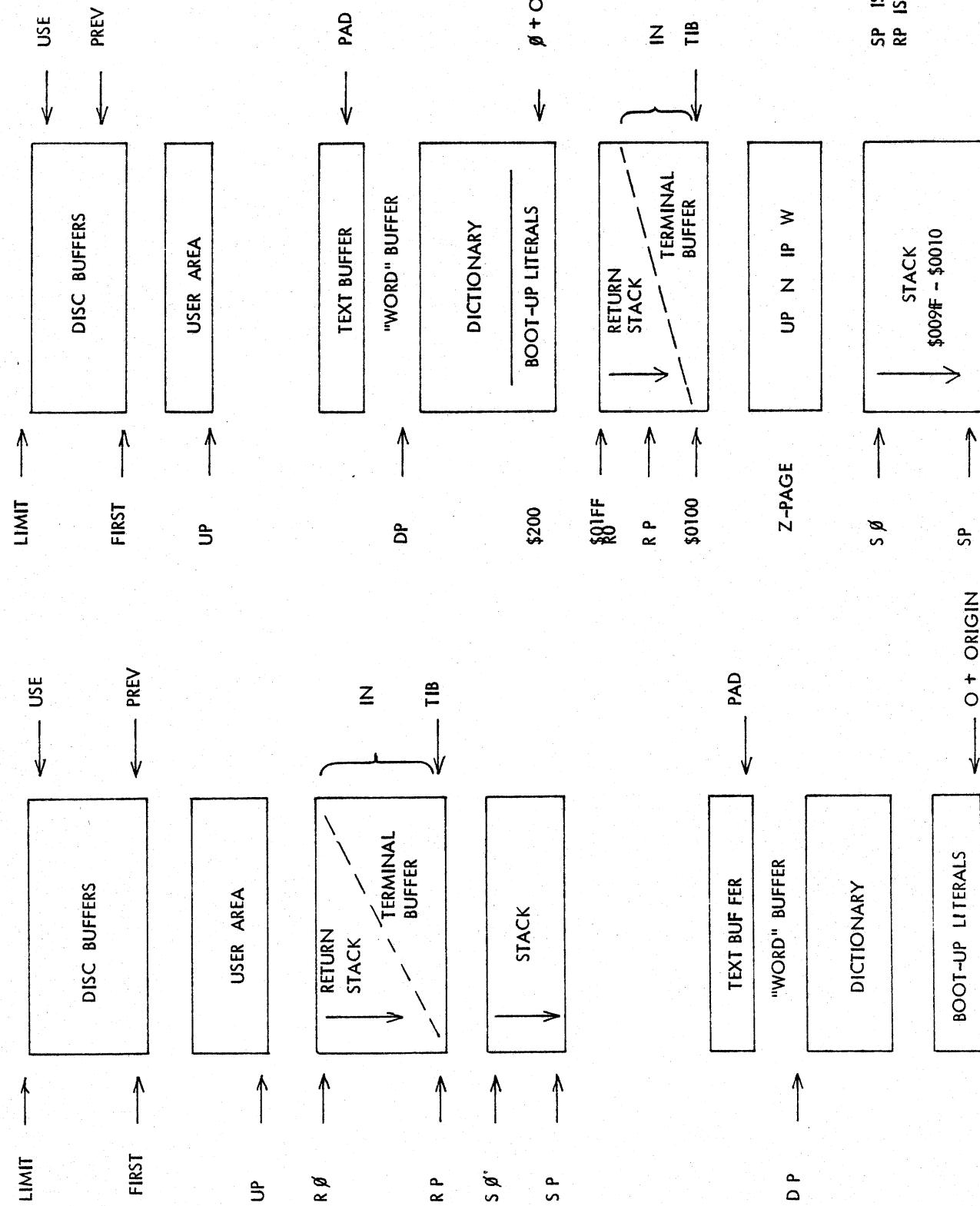
Caltech FORTH Manual, an advanced manual with internal details of Forth. Has some implementation peculiarities. Approx. \$6.50 from the Caltech Book Store, Pasadena, CA.

Kitt Peak Forth Primer, \$20.00 postpaid from the Forth Interest Group, P. O. Box 1105, San Carlos, CA 94070.

microFORTH Primer, \$15.00 Forth, Inc. 815 Manhattan Ave. Manhattan Beach, CA 90266

Forth Dimensions, newsletter of the Forth Interest Group, \$5.00 for 6 issues including membership. F.I.G. P.O. Box 1105, San Carlos, CA. 94070

**STANDARD**  
**fig-FORTH MEMORY MAP**



## fig-FORTH GLOSSARY

This glossary contains all of the word definitions in Release 1 of fig-FORTH. The definitions are presented in the order of their ascii sort.

The first line of each entry shows a symbolic description of the action of the procedure on the parameter stack. The symbols indicate the order in which input parameters have been placed on the stack. Three dashes "\_\_\_" indicate the execution point; any parameters left on the stack are listed. In this notation, the top of the stack is to the right.

The symbols include:

addr	memory address
b	8 bit byte (i.e. hi 8 bits zero)
c	7 bit ascii character (hi 9 bits zero)
d	32 bit signed double integer, most significant portion with sign on top of stack.
f	boolean flag. 0=false, non-zero=true
ff	boolean false flag=0
n	16 bit signed integer number
u	16 bit unsigned integer
tf	boolean true flag=non-zero

The capital letters on the right show definition characteristics:

C	May only be used within a colon definition. A digit indicates number of memory addresses used, if other than one.
E	Intended for execution only.
L0	Level Zero definition of FORTH-78
L1	Level One definition of FORTH-78
P	Has precedence bit set. Will execute even when compiling.
U	A user variable.

Unless otherwise noted, all references to numbers are for 16 bit signed integers. On 8 bit data bus computers, the high byte of a number is on top of the stack, with the sign in the leftmost bit. For 32 bit signed double numbers, the most significant part (with the sign) is on top.

All arithmetic is implicitly 16 bit signed integer math, with error and under-flow indication unspecified.

!	n addr ---	LO	(+LOOP)	n ---	C2
	Store 16 bits of n at address. Pronounced "store".			The run-time procedure compiled by +LOOP, which increments the loop index by n and tests for loop completion. See +LOOP.	
!CSP			(ABORT)		
	Save the stack position in CSP. Used as part of the compiler security.				
#	d1 --- d2	LO		Executes after an error when WARNING is -1. This word normally executes ABORT, but may be altered (with care) to a user's alternative procedure.	
	Generate from a double number d1, the next ascii character which is placed in an output string. Result d2 is the quotient after division by BASE, and is maintained for further processing. Used between <# and #>. See #S.		(DO)		
#>	d --- addr count	LO	(FIND)	addr1 addr2 --- pfa b tf (ok) addr1 addr2 --- ff (bad)	
	Terminates numeric output conversion by dropping d, leaving the text address and character count suitable for TYPE.			Searches the dictionary starting at the name field address addr2, matching to the text at addr1. Returns parameter field address, length byte of name field and boolean true for a good match. If no match is found, only a boolean false is left.	
#S	d1 --- d2	LO	(LINE)	n1 n2 --- addr count	
	Generates ascii text in the text output buffer, by the use of #, until a zero double number n2 results. Used between <# and #>.			Convert the line number n1 and the screen n2 to the disc buffer address containing the data. A count of 64 indicates the full line text length.	
	--- addr	P,LO	(LOOP)		C2
	Used in the form: ' nnnn			The run-time procedure compiled by LOOP which increments the loop index and tests for loop completion. See LOOP.	
	Leaves the parameter field address of dictionary word nnnn. As a compiler directive, executes in a colon-definition to compile the address as a literal. If the word is not found after a search of CONTEXT and CURRENT, an appropriate error message is given. Pronounced "tick".		(NUMBER)	dl addr1 --- d2 addr2	
(				Convert the ascii text beginning at addr1+1 with regard to BASE. The new value is accumulated into double number dl, being left as d2. Addr2 is the address of the first unconvertable digit. Used by NUMBER.	
	P,LO		*	n1 n2 --- prod	
	Used in the form: ( cccc )			Leave the signed product of two signed numbers.	
	Ignore a comment that will be delimited by a right parenthesis on the same line. May occur during execution or in a colon-definition. A blank after the leading parenthesis is required.			n1 n2 n3 --- n4	
(.)		C+		Leave the ratio n4 = n1*n2/n3 where all are signed numbers. Retention of an intermediate 31 bit product permits greater accuracy than would be available with the sequence:	
	The run-time procedure, compiled by ." which transmits the following in-line text to the selected output device. See ."		*/	n1 n2 * n3 /	
(;CODE)		C			
	The run-time procedure, compiled by ;CODE, that rewrites the code field of the most recently defined word to point to the following machine code sequence. See ;CODE.		*/MOD	n1 n2 n3 --- n4 n5	
				Leave the quotient n5 and remainder n4 of the operation n1*n2/n3. A 31 bit intermediate product is used as for */.	

+ nl n2 --- sum LO -DUP nl -- nl (if zero)  
 Leave the sum of nl+n2. nl -- nl nl (non-zero) LO  
 +! n addr --- LO Reproduce nl only if it is non-zero.  
 Add n to the value at the address. This is usually used to copy a value  
 Pronounced "plus-store". just before IF, to eliminate the need  
 for an ELSE part to drop it.

+- nl n2 --- n3 -FIND --- pfa b tf (found)  
 Apply the sign of n2 to nl, which --- ff (not found)  
 is left as n3. Accepts the next text word (delimited  
 by blanks) in the input stream to  
 HERE, and searches the CONTEXT and  
 then CURRENT vocabularies for a  
 matching entry. If found, the  
 dictionary entry's parameter field  
 address, its length byte, and a  
 boolean true is left. Otherwise,  
 only a boolean false is left.

+BUF add1 --- addr2 f -TRAILING addr nl --- addr n2  
 Advance the disc buffer address add1 Adjusts the character count nl of a  
 to the address of the next buffer text string beginning address to  
 addr2. Boolean f is false when addr2 suppress the output of trailing  
 is the buffer presently pointed to blanks. i.e. the characters at  
 by variable PREV. addr+nl to addr+n2 are blanks.

+LOOP nl --- (run) n --- LO  
 addr n2 --- (compile) P,C2,LO Print a number from a signed 16 bit  
 Used in a colon-definition in the two's complement value, converted  
 form: according to the numeric BASE.  
 DO ... nl +LOOP A trailing blanks follows.  
 At run-time, +LOOP selectively  
 controls branching back to the cor- .  
 responding DO based on nl, the loop  
 index and the loop limit. The signed  
 increment nl is added to the index  
 and the total compared to the limit.  
 The branch back to DO occurs until  
 the new index is equal to or greater  
 than the limit ( $nl > 0$ ), or until the  
 new index is equal to or less than  
 the limit ( $nl < 0$ ). Upon exiting the  
 loop, the parameters are discarded  
 and execution continues ahead.

At compile time, +LOOP compiles  
 the run-time word (+LOOP) and the  
 branch offset computed from HERE to  
 the address left on the stack by  
 DO. n2 is used for compile time  
 error checking.

+ORIGIN n --- addr .LINE line scr ---  
 Leave the memory address relative Print on the terminal device, a line  
 by n to the origin parameter area. of text from the disc by its line and  
 n is the minimum address unit, either screen number. Trailing blanks are  
 byte or word. This definition is used suppressed.

, n --- LO .R nl n2 ---  
 Store n into the next available dict- Print the number nl right aligned in  
 ionary memory cell, advancing the a field whose width is n2. No  
 dictionary pointer. (comma) following blank is printed.

- nl n2 --- diff LO /MOD nl n2 --- quot LO  
 Leave the difference of nl-n2. Leave the signed quotient of nl/n2.

--> P,LO nl n2 --- rem quot LO  
 Continue interpretation with the Leave the remainder and signed  
 next disc screen. (pronounced quotient of nl/n2. The remainder has  
 next-screen). the sign of the dividend.



?ERROR	f n ---	B/BUF	--- n This constant leaves the number of bytes per disc buffer, the byte count read from disc by BLOCK.
?EXEC	Issue an error message if not executing.	B/SCR	--- n This constant leaves the number of blocks per editing screen. By convention, an editing screen is 1024 bytes organized as 16 lines of 64 characters each.
?LOADING	Issue an error message if not loading		
?PAIRS	n1 n2 --- Issue an error message if n1 does not equal n2. The message indicates that compiled conditionals do not match.	BACK	addr --- Calculate the backward branch offset from HERE to addr and compile into the next available dictionary memory address.
?STACK	Issue an error message if the stack is out of bounds. This definition may be installation dependent.	BASE	--- addr U,LO A user variable containing the current number base used for input and output conversion.
?TERMINAL --- f	Perform a test of the terminal keyboard for actuation of the break key. A true flag indicates actuation. This definition is installation dependent.	BEGIN	--- addr n (compiling) P,LO Occurs in a colon-definition in form: BEGIN ... UNTIL BEGIN ... AGAIN BEGIN ... WHILE ... REPEAT At run-time, BEGIN marks the start of a sequence that may be repetitively executed. It serves as a return point from the corresponding UNTIL, AGAIN or REPEAT. When executing UNTIL, a return to BEGIN will occur if the top of the stack is false; for AGAIN and REPEAT a return to BEGIN always occurs.
@ addr --- n LO	Leave the 16 bit contents of address.		At compile time BEGIN leaves its return address and n for compiler error checking.
ABORT	LO Clear the stacks and enter the execution state. Return control to the operators terminal, printing a message appropriate to the installation.	BL	---
ABS	n --- u LO Leave the absolute value of n as u.	BLANKS	A constant that leaves the ascii value for "blank".
AGAIN	addr n --- (compiling) P,C2,LO Used in a colon-definition in the form: BEGIN ... AGAIN At run-time, AGAIN forces execution to return to corresponding BEGIN. There is no effect on the stack. Execution cannot leave this loop (unless R> DROP is executed one level below).  At compile time, AGAIN compiles BRANCH with an offset from HERE to addr. n is used for compile-time error checking.	BLK	addr count --- Fill an area of memory beginning at addr with blanks.
ALLOT	n --- LO Add the signed number to the dictionary pointer DP. May be used to reserve dictionary space or re-origin memory. n is with regard to computer address type (byte or word).	BLOCK	--- addr U,LO A user variable containing the block number being interpreted. If zero, input is being taken from the terminal input buffer.
AND	n1 n2 --- n2 LO Leave the bitwise logical and of n1 and n2 as n3.		n --- addr LO Leave the memory address of the block buffer containing block n. If the block is not already in memory, it is transferred from disc to which ever buffer was least recently written. If the block occupying that buffer has been marked as updated, it is re-written to disc before block n is read into the buffer. See also BUFFER, R/W UPDATE FLUSH

## BLOCK-READ

BLOCK-WRITE These are the preferred names for the installation dependent code to read and write one block to the disc.

## BRANCH

C2,L0  
The run-time procedure to unconditionally branch. An in-line offset is added to the interpretive pointer IP to branch ahead or back. BRANCH is compiled by ELSE, AGAIN, REPEAT.

## BUFFER

n --- addr  
Obtain the next memory buffer, assigning it to block n. If the contents of the buffer is marked as updated, it is written to the disc. The block is not read from the disc. The address left is the first cell within the buffer for data storage.

## C!

b addr ---  
Store 8 bits at address. On word addressing computers, further specification is necessary regarding byte addressing.

## C,

b ---  
Store 8 bits of b into the next available dictionary byte, advancing the dictionary pointer. This is only available on byte addressing computers, and should be used with caution on byte addressing mini-computers.

## C@

addr --- b  
Leave the 8 bit contents of memory address. On word addressing computers, further specification is needed regarding byte addressing.

## CFA

pfa --- cfa  
Convert the parameter field address of a definition to its code field address.

## CMOVE

from to count ---  
Move the specified quantity of bytes beginning at address from to address to. The contents of address from is moved first proceeding toward high memory. Further specification is necessary on word addressing computers.

## COLD

The cold start procedure to adjust the dictionary pointer to the minimum standard and restart via ABORT. May be called from the terminal to remove application programs and restart.

## COMPILE

When the word containing COMPILE executes, the execution address of the word following COMPILE is copied (compiled) into the dictionary. This allows specific compilation situations to be handled in addition to simply compiling an execution address (which the interpreter already does).

## CONSTANT

n --- L0  
A defining word used in the form:  
n CONSTANT cccc  
to create word cccc, with its parameter field containing n. When cccc is later executed, it will push the value of n to the stack.

## CONTEXT

--- addr U,L0  
A user variable containing a pointer to the vocabulary within which dictionary searches will first begin.

## COUNT

addr1 --- addr2 n L0  
Leave the byte address addr2 and byte count n of a message text beginning at address addr1. It is presumed that the first byte at addr1 contains the text byte count and the actual text starts with the second byte. Typically COUNT is followed by TYPE.

## CREATE

A defining word used in the form:  
CREATE cccc  
by such words as CODE and CONSTANT to create a dictionary header for a Forth definition. The code field contains the address of the words parameter field. The new word is created in the CURRENT vocabulary.

## CSP

---- addr U  
A user variable temporarily storing the stack pointer position, for compilation error checking.

## D+

d1 d2 --- dsun  
Leave the double number sum of two double numbers.

## D+-

d1 n --- d2  
Apply the sign of n to the double number d1, leaving it as d2.

## D.

d --- L1  
Print a signed double number from a 32 bit two's complement value. The high-order 16 bits are most accessible on the stack. Conversion is performed according to the current BASE. A blank follows. Pronounced D-dot.

D.R	d n ---	DO	n1 n2 --- (execute) addr n --- (compile) P,C2,LO Occurs in a colon-definition in form: DO ... LOOP DO ... +LOOP
DABS	d --- ud		At run time, DO begins a sequence with repetitive execution controlled by a loop limit n1 and an index with initial value n2. DO removes these from the stack. Upon reaching LOOP the index is incremented by one. Until the new index equals or exceeds the limit, execution loops back to just after DO; otherwise the loop parameters are discarded and execution continues ahead. Both n1 and n2 are determined at run-time and may be the result of other operations. Within a loop 'I' will copy the
DECIMAL		LO	urrent value of the index to the stack. See I, LOOP, +LOOP, LEAVE.
DEFINITIONS		L1	When compiling within the colon-definition, DO compiles (DO), leaves the following address addr and n for later error checking.
DIGIT	c nl --- n2 tf (ok) c nl --- ff (bad)	DOES>	A word which defines the run-time action within a high-level defining word. DOES> alters the code field and first parameter of the new word to execute the sequence of compiled word addresses following DOES>. Used in combination with <BUILDS. When the DOES> part executes it begins with the address of the first parameter of the new word on the stack. This allows interpretation using this area or its contents. Typical uses include the Forth assembler, multi-dimensional arrays, and compiler generation.
DLIST	List the names of the dictionary entries in the CONTEXT vocabulary.		----
DLITERAL	d --- d (executing) d --- (compiling) P	DP	addr U,L A user variable, the dictionary pointer, which contains the address of the next free memory above the dictionary. The value may be read by HERE and altered by ALLOT.
DMINUS	d1 --- d2	DPL	----
	Convert d1 to its double number two's complement.		addr U,LO A user variable containing the number of digits to the right of the decimal on double integer input. It may also be used hold output column location of a decimal point, in user generated formatting. The default value on single number input is -1.
		DRO	Installation dependent commands to select disc drives, by presetting OFFSET. The contents of OFFSET is added to the block number in BLOCK to allow for this selection. Offset is suppressed for error text so that it may always originate from drive 0.
		DRI	

DROP	n ---	LO	ENDIF	addr n --- (compile) P,C0,LO Occurs in a colon-definition in form: IF ... ENDIF IF ... ELSE ... ENDIF At run-time, ENDIF serves only as the destination of a forward branch from IF or ELSE. It marks the conclusion of the conditional structure. THEN is another name for ENDIF. Both names are supported in fig-FORTH. See also IF and ELSE.
DUMP	addr n ---	LO		Print the contents of n memory locations beginning at addr. Both addresses and contents are shown in the current numeric base.
DUP	n --- n n	LO		Duplicate the value on the stack.
ELSE	addr1 n1 --- addr2 n2 (compiling)	P,C2,LO		Occurs within a colon-definition in the form: IF ... ELSE ... ENDIF At run-time, ELSE executes after the true part following IF. ELSE forces execution to skip over the following false part and resumes execution after the ENDIF. It has no stack effect.  At compile-time ELSE emplaces BRANCH reserving a branch offset, leaves the address addr2 and n2 for error testing. ELSE also resolves the pending forward branch from IF by calculating the offset from addr1 to HERE and storing at addr1.
EMIT	c ---	LO		Transmit ascii character c to the selected output device. OUT is incremented for each character output.
EMPTY-BUFFERS		LO	EXECUTE	addr -- Execute the definition whose code field address is on the stack. The code field address is also called the compilation address.
ENCLOSE	addr1 c --- ddrl nl n2 n3		EXPECT	addr count --- LO Transfer characters from the terminal to address, until a "return" or the count of characters have been received. One or more nulls are added at the end of the text.
	The text scanning primitive used by WORD. From the text address ddrl and an ascii delimiting character c, is determined the byte offset to the first non-delimiter character nl, the offset to the first delimiter after the text n2, and the offset to the first character not included. This procedure will not process past an ascii 'null', treating it as an unconditional delimiter.		FENCE	--- addr U A user variable containing an address below which FORGETting is trapped. To forget below this point the user must alter the contents of FENCE.
END		P,C2,LO	FILL	addr quan b --- Fill memory at the address with the specified quantity of bytes b.
	This is an 'alias' or duplicate definition for UNTIL.		FIRST	--- n A constant that leaves the address of the first (lowest) block buffer.

FLD	--- addr	U IF	f --- (run-time) --- addr n (compile) P,C2,L0 Occurs is a colon-definition in form: IF (tp) ... ENDIF IF (tp) ... ELSE (fp) ... ENDIF
FORGET		E,LO	At run-time, IF selects execution based on a boolean flag. If f is true (non-zero), execution continues ahead thru the true part. If f is false (zero), execution skips till just after ELSE to execute the false part. After either part, execution resumes after ENDIF. ELSE and its false part are optional.; if missing, false execution skips to just after ENDIF.
FORTH		P,L1	At compile-time IF compiles OBRANCH and reserves space for an offset at addr. addr and n are used later for resolution of the offset and error testing.
HERE	--- addr	L0	IMMEDIATE
	Leave the address of the next available dictionary location.		Mark the most recently made definition so that when encountered at compile time, it will be executed rather than being compiled. i.e. the precedence bit in its header is set. This method allows definitions to handle unusual compiling situations, rather than build them into the fundamental compiler. The user may force compilation of an immediate definition by preceding it with [COMPILE].
HEX		L0	
	Set the numeric conversion base to sixteen (hexadecimal).		
HLD	--- addr	L0	---
	A user variable that holds the address of the latest character of text during numeric output conversion.	IN	A user variable containing the byte offset within the current input text buffer (terminal or disc) from which the next text will be accepted. WORD uses and moves the value of IN.
HOLD	c ---	L0	
	Used between <# and #> to insert an ascii character into a pictured numeric output string. e.g. 2E HOLD will place a decimal point.	INDEX	from to --- Print the first line of each screen over the range from, to. This is used to view the comment lines of an area of text on disc screens.
I	--- n	C,LO	INTERPRET
	Used within a DO-LOOP to copy the loop index to the stack. Other use is implementation dependent. See R.		The outer text interpreter which sequentially executes or compiles text from the input stream (terminal or disc) depending on STATE. If the word name cannot be found after a search of CONTEXT and then CURRENT it is converted to a number according to the current base. That also failing, an error message echoing the name with a "?" will be given. Text input will be taken according to the convention for WORD. If a decimal point is found as part of a number, a double number value will be left. The decimal point has no other purpose than to force this action. See NUMBER.
ID.	addr ---		
	Print a definition's name from its name field address.		

KEY	--- c	LO LOOP	addr n --- (compiling) P,C2,LO Occurs in a colon-definition in form: DO ... LOOP	
LATEST	--- addr		At run-time, LOOP selectively controls branching back to the corresponding DO based on the loop index and limit. The loop index is incremented by one and compared to the limit. The branch back to DO occurs until the index equals or exceeds the limit; at that time, the parameters are discarded and execution continues ahead.	
LEAVE		C,LO	At compile-time, LOOP compiles (LOOP) and uses addr to calculate an offset to DO. n is used for error testing.	
LFA	pfa --- lfa	M*	n1 n2 --- d A mixed magnitude math operation which leaves the double number signed product of two signed number.	
LIMIT	---- n	M/	d n1 --- n2 n3 A mixed magnitude math operator which leaves the signed remainder n2 and signed quotient n3, from a double number dividend and divisor n1. The remainder takes its sign from the dividend.	
LIST	n ---	LO M/MOD	ud1 u2 --- u3 ud4 An unsigned mixed magnitude math operation which leaves a double quotient ud4 and remainder u3, from a double dividend ud1 and single divisor u2.	
LIT	--- n	C2,LO	n1 n2 --- max LO Leave the greater of two numbers.	
LITERAL	n --- (compiling)	P,C2,LO	MESSAGE n --- Print on the selected output device the text of line n relative to screen 4 of drive 0. n may be positive or negative. MESSAGE may be used to print incidental text such as report headers. If WARNING is zero, the message will simply be printed as a number (disc un-available).	
LOAD	n ---	LO	MIN	nl n2 --- min LO Leave the smaller of two numbers.
			MINUS	nl --- n2 LO Leave the two's complement of a number.
			MON	nl n2 --- mod LO Leave the remainder of nl/n2, with the same sign as nl.

MOVE	addr1 addr2 n --- Move the contents of n memory cells (16 bit contents) beginning at addr1 into n cells beginning at addr2. The contents of addr1 is moved first. This definition is appropriate on on word addressing computers.	PAD	--- addr Leave the address of the text output buffer, which is a fixed offset above HERE.
NEXT	This is the inner interpreter that uses the interpretive pointer IP to execute compiled Forth definitions. It is not directly executable but is the return point for all code pro- cedures. It acts by fetching the address pointed by IP, storing this value in register W. It then jumps to the address pointed to by W. W points to the code field of a definition which contains the address of the code which executes for that definition. This usage of indirect threaded code is a major contributor to the power, portability, and extensibility of Forth. Locations of IP and W are computer specific.	POP	The code sequence to remove a stack value and return to NEXT. POP is not directly executable, but is a Forth re-entry point after machine code.
NFA	pfa --- nfa Convert the parameter field address of a definition to its name field.	PREV	---- addr A variable containing the address of the disc buffer most recently ref- erenced. The UPDATE command marks this buffer to be later written to disc.
NUMBER	addr --- d Convert a character string left at addr with a preceding count, to a signed double number, using the current numeric base. If a decimal point is encountered in the text, its position will be given in DPL, but no other effect occurs. If numeric conversion is not possible, an error message will be given.	PUSH	This code sequence pushes machine registers to the computation stack and returns to NEXT. It is not directly executable, but is a Forth re-entry point after machine code.
OFFSET	--- addr A user variable which may contain a block offset to disc drives. The contents of OFFSET is added to the stack number by BLOCK. Messages by MESSAGE are independent of OFFSET. See BLOCK, DRO, DR1, MESSAGE.	PUT	This code sequence stores machine register contents over the topmost computation stack value and returns to NEXT. It is not directly exec- utable, but is a Forth re-entry point after machine code.
OR	n1 n2 -- or Leave the bit-wise logical or of two 16 bit values.	QUERY	Input 80 characters of text (or until a "return") from the operators terminal. Text is positioned at the address contained in TIB with IN set to zero.
OUT	--- addr A user variable that contains a value incremented by EMIT. The user may alter and examine OUT to control display formatting.	R	--- n Copy the top of the return stack to the computation stack.
OVER	n1 n2 --- n1 n2 n1 Copy the second stack value, placing it as the new top.	R#	--- addr A user variable which may contain the location of an editing cursor, or other file related function.

		SMUDGE	Used during word definition to toggle the "smudge bit" in a definitions' name field. This prevents an uncompleted definition from being found during dictionary searches, until compiling is completed without error.
R/W	addr blk f --- The fig-FORTH standard disc read-write linkage. addr specifies the source or destination block buffer, blk is the sequential number of the referenced block; and f is a flag for f=0 write and f=1 read. R/W determines the location on mass storage, performs the read-write and performs any error checking.		
R>	--- n L0 Remove the top value from the return stack and leave it on the computation stack. See >R and R.	SP!	
R0	--- addr U A user variable containing the initial location of the return stack. Pronounced R-zero. See RP!	SP@	--- addr A computer dependent procedure to return the address of the stack position to the top of the stack, as it was before SP@ was executed. (e.g. 1 2 SP@ @ . . . would type 2 2 1)
REPEAT	addr n --- (compiling) P,C2 Used within a colon-definition in the form: BEGIN ... WHILE ... REPEAT At run-time, REPEAT forces an unconditional branch back to just after the corresponding BEGIN.  At compile-time, REPEAT compiles BRANCH and the offset from HERE to addr. n is used for error testing.	SPACE	L0 Transmit an ascii blank to the output device.
ROT	n1 n2 n3 --- n2 n3 n1 L0 Rotate the top three values on the stack, bringing the third to the top.	SPACES	n --- LO Transmit n ascii blanks to the output device.
RP!	A computer dependent procedure to initialize the return stack pointer from user variable R0.	STATE	--- addr LO,U A user variable containing the compilation state. A non-zero value indicates compilation. The value itself may be implementation dependent.
S->D	n --- d Sign extend a single number to form a double number.	SWAP	n1 n2 --- n2 n1 LO Exchange the top two values on the stack.
SO	--- addr U A user variable that contains the initial value for the stack pointer. Pronounced S-zero. See SP!	THEN	P,C0,L0 An alias for ENDIF.
SCR	--- addr U A user variable containing the screen number most recently reference by LIST.	TIB	--- addr U A user variable containing the address of the terminal input buffer.
SIGN	n d --- d L0 Stores an ascii "-" sign just before a converted numeric output string in the text output buffer when n is negative. n is discarded, but double number d is maintained. Must be used between <# and #>.	TOGGLE	addr b --- Complement the contents of addr by the bit pattern b.
		TRAVERSE	addr1 n --- addr2 Move across the name field of a fig-FORTH variable length name field. addr1 is the address of either the length byte or the last letter. If n=1, the motion is toward hi memory; if n=-1, the motion is toward low memory. The addr2 resulting is address of the other end of the name.

	VARIABLE	E,LU
TRIAD	scr --- Display on the selected output device the three screens which include that numbered scr, beginning with a screen evenly divisible by three. Output is suitable for source text records, and includes a reference line at the bottom taken from line 15 of screen4.	
TYPE	addr count --- LO Transmit count characters from addr to the selected output device.	VOC-LINK --- addr U
U*	ul u2 --- ud Leave the unsigned double number product of two unsigned numbers.	A user variable containing the address of a field in the definition of the most recently created vocabulary. All vocabulary names are linked by these fields to allow control for FORGETTING thru multiple vocabularys.
U/	ud ul --- u2 u3 Leave the unsigned remainder u2 and unsigned quotient u3 from the unsigned double dividend ud and unsigned divisor ul.	VOCABULARY --- (run-time) E,L A defining word used in the form: VOCABULARY cccc
UNTIL	f --- (run-time) addr n --- (compile) P,C2,LO Occurs within a colon-definition in the form: BEGIN ... UNTIL At run-time, UNTIL controls the conditional branch back to the corresponding BEGIN. If f is false, execution returns to just after BEGIN; if true, execution continues ahead.  At compile-time, UNTIL compiles (OBRANCH) and an offset from HERE to addr. n is used for error tests.	to create a vocabulary definition cccc. Subsequent use of cccc will make it the CONTEXT vocabulary which is searched first by INTERPRET. The sequence "cccc DEFINITIONS" will also make cccc the CURRENT vocabulary into which new definitions are placed.
UPDATE	LO Marks the most recently referenced block (pointed to by PREV) as altered. The block will subsequently be transferred automatically to disc should its buffer be required for storage of a different block.	VLIST List the names of the definitions in the context vocabulary. "Break" will terminate the listing.
USE	--- addr A variable containing the address of the block buffer to use next, as the least recently written.	WARNING --- addr U A user variable containing a value controlling messages. If = 1 disc is present, and screen 4 of drive 0 is the base location for messages. If = 0, no disc is present and messages will be presented by number. If = -1, execute (ABORT) for a user specified procedure. See MESSAGE, ERROR.
USER	n --- LO A defining word used in the form: n USER cccc which creates a user variable cccc. The parameter field of cccc contains n as a fixed offset relative to the user pointer register UP for this user variable. When cccc is later executed, it places the sum of its offset and the user area base address on the stack as the storage address of that particular variable.	WHILE f --- (run-time) ad1 nl --- ad1 nl ad2 n2 P,C2 Occurs in a colon-definition in the form: BEGIN ... WHILE (tp) ... REPEAT At run-time, WHILE selects conditional execution based on boolean flag f. If f is true (non-zero), WHILE continues execution of the true part thru to REPEAT, which then branches back to BEGIN. If f is false (zero), execution skips to just after REPEAT, exiting the structure.
		At compile time, WHILE emplaces (OBRANCH) and leaves ad2 of the reserved offset. The stack values will be resolved by REPEAT.

WIDTH --- addr U  
In fig-FORTH, a user variable containing the maximum number of letters saved in the compilation of a definitions' name. It must be 1 thru 31, with a default value of 31. The name character count and its natural characters are saved, up to the value in WIDTH. The value may be changed at any time within the above limits.

WORD c --- LO  
Read the next text characters from the input stream being interpreted, until a delimiter c is found, storing the packed character string beginning at the dictionary buffer HERE. WORD leaves the character count in the first byte, the characters, and ends with two or more blanks. Leading occurrences of c are ignored. If BLK is zero, text is taken from the terminal input buffer, otherwise from the disc block stored in BLK.  
See BLK, IN.

X This is pseudonym for the "null" or dictionary entry for a name of one character of ascii null. It is the execution procedure to terminate interpretation of a line of text from the terminal or within a disc buffer, as both buffers always have a null at the end.

XOR n1 n2 --- xor L1  
Leave the bitwise logical exclusive-or of two values.

[ P,L1  
Used in a colon-definition in form:  
: xxx [ words ] more ;  
Suspend compilation. The words after [ are executed, not compiled. This allows calculation or compilation exceptions before resuming compilation with ]. See LITERAL, ].

{COMPILE} P,C  
Used in a colon-definition in form:  
: xxx {COMPILE} FORTH ;  
{COMPILE} will force the compilation of an immediate definition, that would otherwise execute during compilation. The above example will select the FORTH vocabulary when xxx executes, rather than at compile time.

Resume compilation, to the completion of a colon-definition. See [.

SCR # 3

0 \*\*\*\*\* fig-FORTH MODEL \*\*\*\*\*

1

2 Through the courtesy of

3

4 FORTH INTEREST GROUP  
5 P. O. BOX 1105  
6 SAN CARLOS, CA. 94070

7

8

9 RELEASE 1  
10 WITH COMPILER SECURITY  
11 AND  
12 VARIABLE LENGTH NAMES

13

14

15 Further distribution must include the above notice.

SCR # 4

0 ( ERROR MESSAGES )  
1 EMPTY STACK  
2 DICTIONARY FULL  
3 HAS INCORRECT ADDRESS MODE  
4 ISN'T UNIQUE  
5  
6 DISC RANGE ?  
7 FULL STACK  
8 DISC ERROR !

9

10

11

12

13

14

15 FORTH INTEREST GROUP

MAY 1, 1979

SCR # 5

0 ( ERROR MESSAGES )  
1 COMPILE ONLY, USE IN DEFINITION  
2 EXECUTION ONLY  
3 CONDITIONALS NOT PAIRED  
4 DEFINITION NOT FINISHED  
5 IN PROTECTED DICTIONARY  
6 USE ONLY WHEN LOADING  
7 OFF CURRENT EDITING SCREEN  
8 DECLARE VOCABULARY

9

10

11

12

13

14

15

FORTH INTEREST GROUP

MAY 1, 1979

CODE LIT	( PUSH FOLLOWING LITERAL TO STACK *)	1	13
LABEL PUSH	( PUSH ACCUM AS HI-BYTE, ML STACK AS LO-BYTE *)	4	13
LABEL PUT	( REPLACE BOTTOM WITH ACCUM. AND ML STACK *)	6	13
LABEL NEXT	( EXECUTE NEXT FORTH ADDRESS, MOVING IP *)	8	13
HERE ' <CLIT> ! HERE 2+ ,	( MAKE SILENT WORD *)	1	14
LABEL SETUP	( MOVE # ITEMS FROM STACK TO 'N' AREA OF Z-PAGE *)	4	14
CODE EXECUTE	( EXECUTE A WORD BY ITS CODE FIELD *)	9	14
	( ADDRESS ON THE STACK *)	10	14
CODE BRANCH	( ADJUST IP BY IN-LINE 16 BIT LITERAL *)	1	15
CODE OBRANCH	( IF BOT IS ZERO, BRANCH FROM LITERAL *)	6	15
CODE (LOOP)	( INCREMENT LOOP INDEX, LOOP UNTIL => LIMIT *)	1	16
CODE (+LOOP)	( INCREMENT INDEX BY STACK VALUE +/- *)	8	16
CODE (DO)	( MOVE TWO STACK ITEMS TO RETURN STACK *)	2	17
CODE I	( COPY CURRENT LOOP INDEX TO STACK *)	9	17
CODE DIGIT	( CONVERT ASCII CHAR-SECOND, WITH BASE-BOTTOM *)	1	18
	( IF OK RETURN DIGIT-SECOND, TRUE-BOTTOM; *)	2	18
	( OTHERWISE FALSE-BOTTOM. *)	3	18
CODE (FIND)	( HERE, NFA ... PFA, LEN BYTE, TRUE; ELSE FALSE *)	1	19
CODE ENCLOSE	( ENTER WITH ADDRESS-2, DELIM-1. RETURN WITH *)	1	20
	( ADDR-4, AND OFFSET TO FIRST CH-3, END WORD-2, NEXT CH-1 *)	2	20
CODE EMIT	( PRINT ASCII VALUE ON BOTTOM OF STACK *)	5	21
CODE KEY	( ACCEPT ONE TERMINAL CHARACTER TO THE STACK *)	7	21
CODE ?TERMINAL	( 'BREAK' LEAVES 1 ON STACK; OTHERWISE 0 *)	9	21
CODE CR	( EXECUTE CAR. RETURN, LINE FEED ON TERMINAL *)	11	21
CODE CMOVE	( WITHIN MEMORY; ENTER W/ FROM-3, TO-2, QUAN-1 *)	1	22
CODE U*	( 16 BIT MULTIPLICAND-2, 16 BIT MULTIPLIER-1 *)	1	23
	( 32 BIT UNSIGNED PRODUCT: LO WORD-2, HI WORD-1 *)	2	23
CODE U/	( 31 BIT DIVIDEND-2, -3, 16 BIT DIVISOR-1 *)	1	24
	( 16 BIT REMAINDER-2, 16 BIT QUOTIENT-1 *)	2	24
CODE AND	( LOGICAL BITWISE AND OF BOTTOM TWO ITEMS *)	2	25
CODE OR	( LOGICAL BITWISE 'OR' OF BOTTOM TWO ITEMS *)	6	25
CODE XOR	( LOGICAL 'EXCLUSIVE-OR' OF BOTTOM TWO ITEMS *)	10	25
CODE SP@	( FETCH STACK POINTER TO STACK *)	1	26
CODE SP!	( LOAD SP FROM 'SO' *)	5	26
CODE RP!	( LOAD RP FROM RO *)	8	26
CODE ;S	( RESTORE IP REGISTER FROM RETURN STACK *)	12	26
CODE LEAVE	( FORCE EXIT OF DO-LOOP BY SETTING LIMIT *)	1	27
XSAVE STX, TSX, R LDA, R 2+ STA,	( TO INDEX *)	2	27
CODE >R	( MOVE FROM COMP. STACK TO RETURN STACK *)	5	27
CODE R>	( MOVE FROM RETURN STACK TO COMP. STACK *)	8	27
CODE R	( COPY THE BOTTOM OF RETURN STACK TO COMP. STACK *)	11	27
CODE 0=	( REVERSE LOGICAL STATE OF BOTTOM OF STACK *)	2	28
CODE 0<	( LEAVE TRUE IF NEGATIVE; OTHERWISE FALSE *)	6	28
CODE +	( LEAVE THE SUM OF THE BOTTOM TWO STACK ITEMS *)	1	29
CODE D+	( ADD TWO DOUBLE INTEGERS, LEAVING DOUBLE *)	4	29
CODE MINUS	( TWOS COMPLEMENT OF BOTTOM SINGLE NUMBER *)	9	29
CODE DMINUS	( TWOS COMPLEMENT OF BOTTOM DOUBLE NUMBER *)	12	29
CODE OVER	( DUPLICATE SECOND ITEM AS NEW BOTTOM *)	1	30
CODE DROP	( DROP BOTTOM STACK ITEM *)	4	30
CODE SWAP	( EXCHANGE BOTTOM AND SECOND ITEMS ON STACK *)	7	30
CODE DUP	( DUPLICATE BOTTOM ITEM ON STACK *)	11	30
CODE +!	( ADD SECOND TO MEMORY 16 BITS ADDRESSED BY BOTTOM *)	2	31
CODE TOGGLE	( BYTE AT ADDRESS-2, BIT PATTERN-1 ... *)	7	31
CODE @	( REPLACE STACK ADDRESS WITH 16 BIT *)	1	32
BOT X) LDA, PHA,	( CONTENTS OF THAT ADDRESS *)	2	32
CODE C@	( REPLACE STACK ADDRESS WITH POINTED 8 BIT BYTE *)	5	32
CODE !	( STORE SECOND AT 16 BITS ADDRESSED BY BOTTOM *)	8	32

CODE C!	( STORE SECOND AT BYTE ADDRESSED BY BOTTOM	* )	12	32
:	( CREATE NEW COLON-DEFINITION UNTIL ' ; '	* )	2	33
:	( TERMINATE COLON-DEFINITION	* )	9	33
: CONSTANT	( WORD WHICH LATER CREATES CONSTANTS	* )	1	34
: VARIABLE	( WORD WHICH LATER CREATES VARIABLES	* )	5	34
: USER	( CREATE USER VARIABLE	* )	10	34
20 CONSTANT BL	CR ( ASCII BLANK	* )	4	35
40 CONSTANT C/L	( TEXT CHARACTERS PER LINE	* )	5	35
3BEO CONSTANT FIRST	( FIRST BYTE RESERVED FOR BUFFERS	* )	7	35
4000 CONSTANT LIMIT	( JUST BEYOND TOP OF RAM	* )	8	35
80 CONSTANT B/BUF	( BYTES PER DISC BUFFER	* )	9	35
8 CONSTANT B/SCR	( BLOCKS PER SCREEN = 1024 B/BUF /	* )	10	35
: +ORIGIN LITERAL + ;	( LEAVES ADDRESS RELATIVE TO ORIGIN	* )	13	35
HEX	( 0 THRU 5 RESERVED, REFERENCED TO \$00AO	* )	1	36
( 06 USER SO )	( TOP OF EMPTY COMPUTATION STACK	* )	2	36
( 08 USER RO )	( TOP OF EMPTY RETURN STACK	* )	3	36
0A USER TIB	( TERMINAL INPUT BUFFER	* )	4	36
0C USER WIDTH	( MAXIMUM NAME FIELD WIDTH	* )	5	36
0E USER WARNING	( CONTROL WARNING MODES	* )	6	36
10 USER FENCE	CR ( BARRIER FOR FORGETTING	* )	7	36
12 USER DP	( DICTIONARY POINTER	* )	8	36
14 USER VOC-LINK	( TO NEWEST VOCABULARY	* )	9	36
16 USER BLK	( INTERPRETATION BLOCK	* )	10	36
18 USER IN	( OFFSET INTO SOURCE TEXT	* )	11	36
1A USER OUT	( DISPLAY CURSOR POSITION	* )	12	36
1C USER SCR	( EDITING SCREEN	* )	13	36
1E USER OFFSET	( POSSIBLY TO OTHER DRIVES	* )	1	37
20 USER CONTEXT	( VOCABULARY FIRST SEARCHED	* )	2	37
22 USER CURRENT	( SEARCHED SECOND, COMPILED INTO	* )	3	37
24 USER STATE	( COMPILATION STATE	* )	4	37
26 USER BASE	CR ( FOR NUMERIC INPUT-OUTPUT	* )	5	37
28 USER DPL	( DECIMAL POINT LOCATION	* )	6	37
2A USER FLD	( OUTPUT FIELD WIDTH	* )	7	37
2C USER CSP	( CHECK STACK POSITION	* )	8	37
2E USER R#	( EDITING CURSOR POSITION	* )	9	37
30 USER HLD	( POINTS TO LAST CHARACTER HELD IN PAD	* )	10	37
: 1+ 1 + ;	( INCREMENT STACK NUMBER BY ONE	* )	1	38
: 2+ 2 + ;	( INCREMENT STACK NUMBER BY TWO	* )	2	38
: HERE DP @ ;	( FETCH NEXT FREE ADDRESS IN DICT.	* )	3	38
: ALLOT DP +! ;	( MOVE DICT. POINTER AHEAD	* )	4	38
: , HERE ! 2 ALLOT ;	CR ( ENTER STACK NUMBER TO DICT.	* )	5	38
: C, HERE C! 1 ALLOT ;	( ENTER STACK BYTE TO DICT.	* )	6	38
: - MINUS + ;	( LEAVE DIFF. SEC - BOTTOM	* )	7	38
: = - 0= ;	( LEAVE BOOLEAN OF EQUALITY	* )	8	38
: < - 0< ;	( LEAVE BOOLEAN OF SEC < BOT	* )	9	38
: > SWAP < ;	( LEAVE BOOLEAN OF SEC > BOT	* )	10	38
: ROT >R SWAP R> SWAP ;	( ROTATE THIRD TO BOTTOM	* )	11	38
: SPACE BL EMIT ;	CR ( PRINT BLANK ON TERMINAL	* )	12	38
: -DUP DUP IF DUP ENDIF ;	( DUPLICATE NON-ZERO	* )	13	38
: TRAVERSE	( MOVE ACROSS NAME FIELD	* )	1	39
	( ADDRESS-2, DIRECTION-1, I.E. -1=R TO L, +1=L TO R	* )	2	39
: LATEST CURRENT @ @ ;	( NFA OF LATEST WORD	* )	6	39
: LFA 4 - ;	( CONVERT A WORDS PFA TO LFA	* )	11	39
: CFA 2 - ;	CR ( CONVERT A WORDS PFA TO CFA	* )	12	39
: NFA 5 - -1 TRAVERSE ;	( CONVERT A WORDS PFA TO NFA	* )	13	39
: PFA 1 TRAVERSE 5 + ;	( CONVERT A WORDS NFA TO PFA	* )	14	39
: !CSP SP@ CSP ! ;	( SAVE STACK POSITION IN 'CSP'	* )	1	40

:	?ERROR	( BOOLEAN-2, ERROR TYPE-1, WARN FOR TRUE *)	3 40	
:	?COMP STATE @ 0= 11 ?ERROR ;	( ERROR IF NOT COMPILING *)	6 40	
:	?EXEC STATE @ 12 ?ERROR ;	( ERROR IF NOT EXECUTING *)	8 40	
:	?PAIRS - 13 ?ERROR ;	( VERIFY STACK VALUES ARE PAIRED *)	10 40	
:	?CSP SP@ CSP @ - 14 ?ERROR ;	( VERIFY STACK POSITION *)	12 40	
:	?LOADING	( VERIFY LOADING FROM DISC *)	14 40	
:	COMPILE	( COMPILE THE EXECUTION ADDRESS FOLLOWING *)	2 41	
:	[ 0 STATE ! ; IMMEDIATE	( STOP COMPILATION *)	5 41	
:	] CO STATE ! ;	( ENTER COMPILATION STATE *)	7 41	
:	SMUDGE LATEST 20 TOGGLE ;	( ALTER LATEST WORD NAME *)	9 41	
:	HEX 10 BASE ! ;	( MAKE HEX THE IN-OUT BASE *)	11 41	
:	DECIMAL 0A BASE ! ;	( MAKE DECIMAL THE IN-OUT BASE *)	13 41	
:	(;CODE)	( WRITE CODE FIELD POINTING TO CALLING ADDRESS *)	2 42	
:	;CODE	( TERMINATE A NEW DEFINING WORD *)	6 42	
:	<BUILDS 0 CONSTANT ;	( CREATE HEADER FOR 'DOES' WORD *)	2 43	
:	DOES>	( REWRITE PFA WITH CALLING HI-LEVEL ADDRESS *)	4 43	
		( REWRITE CFA WITH 'DOES' CODE *)	5 43	
:	COUNT DUP 1+ SWAP C@ ;	( LEAVE TEXT ADDR. CHAR. COUNT *)	1 44	
:	TYPE	( TYPE STRING FROM ADDRESS-2, CHAR.COUNT-1 *)	2 44	
:	-TRAILING	( ADJUST CHAR. COUNT TO DROP TRAILING BLANKS *)	5 44	
:	(.)	( TYPE IN-LINE STRING, ADJUSTING RETURN *)	8 44	
:	."	22 STATE @	( COMPILE OR PRINT QUOTED STRING *)	12 44
:	EXPECT	( TERMINAL INPUT MEMORY-2, CHAR LIMIT-1 *)	2 45	
:	X BLK @	( END-OF-TEXT IS NULL *)	11 45	
:	FILL	( FILL MEMORY BEGIN-3, QUAN-2, BYTE-1 *)	1 46	
:	ERASE	( FILL MEMORY WITH ZEROS BEGIN-2, QUAN-1 *)	4 46	
:	BLANKS	( FILL WITH BLANKS BEGIN-2, QUAN-1 *)	7 46	
:	HOLD	( HOLD CHARACTER IN PAD *)	10 46	
:	PAD HERE 44 + ;	( PAD IS 68 BYTES ABOVE HERE *)	13 46	
		( DOWNWARD HAS NUMERIC OUTPUTS; UPWARD MAY HOLD TEXT *)	14 46	
:	WORD	( ENTER WITH DELIMITER, MOVE STRING TO 'HERE' *)	1 47	
:	(NUMBER)	( CONVERT DOUBLE NUMBER, LEAVING UNCONV. ADDR. *)	1 48	
:	NUMBER	( ENTER W/ STRING ADDR. LEAVE DOUBLE NUMBER *)	6 48	
:	-FIND	( RETURN PFA-3, LEN BYTE-2, TRUE-1; ELSE FALSE *)	12 48	
:	(ABORT) GAP	( ABORT ) ; ( USER ALTERABLE ERROR ABORT *)	2 49	
:	ERROR	( WARNING: -1=ABORT, 0=NO DISC, 1=DISC *)	4 49	
	WARNING @ 0<	( PRINT TEXT LINE REL TO SCR #4 *)	5 49	
:	ID.	( PRINT NAME FIELD FROM ITS HEADER ADDRESS *)	9 49	
:	CREATE	( A SMUDGED CODE HEADER TO PARAM FIELD *)	2 50	
		( WARNING IF DUPLICATING A CURRENT NAME *)	3 50	
:	[COMPILE]	( FORCE COMPILATION OF AN IMMEDIATE WORD *)	2 51	
:	LITERAL	( IF COMPILING, CREATE LITERAL *)	5 51	
:	DLITERAL	( IF COMPILING, CREATE DOUBLE LITERAL *)	8 51	
:	?STACK	( QUESTION UPON OVER OR UNDERFLOW OF STACK *)	13 51	
:	INTERPRET	( INTERPRET OR COMPILE SOURCE TEXT INPUT WORDS *)	2 52	
:	IMMEDIATE	( TOGGLE PREC. BIT OF LATEST CURRENT WORD *)	1 53	
:	VOCABULARY	( CREATE VOCAB WITH 'V-HEAD' AT VOC INTERSECT. *)	4 53	
VOCABULARY	FORTH	IMMEDIATE	( THE TRUNK VOCABULARY *)	9 53
:	DEFINITIONS	( SET THE CONTEXT ALSO AS CURRENT VOCAB *)	11 53	
:	(	( SKIP INPUT TEXT UNTIL RIGHT PARENTHESIS *)	14 53	
:	QUIT	( RESTART, INTERPRET FROM TERMINAL *)	2 54	
:	ABORT	( WARM RESTART, INCLUDING REGISTERS *)	7 54	
CODE COLD		( COLD START, INITIALIZING USER AREA *)	1 55	
CODE S->D		( EXTEND SINGLE INTEGER TO DOUBLE *)	1 56	
:	+- 0< IF MINUS ENDIF ;	( APPLY SIGN TO NUMBER BENEATH *)	4 56	
:	D+-	( APPLY SIGN TO DOUBLE NUMBER BENEATH *)	6 56	
:	ABS DUP +- ;	( LEAVE ABSOLUTE VALUE *)	9 56	

: DABS	DUP D+- ;	( DOUBLE INTEGER ABSOLUTE VALUE *)	10	56
: MIN		( LEAVE SMALLER OF TWO NUMBERS *)	12	56
: MAX		( LEAVE LARGER OF TWO NUMBERS *)	14	56
: M*	( LEAVE SIGNED DOUBLE PRODUCT OF TWO SINGLE NUMBERS *)		1	57
: M/	( FROM SIGNED DOUBLE-3-2, SIGNED DIVISOR-1 *)		3	57
	( LEAVE SIGNED REMAINDER-2, SIGNED QUOTIENT-1 *)		4	57
: *	U* DROP ;	( SIGNED PRODUCT *)	7	57
: /MOD	>R S->D R> M/ ;	( LEAVE REM-2, QUOT-1 *)	8	57
: /	/MOD SWAP DROP ;	( LEAVE QUOTIENT *)	9	57
: MOD	/MOD DROP ; CR	( LEAVE REMAINDER *)	10	57
: */MOD	( TAKE RATIO OF THREE NUMBERS, LEAVING *)		11	57
	>R M* R> M/ ;	( REM-2, QUOTIENT-1 *)	12	57
: */	*/MOD SWAP DROP ;	( LEAVE RATIO OF THREE NUMBERS *)	13	57
: M/MOD	( DOUBLE, SINGLE DIVISOR ... REMAINDER, DOUBLE *)		14	57
FIRST VARIABLE USE	( NEXT BUFFER TO USE, STALEST *)		1	58
FIRST VARIABLE PREV	( MOST RECENTLY REFERENCED BUFFER *)		2	58
: +BUF	( ADVANCE ADDRESS-1 TO NEXT BUFFER. RETURNS FALSE *)		4	58
84	( I.E. B/BUF+4 ) + DUP LIMIT =	( IF AT PREV *)	5	58
: UPDATE	( MARK THE BUFFER POINTED TO BY PREV AS ALTERED *)		8	58
: EMPTY-BUFFERS	( CLEAR BLOCK BUFFERS; DON'T WRITE TO DISC *)		11	58
: DRO	0 OFFSET ! ;	( SELECT DRIVE #0 *)	14	58
: DR1	07D0 OFFSET ! ; -->	( SELECT DRIVE #1 *)	15	58
: BUFFER	( CONVERT BLOCK# TO STORAGE ADDRESS *)		1	59
: BLOCK	( CONVERT BLOCK NUMBER TO ITS BUFFER ADDRESS *)		1	60
: (LINE)	( LINE#, SCR#, ... BUFFER ADDRESS, 64 COUNT *)		2	61
: .LINE	( LINE#, SCR#, ... PRINTED *)		6	61
: MESSAGE	( PRINT LINE RELATIVE TO SCREEN #4 OF DRIVE 0 *)		9	61
: LOAD	( INTERPRET SCREENS FROM DISC *)		2	62
: -->	( CONTINUE INTERPRETATION ON NEXT SCREEN *)		6	62
6900	CONSTANT DATA	( CONTROLLER PORT *)	1	65
6901	CONSTANT STATUS	( CONTROLLER PORT *)	2	65
: #HL	( CONVERT DECIMAL DIGIT FOR DISC CONTROLLER *)		5	65
CODE D/CHAR	( TEST CHAR-1. EXIT TEST BOOL-2, NEW CHAR-1 *)		1	66
: ?DISC	( UPON NAK SHOW ERR MSG, QUIT. ABSORBS TILL *)		7	66
1	D/CHAR >R 0=	( EOT, EXCEPT FOR SOH *)	8	66
CODE BLOCK-WRITE	( SEND TO DISC FROM ADDRESS-2, COUNT-1 *)		1	67
2	# LDA, SETUP JSR,	( WITH EOT AT END *)	2	67
CODE BLOCK-READ	( BUF.ADDR-1. EXIT AT 128 CHAR OR CONTROL *)		2	68
	( C = I TO READ, O TO WRITE *)		3	69
: R/W		( READ/WRITE DISC BLOCK *)	4	69
	( BUFFER ADDRESS-3, BLOCK #-2, I=READ O=WRITE *)		5	69
:	( FIND NEXT WORDS PFA; COMPILE IT, IF COMPILING *)		2	72
: FORGET	( FOLLOWING WORD FROM CURRENT VOCABULARY *)		6	72
: \	( SKIP INTERPRETATION OF THE REMAINDER OF LINE *)		11	72
: BACK	HERE - , ;	( RESOLVE BACKWARD BRANCH *)	1	73
: D.R	( DOUBLE INTEGER OUTPUT, RIGHT ALIGNED IN FIELD *)		1	76
: D.	0 D.R SPACE ;	( DOUBLE INTEGER OUTPUT *)	5	76
: .R	>R S->D R> D.R ;	( ALIGNED SINGLE INTEGER *)	7	76
: .	S->D D. ;	( SINGLE INTEGER OUTPUT *)	9	76
: ?	@ . ;	( PRINT CONTENTS OF MEMORY *)	11	76
: LIST		( LIST SCREEN BY NUMBER ON STACK *)	2	77
: INDEX	( PRINT FIRST LINE OF EACH SCREEN FROM-2, TO-1 *)		7	77
: TRIAD	( PRINT 3 SCREENS ON PAGE, CONTAINING # ON STACK *)		12	77
: VLIST	( LIST CONTEXT VOCABULARY *)		2	78
CREATE MON	( CALL MONITOR, SAVING RE-ENTRY TO FORTH *)		3	79 OK

## FORTH MODEL IMPLEMENTATION

This model is presented for the serious student as both an example of a large FORTH program and as a complete nucleus of FORTH. That is, it is sufficient to run and to continue to compile itself.

When compiled, the model requires about 2800 bytes of memory. An expanded version with formatted output and compiling aids would require about 4000 bytes. A 'full' implementation usually requires 6000 to 7000 bytes (including editor, assembler, and disk interface).

The following information consists of word definitions you will find in the CODE definitions. These are dependent on the micro-computer used, these being for the MOS Technology 5602.

Note that the notation in the CODE definitions is 'reverse Polish' as is all of FORTH. This means that the operand comes before the operator. Each equivalent of a 'line' of assembly code has a symbolic operand, then any address mode modifier, and finally the op-code mnemonic. (Note that words that generate actual machine code end in a ',' ; i.e. LDA, ). Therefor:

BOT 1+ LDA,               in FORTH would be:  
LDA 1,X                    in usual assembler.

And also:

POINTER )Y STA,           in FORTH would be:  
STA (POINTER),Y          in usual assembler.

It takes a bit of getting used to, but reverse Polish assembler allows full use of FORTH in evaluation of expressions and the easy generation of the equivalent of macros.

## GLOSSARY OF FORTH MODEL

IP	address of the Interpretive Pointer in zero-page.
W	address of the code field pointer in zero-page.
N	address of an 8 byte scratch area in zero-page.
XSAVE	address of a temporary register for X in zero-page.
UP	address of the User Pointer in zero-page.

## GLOSSARY OF FORTH MODEL, cont.

- A specify accumulator address mode.
- # specify immediate mode for machine byte literals.
- ,X ,Y specify memory indexed address mode.
- X) )Y specify indirect memory reference by a zero-page register.
- BOT address of high byte of a 16-bit stack item with ,X address mode. X register locates computation stack in zero-page, relative to address \$0000.
- BOT 1+ address of the low byte of the bottom stack item, with ,X mode preset.
- SEC and SEC 1+ address the second stack item as for BOT.
- TSX, move the return stack pointer (which is located in the CPU machine stack in page-one) to X register.
- R address of low byte of return stack with ,X mode preset.
- R n + address of the n-th byte of the return stack with ,X mode preset. Note that the low byte is at low memory, so 1+ gets the high byte, and 3+ gets the high byte of the second item of return stack.
- PUT address of routine to replace the present computation stack high byte from accumulator, and put from the machine stack one byte which replaces the present low stack byte; continue on to NEXT.
- PUSH address of routine to repeat PUT but creating a new bottom item on the computation stack.
- PUSHOA PUTOA address of routine to place the accumulator at the low stack byte, with the high byte zero. PUTOA over-writes, while PUSHOA creates new item.
- POP POPTWO address of routine to remove one or two 16-bit items from computation stack.
- BINARY address of routine to pop one item and PUT the accumulator (high) and ML stack (low) over what was second.
- SETUP address of a routine to move 16-bit items to zero-page. Item quantity is in accumulator.
- NEXT address of the inner-interpreter, to which all code routines must return. NEXT fetches indirectly referred to IP the next compiled FORTH word address. It then jumps indirectly to pointed machine code.

SCR # 6

```

0 ( INPUT-OUTPUT, TIM WFR-780519 )
1 CODE EMIT XSAVE STX, BOT 1+ LDA, 7F # AND,
2           72C6 JSR, XSAVE LDX, POP JMP,
3 CODE KEY XSAVE STX, BEGIN, BEGIN, 8 # LDX,
4           BEGIN, 6E02 LDA, .A LSR, CS END, 7320 JSR,
5           BEGIN, 731D JSR, 0 X) CMP, 0 X) CMP, 0 X) CMP,
6           0 X) CMP, 0 X) CMP, 6E02 LDA, .A LSR, PHP, TYA,
7           .A LSR, PLP, CS IF, 80 # ORA, THEN, TAY, DEX,
8           0= END, 731D JSR, FF # EOR, 7F # AND, 0= NOT END,
9           7F # CMP, 0= NOT END, XSAVE LDX, PUSHOA JMP,
10 CODE CR XSAVE STX, 728A JSR, XSAVE LDX, NEXT JMP,
11
12 CODE ?TERMINAL 1 # LDA, 6E02 BIT, 0= NOT IF,
13     BEGIN, 731D JSR, 6E02 BIT, 0= END, INY, THEN,
14     TYA, PUSHOA JMP,
15 DECIMAL ;S

```

SCR # 7

```

0 ( INPUT-OUTPUT, APPLE WFR-780730 )
1 CODE HOME FC58 JSR, NEXT JMP,
2 CODE SCROLL FC70 JSR, NEXT JMP,
3
4 HERE ' KEY 2 - ! ( POINT KEY TO HERE )
5 FDOC JSR, 7F # AND, PUSHOA JMP,
6 HERE ' EMIT 2 - ! ( POINT EMIT TO HERE )
7 BOT 1+ LDA, 80 # ORA, FDDE JSR, POP JMP,
8 HERE ' CR 2 - ! ( POINT CR TO HERE )
9 FD8E JSR, NEXT JMP,
10 HERE ' ?TERMINAL 2 - ! ( POINT ?TERM TO HERE )
11 COOO BIT, 0<
12     IF, BEGIN, CO10 BIT, COOO BIT, 0< NOT END, INY,
13     THEN, TYA, PUSHOA JMP,
14
15 DECIMAL ;S

```

SCR # 8

```

0 ( INPUT-OUTPUT, SYM-1 WFR-781015 )
1 HEX
2 CODE KEY 8A58 JSR, 7F # AND, PUSHOA JMP,
3
4 CODE EMIT BOT 1+ LDA, 8A47 JSR, POP JMP,
5
6 CODE CR 834D JSR, NEXT JMP,
7
8 CODE ?TERMINAL ( BREAK TEST FOR ANY KEY )
9 8B3C JSR, CS
10 IF, BEGIN, 8B3C JSR, CS NOT END, INY, THEN,
11     TYA, PUSHOA JMP,
12
13
14
15 DECIMAL ;S

```

SCR # 12

```

0 ( COLD AND WARM ENTRY, USER PARAMETERS           WFR-79APR29 )
1 ASSEMBLER OBJECT MEM HEX
2 NOP, HERE JMP, ( WORD ALIGNED VECTOR TO COLD )
3 NOP, HERE JMP, ( WORD ALIGNED VECTOR TO WARM )
4 0000 , 0001 , ( CPU, AND REVISION PARAMETERS )
5 0000 ,          ( TOPMOST WORD IN FORTH VOCABULARY )
6 7F ,            ( BACKSPACE CHARACTER )
7 3BA0 ,          ( INITIAL USER AREA )
8 009E ,          ( INITIAL TOP OF STACK )
9 01FF ,          ( INITIAL TOP OF RETURN STACK )
10 0100 ,         ( TERMINAL INPUT BUFFER )
11 001F ,         ( INITIAL NAME FIELD WIDTH )
12 0001 ,         ( INITIAL WARNING = 1 )
13 0200 ,         ( INITIAL FENCE )
14 0000 ,         ( COLD START VALUE FOR DP )
15 0000 ,         ( COLD START VALUE FOR VOC-LINK ) -->

```

SCR # 13

```

0 ( START OF NUCLEUS, LIT, PUSH, PUT, NEXT      WFR-78DEC26 )
1 CODE LIT          ( PUSH FOLLOWING LITERAL TO STACK *)
2     IP )Y LDA, PHA, IP INC, 0= IF, IP 1+ INC, THEN,
3     IP )Y LDA,          IP INC, 0= IF, IP 1+ INC, THEN,
4 LABEL PUSH        ( PUSH ACCUM AS HI-BYTE, ML STACK AS LO-BYTE *)
5     DEX, DEX,
6 LABEL PUT         ( REPLACE BOTTOM WITH ACCUM. AND ML STACK *)
7     BOT 1+ STA, PLA, BOT STA,
8 LABEL NEXT        ( EXECUTE NEXT FORTH ADDRESS, MOVING IP *)
9     1 # LDY, IP )Y LDA, W 1+ STA,          ( FETCH CODE ADDRESS )
10    DEY, IP )Y LDA, W STA,
11    CLC, IP LDA, 2 # ADC, IP STA,          ( MOVE IP AHEAD )
12    CS IF, IP 1+ INC, THEN,
13    W 1 - JMP,      ( JUMP INDIR. VIA W THRU CODE FIELD TO CODE )
14
15 -->

```

SCR # 14

```

0 ( SETUP           WFR-790225 )
1             HERE 2+ ,          ( MAKE SILENT WORD *)
2     IP )Y LDA, PHA, TYA, 'T LIT OB + 0= NOT END,
3
4 LABEL SETUP       ( MOVE # ITEMS FROM STACK TO 'N' AREA OF Z-PAGE *)
5     .A ASL, N 1 - STA,
6     BEGIN, BOT LDA, N ,Y STA, INX, INY,
7     N 1 - CPY, 0= END, 0 # LDY, RTS,
8
9 CODE EXECUTE     ( EXECUTE A WORD BY ITS CODE FIELD *)
10            ( ADDRESS ON THE STACK *)
11     BOT LDA, W STA, BOT 1+ LDA, W 1+ STA,
12     INX, INX, W 1 - JMP,
13
14
15 -->

```

SCR # 15

```

0 ( BRANCH, OBRANCH      W/16-BIT OFFSET          WFR-79APR01 )
1 CODE BRANCH            ( ADJUST IP BY IN-LINE 16 BIT LITERAL *)
2   CLC, IP )Y LDA, IP   ADC,                   PHA,
3   INY, IP )Y LDA, IP 1+ ADC, IP 1+ STA,
4                           PLA, IP STA,    NEXT 2+ JMP,
5

6 CODE OBRANCH           ( IF BOT IS ZERO, BRANCH FROM LITERAL *)
7   INX, INX, FE ,X LDA, FF ,X ORA,
8   ' BRANCH 0= NOT END, ( USE 'BRANCH' FOR FALSE )
9 LABEL BUMP:             ( TRUE JUST MOVES IP 2 BYTES *)
10  CLC, IP LDA, 2 # ADC, IP STA,
11  CS IF, IP 1+ INC, THEN, NEXT JMP,
12
13 -->
14
15

```

SCR # 16

```

0 ( LOOP CONTROL          WFR-79MAR20 )
1 CODE (LOOP)             ( INCREMENT LOOP INDEX, LOOP UNTIL => LIMIT *)
2   XSAVE STX, TSX, R INC, 0= IF, R 1+ INC, THEN,
3   LABEL L1: CLC, R 2+ LDA, R SBC, R 3 + LDA, R 1+ SBC,
4   LABEL L2: XSAVE LDX,   ( LIMIT-INDEX-1 )
5   .A ASL, ' BRANCH CS END, ( BRANCH UNTIL D7 SIGN=1 )
6   PLA, PLA, PLA, BUMP: JMP, ( ELSE EXIT LOOP )
7
8 CODE (+LOOP)            ( INCREMENT INDEX BY STACK VALUE +/-   *)
9   INX, INX, XSAVE STX, ( POP INCREMENT )
10  FF ,X LDA, PHA, PHA, FE ,X LDA, TSX, INX, INX,
11  CLC, R ADC, R STA, PLA, R 1 + ADC, R 1 + STA,
12  PLA, L1: 0< END,   ( AS FOR POSITIVE INCREMENT )
13  CLC, R LDA, R 2+ SBC, ( INDEX-LIMIT-1 )
14  R 1+ LDA, R 3 + SBC, L2: JMP,
15 -->

```

SCR # 17

```

0 ( (DO-                  WFR-79MAR30 )
1
2 CODE (DO)               ( MOVE TWO STACK ITEMS TO RETURN STACK *)
3   SEC 1+ LDA, PHA, SEC LDA, PHA,
4   BOT 1+ LDA, PHA, BOT LDA, PHA,
5
6 LABEL POPTWO            INX, INX,
7 LABEL POP                INX, INX, NEXT JMP,
8
9 CODE I                  ( COPY CURRENT LOOP INDEX TO STACK *)
10                         ( THIS WILL LATER BE POINTED TO 'R' )
11
12 -->
13
14
15

```

SCR # 18 WFR-781202 )

```

0 ( DIGIT
1 CODE DIGIT      ( CONVERT ASCII CHAR-SECOND, WITH BASE-BOTTOM *)
2           ( IF OK RETURN DIGIT-SECOND, TRUE-BOTTOM; *)
3           ( OTHERWISE FALSE-BOTTOM. *)
4 SEC, SEC LDA, 30 # SBC,
5 0< NOT IF, 0A # CMP, ( ADJUST FOR ASCII LETTER )
6           0< NOT IF, SEC, 07 # SBC, 0A # CMP,
7           0< NOT IF,
8 SWAP ( AT COMPILE TIME ) THEN, BOT CMP, ( TO BASE )
9           0< IF, SEC STA, 1 # LDA,
10          PHA, TYA, PUT JMP,
11          ( STORE RESULT SECOND AND RETURN TRUE )
12 THEN, THEN, THEN, ( CONVERSION FAILED )
13 TYA, PHA, INX, INX, PUT JMP, ( LEAVE BOOLEAN FALSE )
14
15 -->

```

SCR # 19 WFR-790225 )

```

0 ( FIND FOR VARIABLE LENGTH NAMES
1 CODE (FIND) ( HERE, NFA ... PFA, LEN BYTE, TRUE; ELSE FALSE *)
2 2 # LDA, SETUP JSR, XSAVE STX,
3 BEGIN, 0 # LDY, N )Y LDA, N 2+ )Y EOR, 3F # AND, 0=
4 IF, ( GOOD ) BEGIN, INY, N )Y LDA, N 2+ )Y EOR, .A ASL, 0=
5 IF, ( STILL GOOD ) SWAP CS ( LOOP TILL D7 SET )
6 END, XSAVE LDX, DEX, DEX, DEX, CLC,
7           TYA, 5 # ADC, N ADC, SEC STA, 0 # LDY,
8           TYA, N 1+ ADC, SEC 1+ STA, BOT 1+ STY,
9           N )Y LDA, BOT STA, 1 # LDA, PHA, PUSH JMP, ( FALSE )
10 THEN, CS NOT ( AT LAST CHAR? ) IF, SWAP THEN,
11 BEGIN, INY, N )Y LDA, 0< END, ( TO LAST CHAR )
12 THEN, INY, ( TO LINK ) N )Y LDA, TAX, INY,
13           N )Y LDA, N 1+ STA, N STX, N ORA, ( O LINK ? )
14 O= END, ( LOOP FOR ANOTHER NAME )
15 XSAVE LDX, 0 # LDA, PHA, PUSH JMP, ( FALSE ) -->

```

SCR # 20 WFR-780926 )

```

0 ( ENCLOSURE
1 CODE ENCLOSURE ( ENTER WITH ADDRESS-2, DELIM-1. RETURN WITH *)
2 ( ADDR-4, AND OFFSET TO FIRST CH-3, END WORD-2, NEXT CH-1 *)
3 2 # LDA, SETUP JSR, TXA, SEC, 8 # SBC, TAX,
4 SEC 1+ STY, BOT 1+ STY, ( CLEAR HI BYTES ) DEY,
5 BEGIN, INY, N 2+ )Y LDA, ( FETCH CHAR )
6           N CMP, 0= NOT END, ( STEP OVER LEADING DELIMITERS )
7 BOT 4 + STY, ( SAVE OFFSET TO FIRST CHAR )
8 BEGIN, N 2+ )Y LDA, 0=
9 IF, ( NULL ) SEC STY, ( IN EW ) BOT STY, ( IN NC )
10           TYA, BOT 4 + CMP, 0=
11           IF, ( Y=FC ) SEC INC, ( BUMP EW ) THEN, NEXT JMP,
12           THEN, SEC STY, ( IN EW ) INY, N CMP, ( DELIM ? )
13 O= END, ( IS DELIM ) BOT STY, ( IN NC ) NEXT JMP,
14
15 -->

```

SCR # 21

```

0 ( TERMINAL VECTORS WFR-79MAR30 )
1 ( THESE WORDS ARE CREATED WITH NO EXECUTION CODE, YET. )
2 ( THEIR CODE FIELDS WILL BE FILLED WITH THE ADDRESS OF THEIR )
3 ( INSTALLATION SPECIFIC CODE. )

4
5 CODE EMIT          ( PRINT ASCII VALUE ON BOTTOM OF STACK *)
6
7 CODE KEY           ( ACCEPT ONE TERMINAL CHARACTER TO THE STACK *)
8
9 CODE ?TERMINAL     ( 'BREAK' LEAVES 1 ON STACK; OTHERWISE 0 *)
10
11 CODE CR            ( EXECUTE CAR. RETURN, LINE FEED ON TERMINAL *)
12
13 -->
14
15

```

SCR # 22

```

0 ( CMOVE, WFR-79MAR20 )
1 CODE CMOVE   ( WITHIN MEMORY; ENTER W/ FROM-3, TO-2, QUAN-1 *)
2   3 # LDA, SETUP JSR,      ( MOVE 3 ITEMS TO 'N' AREA )
3   BEGIN, BEGIN, N CPY, 0= ( DECREMENT BYTE COUNTER AT 'N' )
4       IF, N 1+ DEC, 0<    ( EXIT WHEN DONE )
5       IF, NEXT JMP, THEN, THEN,
6           N 4 + )Y LDA, N 2+ )Y STA, INY, 0=
7           END,          ( LOOP TILL Y WRAPS, 22 CYCLES/BYTE )
8           N 5 + INC, N 3 + INC,   ( BUMP HI BYTES OF POINTERS )
9           JMP, ( BACK TO FIRST 'BEGIN' )
10
11 -->
12
13
14
15

```

SCR # 23

```

0 ( U*, UNSIGNED MULTIPLY FOR 16 BITS WFR-79APR08 )
1 CODE U*        ( 16 BIT MULTIPLICAND-2, 16 BIT MULTIPLIER-1 *)
2           ( 32 BIT UNSIGNED PRODUCT: LO WORD-2, HI WORD-1 *)
3   SEC LDA, N STA, SEC STY,
4   SEC 1+ LDA, N 1+ STA, SEC 1+ STY, ( MULTIPLICAND TO N )
5   10 # LDY,
6   BEGIN, BOT 2+ ASL, BOT 3 + ROL, BOT ROL, BOT 1+ ROL,
7           ( DOUBLE PRODUCT WHILE SAMPLING D15 OF MULT )
8           CS IF, ( SET ) CLC,
9           ( ADD MULTIPLICAND TO PARTIAL PRODUCT LOW 24 BITS )
10          N LDA, BOT 2 + ADC, BOT 2 + STA,
11          N 1+ LDA, BOT 3 + ADC, BOT 3 + STA,
12          O # LDA, BOT ADC, BOT STA,
13          THEN, DEY, 0=
14          END,   NEXT JMP,
15 -->

```

SCR # 24

0 ( U/, UNSIGNED DIVIDE FOR 31 BITS WFR-79APR29 )  
1 CODE U/ ( 31 BIT DIVIDEND-2, -3, 16 BIT DIVISOR-1 \*)  
2 ( 16 BIT REMAINDER-2, 16 BIT QUOTIENT-1 \*)  
3 SEC 2 + LDA, SEC LDY, SEC 2 + STY, .A ASL, SEC STA,  
4 SEC 3 + LDA, SEC 1+ LDY, SEC 3 + STY, .A ROL, SEC 1+ STA,  
5 10 # LDA, N STA,  
6 BEGIN, SEC 2 + ROL, SEC 3 + ROL, SEC,  
7 SEC 2 + LDA, BOT SBC, TAY,  
8 SEC 3 + LDA, BOT 1+ SBC,  
9 CS IF, SEC 2+ STY, SEC 3 + STA, THEN,  
10 SEC ROL, SEC 1+ ROL,  
11 N DEC, 0=,  
12 END, POP JMP,  
13 -->  
14  
15

SCR # 25

0 ( LOGICALS WFR-79APR20 )  
1  
2 CODE AND ( LOGICAL BITWISE AND OF BOTTOM TWO ITEMS \*)  
3 BOT LDA, SEC AND, PHA,  
4 BOT 1+ LDA, SEC 1+ AND, INX, INX, PUT JMP,  
5  
6 CODE OR ( LOGICAL BITWISE 'OR' OF BOTTOM TWO ITEMS \*)  
7 BOT LDA, SEC ORA, PHA,  
8 BOT 1+ LDA, SEC 1 + ORA, INX, INX, PUT JMP,  
9  
10 CODE XOR ( LOGICAL 'EXCLUSIVE-OR' OF BOTTOM TWO ITEMS \*)  
11 BOT LDA, SEC EOR, PHA,  
12 BOT 1+ LDA, SEC 1+ EOR, INX, INX, PUT JMP,  
13  
14 -->  
15

SCR # 26

0 ( STACK INITIALIZATION WFR-79MAR30 )  
1 CODE SP@ ( FETCH STACK POINTER TO STACK \*)  
2 TXA,  
3 LABEL PUSHOA PHA, 0 # LDA, PUSH JMP,  
4  
5 CODE SP! ( LOAD SP FROM 'SO' \*)  
6 06 # LDY, UP )Y LDA, TAX, NEXT JMP,  
7  
8 CODE RP! ( LOAD RP FROM R0 \*)  
9 XSAVE STX, 08 # LDY, UP )Y LDA, TAX, TXS,  
10 XSAVE LDX, NEXT JMP,  
11  
12 CODE ;S ( RESTORE IP REGISTER FROM RETURN STACK \*)  
13 PLA, IP STA, PLA, IP 1+ STA, NEXT JMP,  
14  
15 -->

SCR # 27

0 ( RETURN STACK WORDS WFR-79MAR29 )  
1 CODE LEAVE ( FORCE EXIT OF DO-LOOP BY SETTING LIMIT \*)  
2 XSAVE STX, TSX, R LDA, R 2+ STA, ( TO INDEX \*)  
3 R 1+ LDA, R 3 + STA, XSAVE LDX, NEXT JMP,  
4  
5 CODE >R ( MOVE FROM COMP. STACK TO RETURN STACK \*)  
6 BOT 1+ LDA, PHA, BOT LDA, PHA, INX, INX, NEXT JMP,  
7  
8 CODE R> ( MOVE FROM RETURN STACK TO COMP. STACK \*)  
9 DEX, DEX, PLA, BOT STA, PLA, BOT 1+ STA, NEXT JMP,  
10  
11 CODE R ( COPY THE BOTTOM OF RETURN STACK TO COMP. STACK \*)  
12 XSAVE STX, TSX, R LDA, PHA, R 1+ LDA,  
13 XSAVE LDX, PUSH JMP,  
14 R -2 BYTE.IN I !  
15 -->

SCR # 28

0 ( TESTS AND LOGICALS WFR-79MAR19 )  
1  
2 CODE 0= ( REVERSE LOGICAL STATE OF BOTTOM OF STACK \*)  
3 BOT LDA, BOT 1+ ORA, BOT 1+ STY,  
4 0= IF, INY, THEN, BOT STY, NEXT JMP,  
5  
6 CODE 0< ( LEAVE TRUE IF NEGATIVE; OTHERWISE FALSE \*)  
7 BOT 1+ ASL, TYA, .A ROL, BOT 1+ STY, BOT STA, NEXT JMP,  
8  
9  
10 -->  
11  
12  
13  
14  
15

SCR # 29

0 ( MATH WFR-79MAR19 )  
1 CODE + ( LEAVE THE SUM OF THE BOTTOM TWO STACK ITEMS \*)  
2 CLC, BOT LDA, SEC ADC, SEC STA, BOT 1+ LDA, SEC 1+ ADC,  
3 SEC 1+ STA, INX, INX, NEXT JMP,  
4 CODE D+ ( ADD TWO DOUBLE INTEGERS, LEAVING DOUBLE \*)  
5 CLC, BOT 2 + LDA, BOT 6 + ADC, BOT 6 + STA,  
6 BOT 3 + LDA, BOT 7 + ADC, BOT 7 + STA,  
7 BOT LDA, BOT 4 + ADC, BOT 4 + STA,  
8 BOT 1 + LDA, BOT 5 + ADC, BOT 5 + STA, POPTWO JMP,  
9 CODE MINUS ( TWOS COMPLEMENT OF BOTTOM SINGLE NUMBER \*)  
10 SEC, TYA, BOT SBC, BOT STA,  
11 TYA, BOT 1+ SBC, BOT 1+ STA, NEXT JMP,  
12 CODE DMINUS ( TWOS COMPLEMENT OF BOTTOM DOUBLE NUMBER \*)  
13 SEC, TYA, BOT 2 + SBC, BOT 2 + STA,  
14 TYA, BOT 3 + SBC, BOT 3 + STA,  
15 1 BYTE.IN MINUS JMP, -->

SCR # 30

0 ( STACK MANIPULATION WFR-79MAR29 )  
1 CODE OVER ( DUPLICATE SECOND ITEM AS NEW BOTTOM \*)  
2 SEC LDA, PHA, SEC 1+ LDA, PUSH JMP,  
3  
4 CODE DROP ( DROP BOTTOM STACK ITEM \*)  
5 POP -2 BYTE.IN DROP ! ( C.F. VECTORS DIRECTLY TO 'POP' )  
6  
7 CODE SWAP ( EXCHANGE BOTTOM AND SECOND ITEMS ON STACK \*)  
8 SEC LDA, PHA, BOT LDA, SEC STA,  
9 SEC 1+ LDA, BOT 1+ LDY, SEC 1+ STY, PUT JMP,  
10  
11 CODE DUP ( DUPLICATE BOTTOM ITEM ON STACK \*)  
12 BOT LDA, PHA, BOT 1+ LDA, PUSH JMP,  
13  
14 -->  
15

SCR # 31

0 ( MEMORY INCREMENT, WFR-79MAR30 )  
1  
2 CODE +! ( ADD SECOND TO MEMORY 16 BITS ADDRESSED BY BOTTOM \*)  
3 CLC, BOT X) LDA, SEC ADC, BOT X) STA,  
4 BOT INC, O= IF, BOT 1+ INC, THEN,  
5 BOT X) LDA, SEC 1+ ADC, BOT X) STA, POPTWO JMP,  
6  
7 CODE TOGGLE ( BYTE AT ADDRESS-2, BIT PATTERN-1 ... \*)  
8 SEC X) LDA, BOT EOR, SEC X) STA, POPTWO JMP,  
9  
10 -->  
11  
12  
13  
14  
15

SCR # 32

0 ( MEMORY FETCH AND STORE WFR-781202 )  
1 CODE @ ( REPLACE STACK ADDRESS WITH 16 BIT \*)  
2 BOT X) LDA, PHA, ( CONTENTS OF THAT ADDRESS \*)  
3 BOT INC, O= IF, BOT 1+ INC, THEN, BOT X) LDA, PUT JMP,  
4  
5 CODE C@ ( REPLACE STACK ADDRESS WITH POINTED 8 BIT BYTE \*)  
6 BOT X) LDA, BOT STA, BOT 1+ STY, NEXT JMP,  
7  
8 CODE ! ( STORE SECOND AT 16 BITS ADDRESSED BY BOTTOM \*)  
9 SEC LDA, BOT X) STA, BOT INC, O= IF, BOT 1+ INC, THEN,  
10 SEC 1+ LDA, BOT X) STA, POPTWO JMP,  
11  
12 CODE C! ( STORE SECOND AT BYTE ADDRESSED BY BOTTOM \*)  
13 SEC LDA, BOT X) STA, POPTWO JMP,  
14  
15 DECIMAL ;S

SCR # 33

0 ( :, ;, WFR-79MAR30 )  
 1  
 2 : : ( CREATE NEW COLON-DEFINITION UNTIL ';' \*)  
 3 ?EXEC !CSP CURRENT @ CONTEXT !  
 4 CREATE ] ;CODE IMMEDIATE  
 5 IP 1+ LDA, PHA, IP LDA, PHA, CLC, W LDA, 2 # ADC,  
 6 IP STA, TYA, W 1+ ADC, IP 1+ STA, NEXT JMP,  
 7  
 8  
 9 : ; ( TERMINATE COLON-DEFINITION \*)  
 10 ?CSP COMPILE ;S  
 11 SMUDGE [ ; IMMEDIATE  
 12  
 13  
 14  
 15 -->

SCR # 34

0 ( CONSTANT, VARIABLE, USER WFR-79MAR30 )  
 1 : CONSTANT ( WORD WHICH LATER CREATES CONSTANTS \*)  
 2 CREATE SMUDGE , ;CODE  
 3 2 # LDY, W )Y LDA, PHA, INY, W )Y LDA, PUSH JMP,  
 4  
 5 : VARIABLE ( WORD WHICH LATER CREATES VARIABLES \*)  
 6 CONSTANT ;CODE  
 7 CLC, W LDA, 2 # ADC, PHA, TYA, W 1+ ADC, PUSH JMP,  
 8  
 9  
 10 : USER ( CREATE USER VARIABLE \*)  
 11 CONSTANT ;CODE  
 12 2 # LDY, CLC, W )Y LDA, UP ADC, PHA,  
 13 0 # LDA, UP 1+ ADC, PUSH JMP,  
 14  
 15 -->

SCR # 35

0 ( DEFINED CONSTANTS WFR-78MAR22 )  
 1 HEX  
 2 00 CONSTANT 0 01 CONSTANT 1  
 3 02 CONSTANT 2 03 CONSTANT 3  
 4 20 CONSTANT BL ( ASCII BLANK \*)  
 5 40 CONSTANT C/L ( TEXT CHARACTERS PER LINE \*)  
 6  
 7 3BEO CONSTANT FIRST ( FIRST BYTE RESERVED FOR BUFFERS \*)  
 8 4000 CONSTANT LIMIT ( JUST BEYOND TOP OF RAM \*)  
 9 80 CONSTANT B/BUF ( BYTES PER DISC BUFFER \*)  
 10 8 CONSTANT B/SCR ( BLOCKS PER SCREEN = 1024 B/BUF / \*)  
 11  
 12 00 +ORIGIN  
 13 : +ORIGIN LITERAL + ; ( LEAVES ADDRESS RELATIVE TO ORIGIN \*)  
 14 -->  
 15

SCR # 36

0	(	USER VARIABLES	WFR-78APR29 )
1	HEX	( 0 THRU 5 RESERVED,	REFERENCED TO \$00AO *)
2	( 06 USER SO )	( TOP OF EMPTY COMPUTATION STACK *)	
3	( 08 USER RO )	( TOP OF EMPTY RETURN STACK *)	
4	0A USER TIB	( TERMINAL INPUT BUFFER *)	
5	0C USER WIDTH	( MAXIMUM NAME FIELD WIDTH *)	
6	0E USER WARNING	( CONTROL WARNING MODES *)	
7	10 USER FENCE	( BARRIER FOR FORGETTING *)	
8	12 USER DP	( DICTIONARY POINTER *)	
9	14 USER VOC-LINK	( TO NEWEST VOCABULARY *)	
10	16 USER BLK	( INTERPRETATION BLOCK *)	
11	18 USER IN	( OFFSET INTO SOURCE TEXT *)	
12	1A USER OUT	( DISPLAY CURSOR POSITION *)	
13	1C USER SCR	( EDITING SCREEN *)	
14	-->		
15			

SCR # 37

0	(	USER VARIABLES, CONT.	WFR-79APR29 )
1	1E USER OFFSET	( POSSIBLY TO OTHER DRIVES *)	
2	20 USER CONTEXT	( VOCABULARY FIRST SEARCHED *)	
3	22 USER CURRENT	( SEARCHED SECOND, COMPILED INTO *)	
4	24 USER STATE	( COMPILATION STATE *)	
5	26 USER BASE	( FOR NUMERIC INPUT-OUTPUT *)	
6	28 USER DPL	( DECIMAL POINT LOCATION *)	
7	2A USER FLD	( OUTPUT FIELD WIDTH *)	
8	2C USER CSP	( CHECK STACK POSITION *)	
9	2E USER R#	( EDITING CURSOR POSITION *)	
10	30 USER HLD	( POINTS TO LAST CHARACTER HELD IN PAD *)	
11	-->		
12			
13			
14			
15			

SCR # 38

0	(	HI-LEVEL MISC.	WFR-79APR29 )
1	:	1+ 1 + ;	( INCREMENT STACK NUMBER BY ONE *)
2	:	2+ 2 + ;	( INCREMENT STACK NUMBER BY TWO *)
3	:	HERE DP @ ;	( FETCH NEXT FREE ADDRESS IN DICT. *)
4	:	ALLOT DP +! ;	( MOVE DICT. POINTER AHEAD *)
5	:	, HERE ! 2 ALLOT ;	( ENTER STACK NUMBER TO DICT. *)
6	:	C, HERE C! 1 ALLOT ;	( ENTER STACK BYTE TO DICT. *)
7	:	- MINUS + ;	( LEAVE DIFF. SEC - BOTTOM *)
8	:	= - 0= ;	( LEAVE BOOLEAN OF EQUALITY *)
9	:	< - 0< ;	( LEAVE BOOLEAN OF SEC < BOT *)
10	:	> SWAP < ;	( LEAVE BOOLEAN OF SEC > BOT *)
11	:	ROT >R SWAP R> SWAP ;	( ROTATE THIRD TO BOTTOM *)
12	:	SPACE BL EMIT ;	( PRINT BLANK ON TERMINAL *)
13	:	-DUP DUP IF DUP ENDIF ;	( DUPLICATE NON-ZERO *)
14	-->		
15			

SCR # 39

```

0 ( VARIABLE LENGTH NAME SUPPORT WFR-79MAR30 )
1 : TRAVERSE ( MOVE ACROSS NAME FIELD *)
2     ( ADDRESS-2, DIRECTION-1, I.E. -1=R TO L, +1=L TO R *)
3     SWAP
4     BEGIN OVER + 7F OVER C@ < UNTIL SWAP DROP ;
5
6 : LATEST      CURRENT @ @ ; ( NFA OF LATEST WORD *)
7
8
9 ( FOLLOWING HAVE LITERALS DEPENDENT ON COMPUTER WORD SIZE )
10
11 : LFA    4 - ; ( CONVERT A WORDS PFA TO LFA *)
12 : CFA    2 - ; ( CONVERT A WORDS PFA TO CFA *)
13 : NFA    5 - -1 TRAVERSE ; ( CONVERT A WORDS PFA TO NFA *)
14 : PFA    1 TRAVERSE 5 + ; ( CONVERT A WORDS NFA TO PFA *)
15     -->

```

SCR # 40

```

0 ( ERROR PROCEEDURES, PER SHIRA WFR-79MAR23 )
1 : !CSP     SP@ CSP ! ; ( SAVE STACK POSITION IN 'CSP' *)
2
3 : ?ERROR      ( BOOLEAN-2, ERROR TYPE-1, WARN FOR TRUE *)
4     SWAP IF     ERROR ELSE DROP ENDIF ;
5
6 : ?COMP      STATE @ 0= 11 ?ERROR ; ( ERROR IF NOT COMPILING *)
7
8 : ?EXEC      STATE @ 12 ?ERROR ; ( ERROR IF NOT EXECUTING *)
9
10 : ?PAIRS - 13 ?ERROR ; ( VERIFY STACK VALUES ARE PAIRED *)
11
12 : ?CSP     SP@ CSP @ - 14 ?ERROR ; ( VERIFY STACK POSITION *)
13
14 : ?LOADING      ( VERIFY LOADING FROM DISC *)
15     BLK @ 0= 16 ?ERROR ; -->

```

SCR # 41

```

0 ( COMPILE, SMUDGE, HEX, DECIMAL WFR-79APR20 )
1
2 : COMPILE      ( COMPILE THE EXECUTION ADDRESS FOLLOWING *)
3     ?COMP R> DUP 2+ >R @ , ;
4
5 : [ 0 STATE ! ; IMMEDIATE ( STOP COMPILATION *)
6
7 : ] CO STATE ! ; ( ENTER COMPILATION STATE *)
8
9 : SMUDGE      LATEST 20 TOGGLE ; ( ALTER LATEST WORD NAME *)
10
11 : HEX        10 BASE ! ; ( MAKE HEX THE IN-OUT BASE *)
12
13 : DECIMAL    0A BASE ! ; ( MAKE DECIMAL THE IN-OUT BASE *)
14 -->
15

```

SCR # 42

```

0 ( ;CODE
1
2 : (;CODE)      ( WRITE CODE FIELD POINTING TO CALLING ADDRESS *)
3     R> LATEST PFA CFA ! ;
4
5
6 : ;CODE          ( TERMINATE A NEW DEFINING WORD *)
7     ?CSP COMPILE (;CODE)
8     [COMPILE] [ SMUDGE ; IMMEDIATE
9 -->
10
11
12
13
14
15

```

SCR # 43

```

0 ( <BUILD, DOES>                               WFR-79MAR20 )
1
2 : <BUILDS 0 CONSTANT ; ( CREATE HEADER FOR 'DOES' WORD *)
3
4 : DOES>           ( REWRITE PFA WITH CALLING HI-LEVEL ADDRESS *)
5           ( REWRITE CFA WITH 'DOES' CODE *)
6     R> LATEST PFA ! ;CODE
7     IP 1+ LDA, PHA, IP LDA, PHA, ( BEGIN FORTH NESTING )
8     2 # LDY, W ) Y LDA, IP STA, ( FETCH FIRST PARAM )
9     INY, W ) Y LDA, IP 1+ STA, ( AS NEXT INTERP. PTR )
10    CLC, W LDA, 4 # ADC, PHA, ( PUSH ADDRESS OF PARAMS )
11    W 1+ LDA, 00 # ADC, PUSH JMP,
12
13 -->
14
15

```

SCR # 44

```

0 ( TEXT OUTPUTS                               WFR-79APR02 )
1 : COUNT DUP 1+ SWAP C@ ; ( LEAVE TEXT ADDR. CHAR. COUNT *)
2 : TYPE          ( TYPE STRING FROM ADDRESS-2, CHAR.COUNT-1 *)
3     -DUP IF OVER + SWAP
4           DO I C@ EMIT LOOP ELSE DROP ENDIF ;
5 : -TRAILING ( ADJUST CHAR. COUNT TO DROP TRAILING BLANKS *)
6     DUP 0 DO OVER OVER + 1 - C@
7     BL - IF LEAVE ELSE 1 - ENDIF LOOP ;
8 : (.)          ( TYPE IN-LINE STRING, ADJUSTING RETURN *)
9     R COUNT DUP 1+ R> + >R TYPE ;
10
11
12 : ."
13     22 STATE @ ( COMPILE OR PRINT QUOTED STRING *)
14     IF COMPILE (.) WORD HERE C@ 1+ ALLOT
15     ELSE WORD HERE COUNT TYPE ENDIF ;
16     IMMEDIATE -->

```

SCR # 45

```

0 ( TERMINAL INPUT                               WFR-79APR29 )
1
2 : EXPECT          ( TERMINAL INPUT MEMORY-2, CHAR LIMIT-1 *)
3     OVER + OVER DO KEY DUP OE +ORIGIN ( BS ) @ =
4     IF DROP 08 OVER I = DUP R> 2 - + >R -
5     ELSE ( NOT BS ) DUP OD =
6         IF ( RET ) LEAVE DROP BL 0 ELSE DUP ENDIF
7         I C! 0 I 1+ !
8     ENDIF EMIT LOOP DROP ;
9 : QUERY      TIB @ 50 EXPECT '0' IN ! ;
10 8081 HERE
11 : X BLK @           ( END-OF-TEXT IS NULL *)
12     IF ( DISC ) 1 BLK +! 0 IN ! BLK @ 7 AND 0=
13     IF ( SCR END ) ?EXEC R> DROP ENDIF
14     ELSE ( TERMINAL ) R> DROP
15     ENDIF ; ! IMMEDIATE -->

```

SCR # 46

```

0 ( FILL, ERASE, BLANKS, HOLD, PAD               WFR-79APR02 )
1 : FILL          ( FILL MEMORY BEGIN-3, QUAN-2, BYTE-1 *)
2     SWAP >R OVER C! DUP 1+ R> 1 - CMOVE ;
3
4 : ERASE          ( FILL MEMORY WITH ZEROS BEGIN-2, QUAN-1 *)
5     0 FILL ;
6
7 : BLANKS         ( FILL WITH BLANKS BEGIN-2, QUAN-1 *)
8     BL FILL ;
9
10 : HOLD          ( HOLD CHARACTER IN PAD *)
11     -1 HLD +! HLD @ C! ;
12
13 : PAD           HERE 44 + ;          ( PAD IS 68 BYTES ABOVE HERE *)
14     ( DOWNWARD HAS NUMERIC OUTPUTS; UPWARD MAY HOLD TEXT *)
15 -->

```

SCR # 47

```

0 ( WORD,                                         WFR-79APR02 )
1 : WORD          ( ENTER WITH DELIMITER, MOVE STRING TO 'HERE' *)
2     BLK @ IF BLK @ BLOCK ELSE TIB @ ENDIF
3     IN @ + SWAP ( ADDRESS-2, DELIMITER-1 )
4     ENCLOSE        ( ADDRESS-4, START-3, END-2, TOTAL COUNT-1 )
5     HERE 22 BLANKS ( PREPARE FIELD OF 34 BLANKS )
6     IN +!          ( STEP OVER THIS STRING )
7     OVER - >R      ( SAVE CHAR COUNT )
8     R HERE C!      ( LENGTH STORED FIRST )
9     + HERE 1+
10    R> CMOVE ;    ( MOVE STRING FROM BUFFER TO HERE+1 )
11
12
13
14
15 -->

```

SCR # 48

```

0 ( (NUMBER-, NUMBER, -FIND, WFR-79APR29 )
1 : (NUMBER) ( CONVERT DOUBLE NUMBER, LEAVING UNCONV. ADDR. *)
2 BEGIN 1+ DUP >R C@ BASE @ DIGIT
3 WHILE SWAP BASE @ U* DROP ROT BASE @ U* D+
4 DPL @ 1+ IF 1 DPL +! ENDIF R> REPEAT R> ;
5
6 : NUMBER ( ENTER W/ STRING ADDR. LEAVE DOUBLE NUMBER *)
7 0 0 ROT DUP 1+ C@ 2D = DUP >R + -1
8 BEGIN DPL ! (NUMBER) DUP C@ BL -
9 WHILE DUP C@ 2E - 0 ?ERROR 0 REPEAT
10 DROP R> IF DMINUS ENDIF ;
11
12 : -FIND ( RETURN PFA-3, LEN BYTE-2, TRUE-1; ELSE FALSE *)
13 BL WORD HERE CONTEXT @ @ (FIND)
14 DUP 0= IF DROP HERE LATEST (FIND) ENDIF ;
15 -->

```

SCR # 49

```

0 ( ERROR HANDLER WFR-79APR20 )
1
2 : (ABORT) ABORT ; ( USER ALTERABLE ERROR ABORT *)
3
4 : ERROR ( WARNING: -1=ABORT, 0=NO DISC, 1=DISC *)
5 WARNING @ 0< ( PRINT TEXT LINE REL TO SCR #4 *)
6 IF (ABORT) ENDIF HERE COUNT TYPE ." ? "
7 MESSAGE SP! IN @ BLK @ QUIT ;
8
9 : ID. ( PRINT NAME FIELD FROM ITS HEADER ADDRESS *)
10 PAD 020 5F FILL DUP PFA LFA OVER -
11 PAD SWAP CMOVE PAD COUNT 01F AND TYPE SPACE ;
12 -->
13
14
15

```

SCR # 50

```

0 ( CREATE WFR-79APR28 )
1
2 : CREATE ( A SMUDGED CODE HEADER TO PARAM FIELD *)
3 ( WARNING IF DUPLICATING A CURRENT NAME *)
4 TIB HERE 0AO + < 2 ?ERROR ( FREE SPACE ? )
5 -FIND ( CHECK IF UNIQUE IN CURRENT AND CONTEXT )
6 IF ( WARN USER ) DROP NFA ID.
7 4 MESSAGE SPACE ENDIF
8 HERE DUP C@ WIDTH @ MIN 1+ ALLOT
9 DP C@ OFD = ALLOT
10 DUP AO TOGGLE HERE 1 - 80 TOGGLE ( DELIMIT BITS )
11 LATEST , CURRENT @ !
12 HERE 2+ , ;
13 -->
14
15

```

SCR # 51

```

0 ( LITERAL, DLITERAL, [COMPILE], ?STACK           WFR-79APR29 )
1
2 : [COMPILE]          ( FORCE COMPIRATION OF AN IMMEDIATE WORD *)
3     -FIND 0= 0 ?ERROR DROP CFA , ; IMMEDIATE
4
5 : LITERAL            ( IF COMPILING, CREATE LITERAL *)
6     STATE @ IF COMPILE LIT , ENDIF ; IMMEDIATE
7
8 : DLITERAL           ( IF COMPILING, CREATE DOUBLE LITERAL *)
9     STATE @ IF SWAP [COMPILE] LITERAL
10    [COMPILE] LITERAL ENDIF ; IMMEDIATE
11
12 ( FOLLOWING DEFINITION IS INSTALLATION DEPENDENT )
13 : ?STACK            ( QUESTION UPON OVER OR UNDERFLOW OF STACK *)
14     09E SP@ < 1 ?ERROR SP@ 020 < 7 ?ERROR ;
15 -->

```

SCR # 52

```

0 ( INTERPRET,           WFR-79APR18 )
1
2 : INTERPRET          ( INTERPRET OR COMPILE SOURCE TEXT INPUT WORDS *)
3     BEGIN -FIND
4       IF ( FOUND ) STATE @ <
5         IF CFA , ELSE CFA EXECUTE ENDIF ?STACK
6       ELSE HERE NUMBER DPL @ 1+
7         IF [COMPILE] DLITERAL
8           ELSE DROP [COMPILE] LITERAL ENDIF ?STACK
9       ENDIF AGAIN ;
10 -->
11
12
13
14
15

```

SCR # 53

```

0 ( IMMEDIATE, VOCAB, DEFIN, FORTH, (      DJK-WFR-79APR29 )
1 : IMMEDIATE          ( TOGGLE PREC. BIT OF LATEST CURRENT WORD *)
2     LATEST 40 TOGGLE ;
3
4 : VOCABULARY          ( CREATE VOCAB WITH 'V-HEAD' AT VOC INTERSECT. *)
5   <BUILDS A081 , CURRENT @ CFA ,
6   HERE VOC-LINK @ , VOC-LINK !
7   DOES> 2+ CONTEXT ! ;
8
9 VOCABULARY FORTH IMMEDIATE          ( THE TRUNK VOCABULARY *)
10
11 : DEFINITIONS          ( SET THE CONTEXT ALSO AS CURRENT VOCAB *)
12   CONTEXT @ CURRENT ! ;
13
14 : (                  ( SKIP INPUT TEXT UNTIL RIGHT PARENTHESIS *)
15   29 WORD ; IMMEDIATE -->

```

SCR # 54

```

0 ( QUIT, ABORT
1
2 : QUIT ( RESTART, INTERPRET FROM TERMINAL *)
3     0 BLK ! [COMPILE] [
4     BEGIN RP! CR QUERY INTERPRET
5     STATE @ 0= IF ." OK" ENDIF AGAIN ;
6
7 : ABORT ( WARM RESTART, INCLUDING REGISTERS *)
8     SP! DECIMAL DRO
9     CR ." FORTH-65 V 4.0"
10    [COMPILE] FORTH DEFINITIONS QUIT ;
11
12
13 -->
14
15

```

SCR # 55

```

0 ( COLD START
1 CODE COLD ( COLD START, INITIALIZING USER AREA *)
2 HERE 02 +ORIGIN ! ( POINT COLD ENTRY TO HERE )
3     OC +ORIGIN LDA, 'T FORTH 4 + STA, ( FORTH VOCAB. )
4     OD +ORIGIN LDA, 'T FORTH 5 + STA,
5     15 # LDY, ( INDEX TO VOC-LINK ) 0= IF, ( FORCED )
6 HERE 06 +ORIGIN ! ( POINT RE-ENTRY TO HERE )
7     OF # LDY, ( INDEX TO WARNING ) THEN, ( FROM IF, )
8     10 +ORIGIN LDA, UP STA, ( LOAD UP )
9     11 +ORIGIN LDA, UP 1+ STA,
10    BEGIN, OC +ORIGIN ,Y LDA, ( FROM LITERAL AREA )
11          UP )Y STA, ( TO USER AREA )
12          DEY, 0< END,
13     'T ABORT 100 /MOD # LDA, IP 1+ STA,
14          # LDA, IP STA,
15     6C # LDA, W 1 - STA, 'T RP! JMP, ( RUN ) -->

```

SCR # 56

```

0 ( MATH UTILITY
1 CODE S->D ( EXTEND SINGLE INTEGER TO DOUBLE *)
2     BOT 1+ LDA, 0< IF, DEY, THEN, TYA, PHA, PUSH JMP,
3
4 : +- 0< IF MINUS ENDIF ; ( APPLY SIGN TO NUMBER BENEATH *)
5
6 : D+- ( APPLY SIGN TO DOUBLE NUMBER BENEATH *)
7     0< IF DMINUS ENDIF ;
8
9 : ABS DUP +- ; ( LEAVE ABSOLUTE VALUE *)
10: DABS DUP D+- ; ( DOUBLE INTEGER ABSOLUTE VALUE *)
11
12: MIN OVER OVER > IF SWAP ENDIF DROP ;
13: MAX OVER OVER < IF SWAP ENDIF DROP ; -->
14
15

```

SCR # 57

```

0 ( MATH PACKAGE                               DJK-WFR-79APR29 )
1 : M*      ( LEAVE SIGNED DOUBLE PRODUCT OF TWO SINGLE NUMBERS *)
2       OVER OVER XOR >R ABS SWAP ABS U* R> D+- ;
3 : M/      ( FROM SIGNED DOUBLE-3-2, SIGNED DIVISOR-1 *)
4       ( LEAVE SIGNED REMAINDER-2, SIGNED QUOTIENT-1 *)
5       OVER >R >R DABS R ABS U/
6       R> R XOR +- SWAP R> +- SWAP ;
7 : *      U* DROP ;                           ( SIGNED PRODUCT *)
8 : /MOD   >R S->D R> M/ ;                 ( LEAVE REM-2, QUOT-1 *)
9 : /      /MOD SWAP DROP ;                  ( LEAVE QUOTIENT *)
10 : MOD   /MOD DROP ;                      ( LEAVE REMAINDER *)
11 : */MOD  ( TAKE RATION OF THREE NUMBERS, LEAVING *)
12       >R M* R> M/ ;                      ( REM-2, QUOTIENT-1 *)
13 : */    */MOD SWAP DROP ;                ( LEAVE RATIO OF THREE NUMBS *)
14 : M/MOD  ( DOUBLE, SINGLE DIVISOR ... REMAINDER, DOUBLE *)
15       >R O R U/ R> SWAP >R U/ R> ; -->

```

SCR # 58

```

0 ( DISC UTILITY, GENERAL USE               WFR-79APR02 )
1 FIRST VARIABLE USE           ( NEXT BUFFER TO USE, STALEST *)
2 FIRST VARIABLE PREV         ( MOST RECENTLY REFERENCED BUFFER *)
3
4 : +BUF      ( ADVANCE ADDRESS-1 TO NEXT BUFFER. RETURNS FALSE *)
5       84 ( I.E. B/BUF+4 ) + DUP LIMIT = ( IF AT PREV *)
6       IF DROP FIRST ENDIF DUP PREV @ - ;
7
8 : UPDATE     ( MARK THE BUFFER POINTED TO BY PREV AS ALTERED *)
9       PREV @ @ 8000 OR PREV @ ! ;
10
11 : EMPTY-BUFFERS ( CLEAR BLOCK BUFFERS; DON'T WRITE TO DISC *)
12       FIRST LIMIT OVER - ERASE ;
13
14 : DRO      0 OFFSET ! ;                   ( SELECT DRIVE #0 *)
15 : DR1      07DO OFFSET ! ; -->          ( SELECT DRIVE #1 *)

```

SCR # 59

```

0 ( BUFFER                                WFR-79APR02 )
1 : BUFFER      ( CONVERT BLOCK# TO STORAGE ADDRESS *)
2       USE @ DUP >R ( BUFFER ADDRESS TO BE ASSIGNED )
3       BEGIN +BUF UNTIL ( AVOID PREV ) USE ! ( FOR NEXT TIME )
4       R @ 0< ( TEST FOR UPDATE IN THIS BUFFER )
5       IF ( UPDATED, FLUSH TO DISC )
6       R 2+ ( STORAGE LOC. )
7       R @ 7FFF AND ( ITS BLOCK # )
8       0 R/W      ( WRITE SECTOR TO DISC )
9       ENDIF
10      R ! ( WRITE NEW BLOCK # INTO THIS BUFFER )
11      R PREV ! ( ASSIGN THIS BUFFER AS 'PREV' )
12      R> 2+ ( MOVE TO STORAGE LOCATION ) ;
13
14 -->
15

```

SCR # 60

```

0 ( BLOCK                               WFR-79APR02 )
1 : BLOCK      ( CONVERT BLOCK NUMBER TO ITS BUFFER ADDRESS *)
2   OFFSET @ + >R  ( RETAIN BLOCK # ON RETURN STACK )
3   PREV @ DUP @ R - DUP + ( BLOCK = PREV ? )
4   IF ( NOT PREV )
5     BEGIN +BUF 0= ( TRUE UPON REACHING 'PREV' )
6     IF ( WRAPPED ) DROP R BUFFER
7       DUP R 1      R/W    ( READ SECTOR FROM DISC )
8       2 - ( BACKUP )
9     ENDIF
10    DUP @ R - DUP + 0=
11    UNTIL ( WITH BUFFER ADDRESS )
12    DUP PREV !
13  ENDIF
14  R> DROP 2+ ;
15 -->

```

SCR # 61

```

0 ( TEXT OUTPUT FORMATTING           WFR-79MAY03 )
1
2 : (LINE)      ( LINE#, SCR#, ... BUFFER ADDRESS, 64 COUNT *)
3   >R C/L B/BUF */MOD R> B/SCR * +
4   BLOCK + C/L ;
5
6 : .LINE ( LINE#, SCR#, ... PRINTED *)
7   (LINE) -TRAILING TYPE ;
8
9 : MESSAGE      ( PRINT LINE RELATIVE TO SCREEN #4 OF DRIVE 0 *)
10  WARNING @
11  IF ( DISC IS AVAILABLE )
12    -DUP IF 4 OFFSET @ B/SCR / - .LINE ENDIF
13  ELSE ." MSG # " . ENDIF ;
14 -->
15

```

SCR # 62

```

0 ( LOAD, -->                           WFR-79APR02 )
1
2 : LOAD      ( INTERPRET SCREENS FROM DISC *)
3   BLK @ >R IN @ >R O IN ! B/SCR * BLK !
4   INTERPRET R> IN ! R> BLK ! ;
5
6 : -->      ( CONTINUE INTERPRETATION ON NEXT SCREEN *)
7   ?LOADING O IN ! B/SCR BLK @ OVER
8   MOD - BLK +! ; IMMEDIATE
9
10 -->
11
12
13
14
15

```

SCR # 63

0 ( INSTALLATION DEPENDENT TERMINAL I-O, TIM WFR-79APR26 )  
1 ( EMIT ) ASSEMBLER  
2 HERE -2 BYTE.IN EMIT ! ( VECTOR EMITS' CF TO HERE )  
3 XSAVE STX, BOT LDA, 7F # AND, 72C6 JSR, XSAVE LDX,  
4 CLC, 1A # LDY, UP )Y LDA, 01 # ADC, UP )Y STA,  
5 INY, UP )Y LDA, 00 # ADC, UP )Y STA, POP JMP,  
6 ( AND INCREMENT 'OUT' )  
7 ( KEY )  
8 HERE -2 BYTE.IN KEY ! ( VECTOR KEYS' CF TO HERE )  
9 XSAVE STX, BEGIN, 8 # LDX,  
10 BEGIN, 6E02 LDA, .A LSR, CS END, 7320 JSR,  
11 BEGIN, 731D JSR, 0 X) CMP, 0 X) CMP, 0 X) CMP,  
12 0 X) CMP, 0 X) CMP, 6E02 LDA, .A LSR, PHP, TYA,  
13 .A LSR, PLP, CS IF, 80 # ORA, THEN, TAY, DEX,  
14 0= END, 731D JSR, FF # EOR, 7F # AND, 0= NOT END,  
15 XSAVE LDX, PUSHOA JMP, -->

SCR # 64

0 ( INSTALLATION DEPENDENT TERMINAL I-O, TIM WFR-79APR02 )  
1  
2 ( ?TERMINAL )  
3 HERE -2 BYTE.IN ?TERMINAL ! ( VECTOR LIKEWISE )  
4 1 # LDA, 6E02 BIT, 0= NOT IF,  
5 BEGIN, 731D JSR, 6E02 BIT, 0= END, INY, THEN,  
6 TYA, PUSHOA JMP,  
7  
8 ( CR )  
9 HERE -2 BYTE.IN CR ! ( VECTOR CRS' CF TO HERE )  
10 XSAVE STX, 728A JSR, XSAVE LDX, NEXT JMP,  
11  
12 -->  
13  
14  
15

SCR # 65

0 ( INSTALLATION DEPENDENT DISC WFR-79APR02 )  
1 6900 CONSTANT DATA ( CONTROLLER PORT \*)  
2 6901 CONSTANT STATUS ( CONTROLLER PORT \*)  
3  
4  
5 : #HL ( CONVERT DECIMAL DIGIT FOR DISC CONTROLLER \*)  
6 0 OA U/ SWAP 30 + HOLD ;  
7  
8 -->  
9  
10  
11  
12  
13  
14  
15

SCR # 66

0 ( D/CHAR, ?DISC, WFR-79MAR23 )  
1 CODE D/CHAR ( TEST CHAR-1. EXIT TEST BOOL-2, NEW CHAR-1 \*)  
2 DEX, DEX, BOT 1+ STY, CO # LDA,  
3 BEGIN, STATUS BIT, 0= NOT END, ( TILL CONTROL READY )  
4 DATA LDA, BOT STA, ( SAVE CHAR )  
5 SEC CMP, 0= IF, INY, THEN, SEC STY, NEXT JMP,  
6  
7 : ?DISC ( UPON NAK SHOW ERR MSG, QUIT. ABSORBS TILL \*)  
8 1 D/CHAR >R 0= ( EOT, EXCEPT FOR SOH \*)  
9 IF ( NOT SOH ) R 15 =  
10 IF ( NAK ) CR  
11 BEGIN 4 D/CHAR EMIT  
12 UNTIL ( PRINT ERR MSG TIL EOT ) QUIT  
13 ENDIF ( FOR ENQ, ACK )  
14 BEGIN 4 D/CHAR DROP UNTIL ( AT EOT )  
15 ENDIF R> DROP ; -->

SCR # 67

0 ( BLOCK-WRITE WFR-790103 )  
1 CODE BLOCK-WRITE ( SEND TO DISC FROM ADDRESS-2, COUNT-1 \*)  
2 2 # LDA, SETUP JSR, ( WITH EOT AT END \*)  
3 BEGIN, 02 # LDA,  
4 BEGIN, STATUS BIT, 0= END, ( TILL IDLE )  
5 N CPY, 0=  
6 IF, ( DONE ) 04 # LDA, STATUS STA, DATA STA,  
7 NEXT JMP,  
8 THEN,  
9 N 2+ )Y LDA, DATA STA, INY,  
10 0= END, ( FORCED TO BEGIN )  
11  
12 -->  
13  
14  
15

SCR # 68

0 ( BLOCK-READ, WFR-790103 )  
1  
2 CODE BLOCK-READ ( BUF.ADDR-1. EXIT AT 128 CHAR OR CONTROL \*)  
3 1 # LDA, SETUP JSR,  
4 BEGIN, CO # LDA,  
5 BEGIN, STATUS BIT, 0= NOT END, ( TILL FLAG )  
6 50 ( BVC, D6=DATA )  
7 IF, DATA LDA, N )Y STA, INY, SWAP  
8 0< END, ( LOOP TILL 128 BYTES )  
9 THEN, ( OR D6=0, SO D7=1, )  
10 NEXT JMP,  
11  
12 -->  
13  
14  
15

SCR # 69

```

0 ( R/W FOR PERSCI 1070 CONTROLLER WFR-79MAY03 )
1 0A ALLOT HERE ( WORKSPACE TO PREPARE DISC CONTROL TEXT )
2 ( IN FORM: C TT SS /D, TT=TRACK, SS=SECTOR, D=DRIVE )
3 ( C = I TO READ, O TO WRITE *)
4 : R/W ( READ/WRITE DISC BLOCK *)
5 ( BUFFER ADDRESS-3, BLOCK #-2, I=READ O=WRITE *)
6 LITERAL HLD ! ( JUST AFTER WORKSPACE ) SWAP
7 0 OVER > OVER OF9F > OR 6 ?ERROR
8 07DO ( 2000 SECT/DR ) /MOD #HL DROP 2F HOLD BL HOLD
9 1A /MOD SWAP 1+ #HL #HL DROP BL HOLD ( SECTOR 01-26 )
10 #HL #HL DROP BL HOLD ( TRACK 00-76 )
11 DUP
12 IF 49 ( I=READ ) ELSE 4F ( O=WRITE ) ENDIF
13 HOLD HLD @ OA BLOCK-WRITE ( SEND TEXT ) ?DISC
14 IF BLOCK-READ ELSE B/BUF BLOCK-WRITE ENDIF
15 ?DISC ; -->

```

SCR # 70

0 ( FORWARD REFERENCES		WFR-79MAR30 )
1 00 BYTE.IN :	REPLACED.BY	?EXEC
2 02 BYTE.IN :	REPLACED.BY	!CSP
3 04 BYTE.IN :	REPLACED.BY	CURRENT
4 08 BYTE.IN :	REPLACED.BY	CONTEXT
5 0C BYTE.IN :	REPLACED.BY	CREATE
6 0E BYTE.IN :	REPLACED.BY	]
7 10 BYTE.IN :	REPLACED.BY	(;CODE)
8 00 BYTE.IN ;	REPLACED.BY	?CSP
9 02 BYTE.IN ;	REPLACED.BY	COMPILE
10 06 BYTE.IN ;	REPLACED.BY	SMUDGE
11 08 BYTE.IN ;	REPLACED.BY	[
12 00 BYTE.IN CONSTANT	REPLACED.BY	CREATE
13 02 BYTE.IN CONSTANT	REPLACED.BY	SMUDGE
14 04 BYTE.IN CONSTANT	REPLACED.BY	,
15 06 BYTE.IN CONSTANT	REPLACED.BY	(;CODE) -->

SCR # 71

0 ( FORWARD REFERENCES		WFR-79APR29 )
1 02 BYTE.IN VARIABLE	REPLACED.BY	(;CODE)
2 02 BYTE.IN USER	REPLACED.BY	(;CODE)
3 06 BYTE.IN ?ERROR	REPLACED.BY	ERROR
4 0F BYTE.IN ."	REPLACED.BY	WORD
5 1D BYTE.IN ."	REPLACED.BY	WORD
6 00 BYTE.IN (ABORT)	REPLACED.BY	ABORT
7 19 BYTE.IN ERROR	REPLACED.BY	MESSAGE
8 25 BYTE.IN ERROR	REPLACED.BY	QUIT
9 0C BYTE.IN WORD	REPLACED.BY	BLOCK
10 1E BYTE.IN CREATE	REPLACED.BY	MESSAGE
11 2C BYTE.IN CREATE	REPLACED.BY	MIN
12 04 BYTE.IN ABORT	REPLACED.BY	DRO
13 2C BYTE.IN BUFFER	REPLACED.BY	R/W
14 30 BYTE.IN BLOCK	REPLACED.BY	R/W DECIMAL ;S

SCR # 72

```

0 ( ', FORGET, \
1 HEX   3 WIDTH !
2 :           ( FIND NEXT WORDS PFA; COMPILE IT, IF COMPILING *)
3 -FIND 0= 0 ?ERROR DROP [COMPILE] LITERAL ;
4                                     IMMEDIATE
5
6 : FORGET          ( FOLLOWING WORD FROM CURRENT VOCABULARY *)
7   CURRENT @ CONTEXT @ - 18 ?ERROR
8   [COMPILE] DUP FENCE @ < 15 ?ERROR
9   DUP NFA DP ! LFA @ CURRENT @ ! ;
10
11
12
13 -->
14
15

```

SCR # 73

```

0 ( CONDITIONAL COMPILER, PER SHIRA                               WFR-79APR01 )
1 : BACK HERE - , ;      ( RESOLVE BACKWARD BRANCH *)
2
3 : BEGIN ?COMP HERE 1 ;           IMMEDIATE
4
5 : ENDIF ?COMP 2 ?PAIRS HERE OVER - SWAP ! ; IMMEDIATE
6
7 : THEN [COMPILE] ENDIF ; IMMEDIATE
8
9 : DO     COMPILE (DO) HERE 3 ; IMMEDIATE
10
11 : LOOP 3 ?PAIRS COMPILE (LOOP) BACK ; IMMEDIATE
12
13 : +LOOP 3 ?PAIRS COMPILE (+LOOP) BACK ; IMMEDIATE
14
15 : UNTIL 1 ?PAIRS COMPILE OBRANCH BACK ; IMMEDIATE -->

```

SCR # 74

```

0 ( CONDITIONAL COMPILER                                         WFR-79APR01 )
1 : END [COMPILE] UNTIL ; IMMEDIATE
2
3 : AGAIN 1 ?PAIRS COMPILE BRANCH BACK ; IMMEDIATE
4
5 : REPEAT >R >R [COMPILE] AGAIN
6       R> R> 2 - [COMPILE] ENDIF ; IMMEDIATE
7
8 : IF     COMPILE OBRANCH HERE 0 , 2 ; IMMEDIATE
9
10 : ELSE    2 ?PAIRS COMPILE BRANCH HERE 0 ,
11       SWAP 2 [COMPILE] ENDIF 2 ; IMMEDIATE
12
13 : WHILE  [COMPILE] IF 2+ ; IMMEDIATE
14
15 -->

```

SCR # 75

```

0 ( NUMERIC PRIMITIVES                               WFR-79APR01 )
1 : SPACES      0 MAX -DUP IF 0 DO SPACE LOOP ENDIF ;
2
3 : <#        PAD HLD ! ;
4
5 : #>        DROP DROP HLD @ PAD OVER - ;
6
7 : SIGN      ROT 0< IF 2D HOLD ENDIF ;
8
9 : #          ( CONVERT ONE DIGIT, HOLDING IN PAD * )
10           BASE @ M/MOD ROT 9 OVER < IF 7 + ENDIF 30 + HOLD ;
11
12 : #S        BEGIN # OVER OVER OR 0= UNTIL ;
13 -->
14
15

```

SCR # 76

```

0 ( OUTPUT OPERATORS                               WFR-79APR20 )
1 : D.R          ( DOUBLE INTEGER OUTPUT, RIGHT ALIGNED IN FIELD * )
2       >R SWAP OVER DABS <# #S SIGN #
3       R> OVER - SPACES TYPE ;
4
5 : D.          0 D.R SPACE ;                   ( DOUBLE INTEGER OUTPUT * )
6
7 : .R          >R S->D R> D.R ;           ( ALIGNED SINGLE INTEGER * )
8
9 : .          S->D D. ;                   ( SINGLE INTEGER OUTPUT * )
10
11 : ?          @ . ;                      ( PRINT CONTENTS OF MEMORY * )
12
13 : . CFA      MESSAGE 2A + ! ( PRINT MESSAGE NUMBER )
14 -->
15

```

SCR # 77

```

0 ( PROGRAM DOCUMENTATION                         WFR-79APR20 )
1 HEX
2 : LIST          ( LIST SCREEN BY NUMBER ON STACK * )
3           DECIMAL CR DUP SCR !
4           ." SCR #" . 10 0 DO CR I 3 .R SPACE
5           I SCR @ .LINE LOOP CR ;
6
7 : INDEX         ( PRINT FIRST LINE OF EACH SCREEN FROM-2, TO-1 * )
8           OC EMIT ( FORM FEED ) CR 1+ SWAP
9           DO CR I 3 .R SPACE
10          O I .LINE
11          ?TERMINAL IF LEAVE ENDIF LOOP ;
12 : TRIAD        ( PRINT 3 SCREENS ON PAGE, CONTAINING # ON STACK * )
13           OC EMIT ( FF ) 3 / 3 * 3 OVER + SWAP
14           DO CR I LIST LOOP CR
15           OF MESSAGE CR ;     DECIMAL -->

```

SCR # 78  
0 ( TOOLS  
1 HEX

WFR-79APR20 )

2 : VLIST

( LIST CONTEXT VOCABULARY \*)

3 80 OUT ! CONTEXT @ @  
4 BEGIN OUT @ C/L > IF CR 0 OUT ! ENDIF  
5 DUP ID. SPACE SPACE PFA LFA @  
6 DUP 0= ?TERMINAL OR UNTIL DROP ;  
7 -->

8

9

10

11

12

13

14

15

SCR # 79

0 ( TOOLS  
1 HEX

WFR-79MAY03 )

2  
3 CREATE MON  
4 0 C, 4C C, LIT 18 + , SMUDGE

5

6

7

8

9

10 DECIMAL

11 HERE FENCE !

12 HERE 28 +ORIGIN ! ( COLD START FENCE )

13 HERE 30 +ORIGIN ! ( COLD START DP )

14 LATEST 12 +ORIGIN ! ( TOPMOST WORD )

15 FORTH 6 + 32 +ORIGIN ! ( COLD VOC-LINK ) ;S

SCR # 80

0 -->

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

FORTH INTEREST GROUP

MAY 1, 1979

This is a sample editor, compatible with the fig-FORTH model and simple terminal devices. The line and screen editing functions are portable. The code definition for the string MATCH could be written high level or translated.

SCR # 87

0 ( TEXT, LINE WFR-79MAY01 )  
1 FORTH DEFINITIONS HEX  
2 : TEXT ( ACCEPT FOLLOWING TEXT TO PAD \*)  
3 HERE C/L 1+ BLANKS WORD HERE PAD C/L 1+ CMOVE ;  
4  
5 : LINE ( RELATIVE TO SCR, LEAVE ADDRESS OF LINE \*)  
6 DUP FFF0 AND 17 ?ERROR ( KEEP ON THIS SCREEN )  
7 SCR @ (LINE) DROP ;  
8 -->  
9  
10  
11  
12  
13  
14  
15

SCR # 88

0 ( LINE EDITOR WFR-79MAY03 )  
1 VOCABULARY EDITOR IMMEDIATE HEX  
2 : WHERE ( PRINT SCREEN # AND IMAGE OF ERROR \*)  
3 DUP B/SCR / DUP SCR ! ." SCR # " DECIMAL .  
4 SWAP C/L /MOD C/L \* ROT BLOCK + CR C/L TYPE  
5 CR HERE C@ - SPACES 5E EMIT [COMPILE] EDITOR QUIT ;  
6  
7 EDITOR DEFINITIONS  
8 : #LOCATE ( LEAVE CURSOR OFFSET-2, LINE-1 \*)  
9 R# @ C/L /MOD ;  
10 : #LEAD ( LINE ADDRESS-2, OFFSET-1 TO CURSOR \*)  
11 #LOCATE LINE SWAP ;  
12 : #LAG ( CURSOR ADDRESS-2, COUNT-1 AFTER CURSOR \*)  
13 #LEAD DUP >R + C/L R> - ;  
14 : -MOVE ( MOVE IN BLOCK BUFFER ADDR FROM-2, LINE TO-1 \*)  
15 LINE C/L CMOVE UPDATE ; -->

SCR # 89

0 ( LINE EDITING COMMANDS WFR-79MAY03 )  
1 : H ( HOLD NUMBERED LINE AT PAD \*)  
2 LINE PAD 1+ C/L DUP PAD C! CMOVE ;  
3  
4 : E ( ERASE LINE-1 WITH BLANKS \*)  
5 LINE C/L BLANKS UPDATE ;  
6  
7 : S ( SPREAD MAKING LINE # BLANK \*)  
8 DUP I - ( LIMIT ) OE ( FIRST TO MOVE )  
9 DO I LINE I 1+ -MOVE -1 +LOOP E ;  
10  
11 : D ( DELETE LINE-1, BUT HOLD IN PAD \*)  
12 DUP H OF DUP ROT  
13 DO I 1+ LINE I -MOVE LOOP E ;  
14  
15 -->

SCR # 90  
0 ( LINE EDITING COMMANDS WFR-79MAY03 )

1  
2 : M ( MOVE CURSOR BY SIGNED AMOUNT-1, PRINT ITS LINE \*)  
3 R# +! CR SPACE #LEAD TYPE 5F EMIT  
4 #LAG TYPE #LOCATE . DROP ;  
5  
6 : T ( TYPE LINE BY #-1, SAVE ALSO IN PAD \*)  
7 DUP C/L \* R# ! DUP H O M ;  
8  
9 : L ( RE-LIST SCREEN \*)  
10 SCR @ LIST 0 M ;  
11 -->  
12  
13  
14  
15

SCR # 91  
0 ( LINE EDITING COMMANDS WFR-790105 )  
1 : R ( REPLACE ON LINE #-1, FROM PAD \*)  
2 PAD 1+ SWAP -MOVE ;  
3  
4 : P ( PUT FOLLOWING TEXT ON LINE-1 \*)  
5 1 TEXT R ;  
6  
7 : I ( INSERT TEXT FROM PAD ONTO LINE # \*)  
8 DUP S R ;  
9 CR  
10 : TOP ( HOME CURSOR TO TOP LEFT OF SCREEN \*)  
11 0 R# ! ;  
12 -->  
13  
14  
15

SCR # 92  
0 ( SCREEN EDITING COMMANDS WFR-79APR27 )  
1 : CLEAR ( CLEAR SCREEN BY NUMBER-1 \*)  
2 SCR ! 10 0 DO FORTH I EDITOR E LOOP ;  
3  
4 : FLUSH ( WRITE ALL UPDATED BLOCKS TO DISC \*)  
5 [ LIMIT FIRST - B/BUF 4 + / ] ( NUMBER OF BUFFERS)  
6 LITERAL 0 DO 7FFF BUFFER DROP LOOP ;  
7  
8 : COPY ( DUPLICATE SCREEN-2, ONTO SCREEN-1 \*)  
9 B/SCR \* OFFSET @ + SWAP B/SCR \* B/SCR OVER + SWAP  
10 DO DUP FORTH I BLOCK 2 - ! 1+ UPDATE LOOP  
11 DROP FLUSH ;  
12 -->  
13  
14  
15

SCR # 93

```

0 ( STRING EDITING PRIMITIVES                               WFR-79APR22 )
1 CODE MATCH ( CURSOR ADDRESS-4, BYTES LEFT-3, STRING ADDR-2 *)
2   ( ITS COUNT-1. LEAVE BOOLEAN-2, CURSOR ADVANCEMENT-1 *)
3   4 # LDA, SETUP JSR, DEX, DEX, DEX, DEX,
4     BOT STY, BOT 1+ STY,
5     BEGIN, ( NEW MATCH ) DROP ( ERR ) FF # LDY,
6     BEGIN, DROP ( ERR ) INY, N CPY, CS NOT
7       IF, ( Y < STRING ) N 2+ )Y LDA, N 6 + )Y CMP,
8 ROT 1     0= NOT UNTIL, ( REPEAT FOR GOOD MATCH )
9       N 6 + INC, 0= IF, N 7 + INC, ENDIF,
10      BOT INC, 0= IF, BOT 1+ INC, ENDIF, ( CUR MOT )
11      N 4 + LDA, 0= IF, N 5 + DEC, ENDIF,
12        N 4 + DEC, ( DECREMENT BUFFER REMAINING )
13        N 4 + LDA, N CMP, ( REMAINING - STRING SIZE )
14        N 5 + LDA, N 1+ SBC,
15 ROT 1     CS NOT UNTIL, --> ( REPT TILL OUT OF BUFFER )

```

SCR # 94

```

0 ( CONCLUSION OF STRING MATCH                           WFR-79APR22 )
1   0 # LDA, SEC STA, SEC 1+ STA, ( BOOLEAN FALSE )
2     N 4 + LDY, ( SPACE UNTIL END OF BUFFER )
3     ENDIF,
4     CLC, TYA, BOT ADC, PHA,
5     0 # LDA, BOT 1+ ADC, ( ADJUST CURSOR MOTION )
6     PUT JMP, C;
7 -->
8
9
10
11
12
13
14
15

```

SCR # 95

```

0 ( STRING EDITING COMMANDS                            WFR-79MAR24 )
1 : ILINE      ( SCAN LINE WITH CURSOR FOR MATCH TO PAD TEXT, *)
2                                ( UPDATE CURSOR, RETURN BOOLEAN *)
3     #LAG PAD COUNT MATCH R# +! ;
4
5 : FIND      ( STRING AT PAD OVER FULL SCREEN RANGE, ELSE ERROR *)
6     BEGIN 3FF R# @ <
7       IF TOP PAD HERE C/L 1+ CMOVE 0 ERROR ENDIF
8     ILINE UNTIL ;
9
10 : DELETE     ( BACKWARDS AT CURSOR BY COUNT-1 *)
11     >R #LAG + FORTH R - ( SAVE BLANK FILL LOCATION )
12     #LAG R MINUS R# +! ( BACKUP CURSOR )
13     #LEAD + SWAP CMOVE
14     R> BLANKS UPDATE ; ( FILL FROM END OF TEXT )
15 -->

```

SCR # 96

0 ( STRING EDITOR COMMANDS  
1 : N ( FIND NEXT OCCURANCE OF PREVIOUS TEXT \*)  
2 FIND O M ;  
3  
4 : F ( FIND OCCURANCE OF FOLLOWING TEXT \*)  
5 1 TEXT N ;  
6  
7 : B ( BACKUP CURSOR BY TEXT IN PAD \*)  
8 PAD C@ MINUS M ;  
9  
10 : X ( DELETE FOLLOWING TEXT \*)  
11 1 TEXT FIND PAD C@ DELETE O M ;  
12  
13 : TILL ( DELETE ON CURSOR LINE, FROM CURSOR TO TEXT END \*)  
14 #LEAD + 1 TEXT ILINE 0= 0 ?ERROR  
15 #LEAD + SWAP - DELETE O M ; -->

WFR-79MAR24 )

SCR # 97

0 ( STRING EDITOR COMMANDS  
1 : C ( SPREAD AT CURSOR AND COPY IN THE FOLLOWING TEXT \*)  
2 1 TEXT PAD COUNT  
3 #LAG ROT OVER MIN >R  
4 FORTH R R# +! ( BUMP CURSOR )  
5 R - >R ( CHARS TO SAVE )  
6 DUP HERE R CMOVE ( FROM OLD CURSOR TO HERE )  
7 HERE #LEAD + R> CMOVE ( HERE TO CURSOR LOCATION )  
8 R> CMOVE UPDATE ( PAD TO OLD CURSOR )  
9 O M ( LOOK AT NEW LINE ) ;  
10 FORTH DEFINITIONS DECIMAL  
11 LATEST 12 +ORIGIN ! ( TOP NFA )  
12 HERE 28 +ORIGIN ! ( FENCE )  
13 HERE 30 +ORIGIN ! ( DP )  
14 EDITOR 6 + 32 +ORIGIN ! ( VOC-LINK )  
15 HERE FENCE ! ;S

WFR-79MAR23 )

SCR # 98

0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15