fig-FORTH under OS-65U
Documentation
(PRELIMINARY)

Software Consultants
7053 Rose Trail
Memphis, TN 38134
(901) 377-3503

INTRODUCTION.

The Software Consultants fig-FORTH under OS-65U is a (mostly) faithfull implementation of the fig model as defined in the fig-FORTH Installation Manual & Glossary. The areas where we have deviated from the model are defined later in this document.

We assume that anyone who has purchased this package has at least some familiarity with the FORTH language, so we won't give a detailed explanation of the workings of the language. However, we would like to make a few statements to those who are not yet polished FORTH programmers.

Even more than some other languages, FORTH has a definate "learning curve" before you will feel confident in your programming. This learning curve will differ greatly among individuals, mostly depending upon the type of programming previously done.

Typically, assembler programmers will take to FORTH almost immediately, while those who's background is strictly BASIC require a longer period of time before they begin to "think" in FORTH. Regardless of your previous background, if you will take the time to become familiar with FORTH, we are sure you will agree that FORTH is a near perfect tool for writing most any type of application you can devise.

We strongly recommend that you join the Forth Interest Group, P.O. Box 1105, San Carlos, CA 94070. Current membership is $12.00/year in the US. Membership includes a subscription to Forth Dimensions. This magazine is currently the best avaiable forum for information concerning FORTH. Also available from fig are a number of manuals which can be of great help to the beginning FORTH programmer.

DEVIATIONS FROM THE MODEL.

This version of FORTH was designed with business applications in mind. Toward this end, several modifications of the original FORTH system have been made as defined below.

1. The words DR0, DR1 and OFFSET have been deleted. Since this version was designed mainly for hard disk based systems, they are not necessary.

2. The use of lower case in definitions is now allowed. As well as the additional flexability in naming, compile time is also increased by about ten percent.

3. An auto-load feature has been added which automatically loads screen three upon boot or when the words COLD or ABORT are executed. This was done to allow "turn-key" systems with minimal operator training.

4. A "quiet compile" feature has been added. If the variable QUIET is non-zero, the compiler will not report the "<NAME> ISN'T UNIQUE" message. This again is for turn-key systems.

5. The words LIST, INDEX, TRIAD, and ULIST have been removed from the source and written high-level. Normally they are loaded automatically, but may be eliminated from the auto-load screen to provide a "locked" system.

6. The word MON from the model has been replaced by the word BASIC, which will reload the BASIC interpreter and go to the console mode. CAUTION: Be sure and do a FLUSH before returning to BASIC to avoid losing any disk modifications.

7. All references to multiple sectors comprising a single FORTH screen have been removed since this is not required under OS-65U.

ENHANCEMENTS.

A number of useful routines have been included with the system. Each is described below. Please note that these routines were written by us and are NOT in the public domain.

TERMINAL & PRINTER TOOLS: In order to make dealing with the terminal and printer easier, a number of useful words have been defined on screens 6 thru 8. These words are well commented and should be self-explanatory.

LISTING WORDS: The words LIST, INDEX, TRIAD, and ULIST along with some supporting words are defined on screens 9 thru 11. Two types of modifications were made to the standard words. The first three words are made to operate differently depending on whether the output is to the terminal or printer. Also, the listing words have been modified to allow for object screens (defined below).

DOUBLE PRECISION SUPPORT: Screen 18 adds some words useful in dealing with double precision numbers.

DISK I/O SUPPORT: Screen 19 defines the word DISK which may be used to do direct disk access anywhere on the disk. This word may be used to read and write standard 65U disk files. For an example of it's use, see the OS-65U disk directory words.

Also on screen 19 is the constant TCB. This is the location of the transfer control block in 65U. You may change the current drive for disk access by storing the proper value at this address. I.E., to change to floppy drive B, do "1 TCB C!". All further disk access would then be from drive B.

CASE STATEMENT: Screen 20 contains a general case statement as found in such languages as Pascal. The syntax of the case statement is as follows:

```
DO-CASES   ( start case, number to test on top of stack )
   n CASE ... ESAC  ( if n = tos then do everything between
        .             CASE and ESAC, then continue execution
        .             after CASES-DONE.  This construct can
        .             be repeated any number of times. )
   default statements  ( any statements between the last CASE, ESAC
                         pair will be executed if none of the
                         previous cases was true. )
CASES-DONE   ( end of case statment )
```

DISK DIRECTORY: Screens 21 thru 23 defines words which emulate the BASIC program "DIR". Once loaded, typing DIR will display the 65U disk directory. PRN DIR will send the directory to the printer. Note: Screens 18 and 19 must be loaded before loading screen 21.

FIND FILE IN DIRECTORY: Screen 48 defines the word FINDFL which will search the OS-65U directory for a given file name. The word TEST is also defined as an example of how to use FINDFL.

Before calling FINDFL, the name of the file to be found must be at HERE in the normal fashion left by the word WORD, i.e., the first byte is the length followed by the file name. If the file name is not found, the only entry on the stack after calling FINDFL will be the boolean flag 0 to denote failure. If the name is found, a 1 will be the first entry on the stack, followed by a double precision file length, and a double precision disk address. These entries may be used with the word DISK defined on screen 19 to do disk I/O on a standard OS-65U data file.

FINDFL requires the definitions on screen 21. Therefore, either the entire disk directory must be loaded prior to loading screen 48, or the contents of screen 21 only may be loaded.

RANDOM NUMBER GENERATOR: Screen 33 contains a simple random number routine which allows specifying the lowest and highest allowed numbers.

PRINT STACK CONTENTS: Screen 34 defines the word S? which will non-destructively display the contents of the stack in both decimal and hex. This is a very useful word during the debugging process.

LOAD/UNLOAD ASSEMBLER: Screen 35 contains an alternative way to use the standard fig assembler (on screens 12 - 17). Normally the fig assembler is loaded, then any required code words are defined. The only problem with this is that the assembler occupies about 1300 bytes of code, which are not used once the code words are compiled.

Screen 35 will load the assembler in high memory. After all required code words are defined, executing the word KILL.ASSM will remove the assembler. Therefore, code words may be used without the 1300 byte overhead which is normally required.

Note: At this time, there is still a "bug" in the load/unload assembler which will create problems if another vocabulary has been defined after screen 35 is loaded. Until this is fixed, we strongly suggest that all words defined (both code and high-level) between the time screen 35 is loaded and KILL.ASSM is executed should go in the FORTH vocabulary.

GET/PUT COMPILED CODE: The normal method of executing a FORTH application is to load the required colon definitions from disk and then execute them. The only difficulty is that loading definitions requires compiling them. While the FORTH compiler is fast, it is certainly not instantaneous. Therefore, it can take many seconds to load even a relatively small application.

For development purposes, the speed of the LOAD is not a problem, but for turn-key systems, execution of an application should be as fast as possible.

The definitions on screens 36 - 38 allow storing compiled code on disk and then recalling them for use later at extreme high speed. This is done by using the word PUT to write the memory image of compiled words to disk, then using the word GET to recall them.

The word PUT is used as <screen #> PUT <name>, where screen # is the first screen the compiled code will occupy, and name is the name of the first word in the dictionary to be put to disk. All the following words up to HERE are saved. Please note that variables and constants can be put to disk just as colon defined words can.

The word GET is used as <screen #> GET, where screen # is the first screen containing previously put code. A major word of caution: The dictionary MUST be in exactly the same condition as when the corresponding PUT was executed. If it is not, you can expect the system to just go totally out to lunch.

This is not a limitation, but a requirement for doing the GET in the fashion we are. GET is not a relocating linking loader, but a simple disk to memory routine. While we could have written GET to allow loading code onto a modified dictionary, the extra processing required would probably have required as much time (or even more!) as the normal LOAD command.

The words LOAD, EDIT, LIST, and INDEX have been modified to work correctly with object code screens. Attempting to LOAD or EDIT an object screen will give an error. LISTing an object screen will show the starting and ending memory addresses and length of the PUT block.

TERMINAL ORIENTED EDITOR: We've saved the best for last. If you have ever used the standard fig editor, you know it leaves a great deal to be desired. Screens 24 thru 32 contains a very flexible, easy to use, FORTH screen editor.

The normal method for editing a screen is by the command <screen #> EDIT. This will clear the screen, list the screen in the upper portion of the display, and position the cursor at the beginning of line zero. A number of control functions are allowed to simplify entry and editing of screen data.

1) The cursor control keys; up, down, left, and right; may be used to put the cursor anywhere on the display within the screen. All four functions "wrap around", i.e., pressing the up arrow at the top line will move the cursor to the same column on the bottom line. Likewise, pressing the left arrow key at the beginning of a line will move the cursor to the end of the previous line.

2) The tab key will move the cursor to the right by the number of spaces contained in the variable TABAMT. Shift tab will move the cursor to the left by the same amount. TABAMT is initially loaded with eight, but may be modified to any value you desire. The tab function also will wrap around.

3) CTRL E will erase the line containing the cursor.

4) CTRL S will insert a blank line at the position of the cursor. Line 15 will be lost.

5) CTRL D will delete the line containing the cursor and move all following lines up. Line 15 will be blank.

6) CTRL A will insert a single blank at the current cursor position. All remaining characters on the line will be moved to the right., The last character on the line is lost.

7) The delete key will remove the character presently under the cursor. All remaining characters on the line will be moved to the left. The last character on the line will be a space.

8) CTRL P and CTRL O work in conjunction with each other. CTRL P will save the line containing the cursor at PAD. CTRL O will replace the line containing the cursor with the line previously saved with CTRL P. This allows moving a line from one location to another on a screen, or moving a line from one screen to another.

9) CTRL T will position the cursor to the beginning of line zero.

10) The escape key will end editing of the current screen. If any modifications were made, the screen will be marked as updated.

Also available within the EDITOR vocabulary are several other helpful words. The word N will edit the next screen. The word S will re-edit the last screen edited. The word P will edit the previous screen. <screen #> CLEAR will fill the given screen with blanks. <from screen> <to screen> COPY will copy the from screen onto the to screen.

If a compile-time error occurs, the word WHERE may be used to edit the screen containing the error and place the cursor at the location of the error.

INSTALLATION PROCEDURE.

As with any purchased software, the first thing you should do is copy this disk, and put the original away. The original disk should never be modified.

If you will be using our FORTH on a hard disk, you will need to create three files on the disk. The two files FORTH and BASIC should be created at the same size (or larger) as shown on the floppy. The file SCREEN may be created as large as you wish (max size is 64 Megabytes!). This is the file which contains the FORTH disk "work space". Each one K of disk space will give you one FORTH screen.

The file FORTH is a BASIC program which is run to initiate FORTH. It may be loaded from the floppy and saved to the hard disk. The file BASIC is a work file to hold the BASIC interpreter while FORTH is executing. It need not be copied. The file SCREEN may be moved using the OSI utility COPYFI.

If a larger SCREEN file is created, the new screens should be cleared before using them. Assume you create a one megabyte file for SCREEN (1024 screens). The following word should be defined and executed before attempting to use the new screens:

```
: CLEAN  EDITOR  1025  225  DO  I  CLEAR  LOOP  ;
```

If you will be using FORTH on a floppy based system, the following procedure should be used to prepare a working disk. Place an OS-65U disk in drive A and a blank disk in drive B. Using the COPIER utility, initialize the disk in drive B and copy the operating system onto it from drive A. Replace the disk in drive A with the delivered FORTH disk, and copy the files from drive A to B.

REQUIRED MODIFICATIONS.

The only modifications that will normally be required before you can take advantage of all the existing FORTH definitions are in the terminal and printer areas. All terminal oriented commands are delivered set up for a Micro-Term, Inc. ACT-5A. If you are using another type of terminal, certain changes must be made.

Since the terminal oriented editor will not function until these changes are made, a minimal line oriented editor is provided on screens 45 and 46. After loading this, a screen may be modifyed by typing EDITOR, then LISTing the required screen, and using the following commands:

L : The L command will relist the current screen and show any changes which have been made.
<line #> P <text> : The P command is the method used to put text on a screen. The entered text will replace the existing line's data. Note that at least one space must follow the text, but the first space will not be put on the screen.
<line #> S : This is the same as the CTRL S in the video editor
<line #> E : Same as the video's CTRL E.
<line #> D : Same as the video's CTRL D.
<line #> H : Hold the designated line at PAD.
<line #> R : Replace the line with the data held at PAD.
<line #> I : Insert the data at PAD at the line entered and move following lines down. Line 15 is lost.

The words CLEAR and COPY are also provided. They are the same as those under the video editor.

The only changes that should be required are on screens 6 thru 8 and 28 thru 30. If you are not using a serial printer (BASIC device #3), then the value of the constant PRNUAL on screen 6 should be changed. The proper value for a parallel printer would be 10. For a device #8 serial printer, the value would be 80. The 0C EMIT on line 4 of screen 7 should be changed to whatever is required to clear the screen on your terminal. Likewise, the 14 EMIT on line 6 of screen 8 should be set to the cursor positioning command. If your terminal requires the column address before the row address, then put a SWAP before the last two EMITs on this line.

The possible changes on screens 28 thru 30 are in the terminal oriented editor. In particular, the codes for the up, down, left, and right arrow keys may require changing. Note that the codes for each key is in decimal.

Once all changes have been made, executing "5 LOAD" will recompile all definitions loaded at boot and save the compiled code. Executing the word "COLD" will then make the changes effective.

RUNTIME MODIFICATIONS.

Quite often it is better to have several smaller FORTH disk work spaces rather than one large one. This can easily be done with this version of FORTH. The base disk address of the file used by FORTH (default is SCREEN) is contained at memory address zero. To change to another file for the work space, just store the proper disk address here.

The disk address is a double precision number, i.e., 4 bytes. If the double precision tools are loaded, the D! may be used to put the proper value at zero. If not, the disk address will need to be broken into two pieces and stored at zero (low order) and two (high order).

Changing from one floppy disk drive to another only requires storing the proper value at address 26A1 hex. If the disk I/O support tools are loaded, this value is stored in the constant TCB (transfer control block). This value is a single byte; 0 - 4 for drives A - D. It may be set with the C! word.

If either of the above changes are made, you should precede them with a FLUSH to write any updated buffers to disk, and an EMPTY-BUFFERS to prevent copying over any existing buffers to the new disk area.

Please note that this FORTH will accept screen numbers outside of the legal range for the file used as the disk work space. This was done to allow accessing other 65U disk files using the standard FORTH words, such as BUFFER and BLOCK.

IN CONCLUSION....

As with all our products, we at Software Consultants fully support our FORTH. If you have any questions or problems, do not hesitate to contact us. We would prefer that problems be communicated by mail, so that you may provide as much detail as possible in defining the problem, and we may be as explicit as possible with our answer. However, unlike some vendors (who shall remain nameless), we will gladly speak to you by phone if you have an immediate need.

As usual, the source code for FORTH is available on disk for our standard $10.00 nominal fee. Due to the large size of the source, it cannot conveniently be assembled with the standard OSI assembler. The source is delivered set up for assembly using the Pegasus Software A/65 assembler.

Unlike most interpreters or compilers, the FORTH source is mostly in FORTH itself. Modifications are normally much easier to make than it would be for a more common language. If you have any assembler experience, and feel that certain things could be done more to your liking, jump in! It really is quite a lot of fun!

While this FORTH is complete as delivered, and even includes a number of "extras", it is by no means the end of the line. We intend to offer several additions in the near future. String handling and floating point routines are currently working in house, and will be released shortly. Even more exciting developments are on the drawing board, so watch for our ads.

fig-FORTH

INSTALLATION MANUAL

GLOSSARY

MODEL

RELEASE 1

WITH COMPILER SECURITY

AND

VARIABLE LENGTH NAMES

BY
WILLIAM F. RAGSDALE

AUGUST 1980

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

fig-FORTH INSTALLATION MANUAL

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 commerical 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 defintions. 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 intepreter 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 preceeding. 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 reentrant 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 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 contol H type:

```
HEX  08  0E  +ORIGIN  !  ( byte addresses)
```

6.0 DOCUMENTATION SUMMARY

The following manuals are in print:

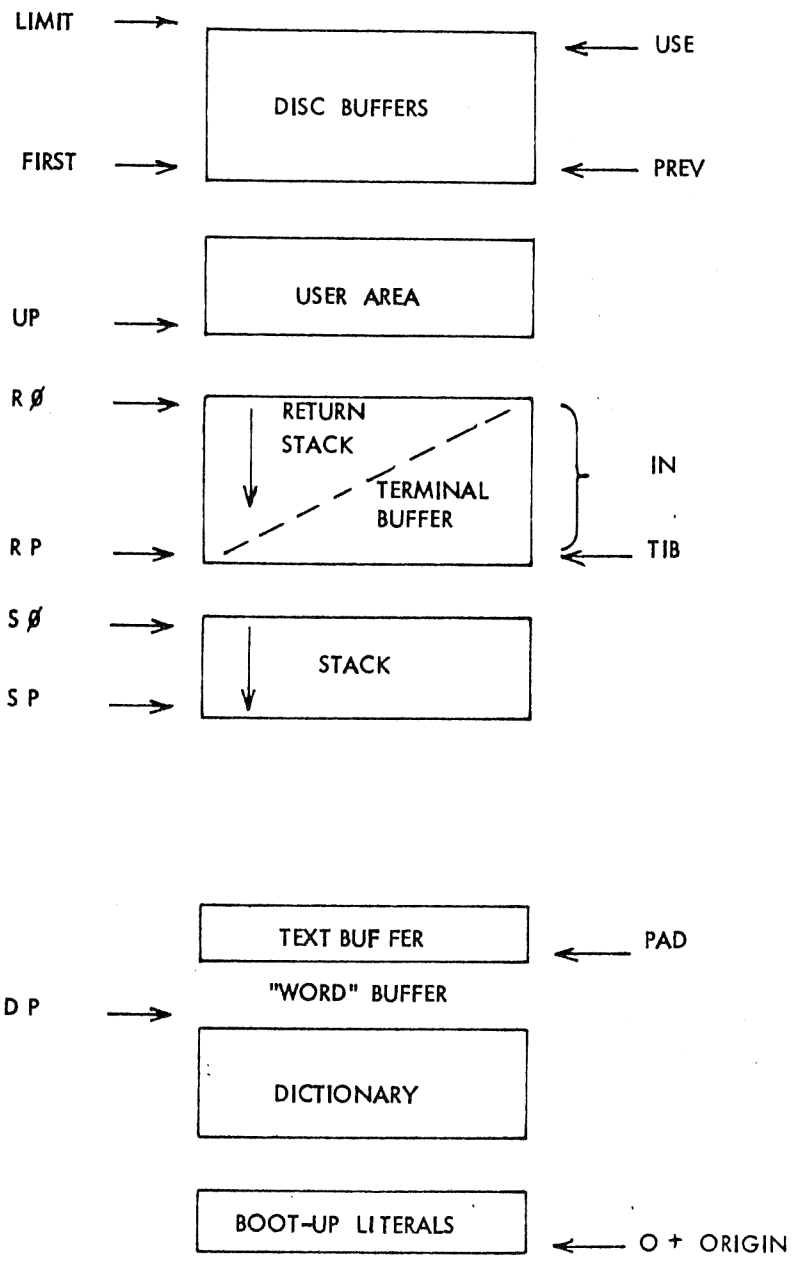
Caltech FORTH Manual, an advanced manual with internal details of Forth. Has some implementation peculiarities. Approx. $6.50 from the Caltech Book Store, Pasadena, CA.

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

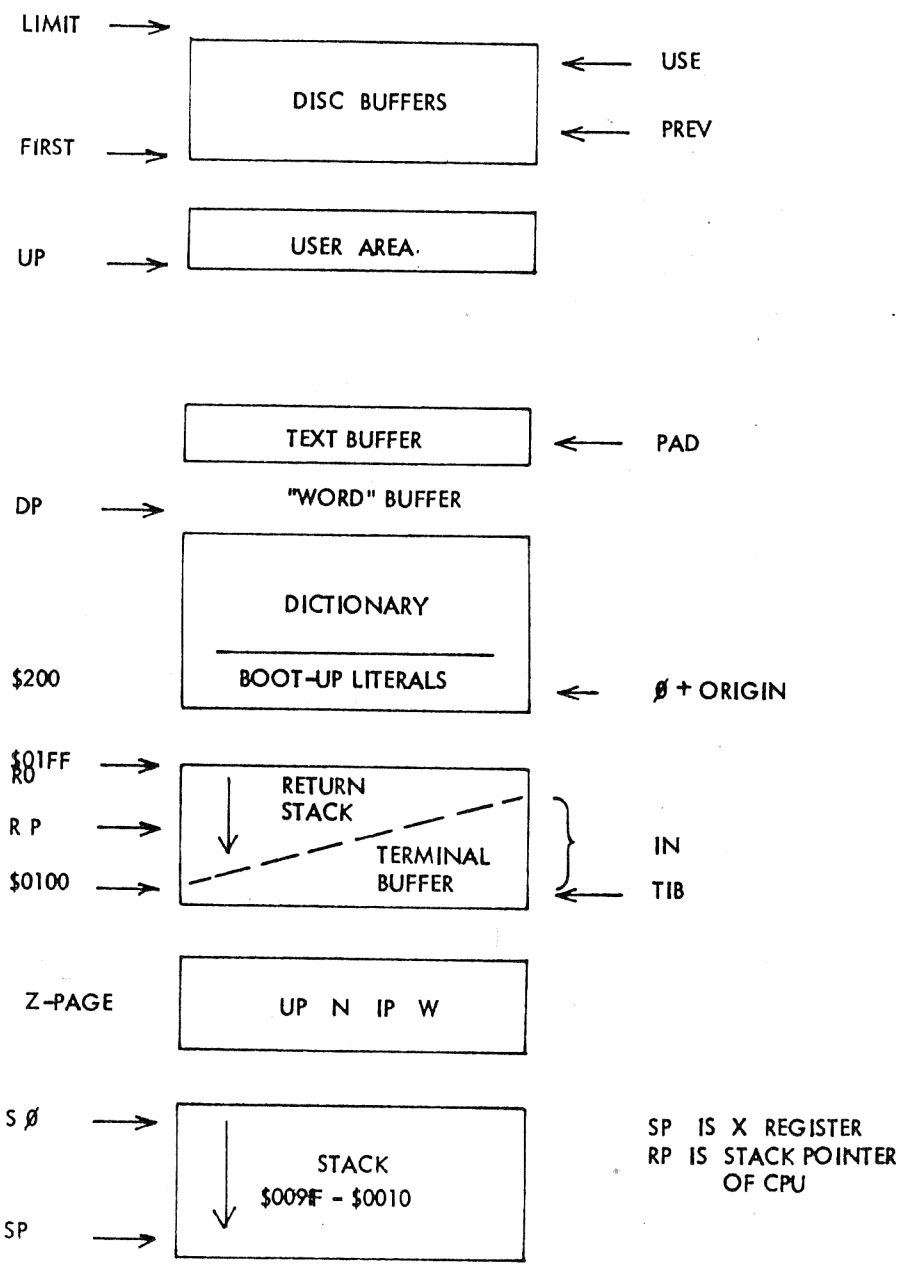
microFORTH Primer, $15.00 Forth, Inc. 815 Manhattan Ave. Manhattan Beach, CA 90266

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



6502
fig-FORTH MEMORY MAP



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

! n addr --- L0
Store 16 bits of n at address. Pronounced "store".

!CSP
Save the stack position in CSP. Used as part of the compiler security.

d1 --- d2 L0
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.

#> d --- addr count L0
Terminates numeric output conversion by dropping d, leaving the text address and character count suitable for TYPE.

#S d1 --- d2 L0
Generates ascii text in the text output buffer, by the use of #, until a zero double number n2 results. Used between <# and #>.

' --- addr P,L0
Used in the form:
' nnnn
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".

(P,L0
Used in the form:
(cccc)
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.

(.") C+
The run-time proceedure, compiled by ." which transmits the following in-line text to the selected output device. See ."

(;CODE) C
The run-time proceedure, compiled by ;CODE, that rewrites the code field of the most recently defined word to point to the following machine code sequence. See ;CODE.

(+LOOP) n --- C2
The run-time proceedure compiled by +LOOP, which increments the loop index by n and tests for loop completion. See +LOOP.

(ABORT)
Executes after an error when WARNING is -1. This word normally executes ABORT, but may be altered (with care) to a user's alternative proceedure.

(DO) C
The run-time proceedure compiled by DO which moves the loop control parameters to the return stack. See DO.

(FIND) addr1 addr2 --- pfa b tf (ok)
addr1 addr2 --- ff (bad)
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.

(LINE) n1 n2 --- addr count
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.

(LOOP) C2
The run-time proceedure compiled by LOOP which increments the loop index and tests for loop completion. See LOOP.

(NUMBER) d1 addr1 --- d2 addr2
Convert the ascii text beginning at addr1+1 with regard to BASE. The new value is accumulated into double number d1, being left as d2. Addr2 is the address of the first unconvertable digit. Used by NUMBER.

* n1 n2 --- prod L0
Leave the signed product of two signed numbers.

*/ n1 n2 n3 --- n4 L0
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:
n1 n2 * n3 /

*/MOD n1 n2 n3 --- n4 n5 L0
Leave the quotient n5 and remainder n4 of the operation n1*n2/n3
A 31 bit intermediate product is used as for */.

+ n1 n2 --- sum LO
Leave the sum of n1+n2.

+! n addr --- LO
Add n to the value at the address. Pronounced "plus-store".

+- n1 n2 --- n3
Apply the sign of n2 to n1, which is left as n3.

+BUF addl --- addr2 f
Advance the disc buffer address addr1 to the address of the next buffer addr2. Boolean f is false when addr2 is the buffer presently pointed to by variable PREV.

+LOOP n1 --- (run)
addr n2 --- (compile) P,C2,LO
Used in a colon-definition in the form:
DO ... n1 +LOOP
At run-time, +LOOP selectively controls branching back to the corresponding DO based on n1, the loop index and the loop limit. The signed increment n1 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 (n1>0), or until the new index is equal to or less than the limit (n1<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
Leave the memory address relative by n to the origin parameter area. n is the minimum address unit, either byte or word. This definition is used to access or modify the boot-up parameters at the origin area.

, n --- LO
Store n into the next available dictionary memory cell, advancing the dictionary pointer. (comma)

- n1 n2 --- diff LO
Leave the difference of n1-n2.

--> P,LO
Continue interpretation with the next disc screen. (pronounced next-screen).

-DUP n1 -- n1 (if zero)
n1 -- n1 n1 (non-zero) LO
Reproduce n1 only if it is non-zero. This is usually used to copy a value just before IF, to eliminate the need for an ELSE part to drop it.

-FIND --- pfa b tf (found)
--- ff (not found)
Accepts the next text word (delimited by blanks) in the input strean 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.

-TRAILING addr n1 --- addr n2
Adjusts the character count n1 of a text string beginning address to suppress the output of trailing blanks. i.e. the characters at addr+n1 to addr+n2 are blanks.

. n --- LO
Print a number from a signed 16 bit two's complement value, converted according to the numeric BASE. A trailing blanks follows. Pronounced "dot".

." P,LO
Used in the form:
." cccc"
Compiles an in-line string cccc (delimited by the trailing ") with an execution proceedure to transmit the text to the selected output device. If executed outside a definition, ." will immediately print the text until the final ". The maximum number of characters may be an installation dependent value. See (.").

.LINE line scr ---
Print on the terminal device, a line of text from the disc by its line and screen number. Trailing blanks are suppressed.

.R n1 n2 ---
Print the number n1 right aligned in a field whose width is n2. No following blank is printed.

/ n1 n2 --- quot LO
Leave the signed quotient of n1/n2.

/MOD n1 n2 --- rem quot LO
Leave the remainder and signed quotient of n1/n2. The remainder has the sign of the dividend.

0 1 2 3 --- n

These small numbers are used so often that is is attractive to define them by name in the dictionary as constants.

0< n --- f L0

Leave a true flag if the number is less than zero (negative), otherwise leave a false flag.

0= n --- f L0

Leave a true flag is the number is equal to zero, otherwise leave a false flag.

0BRANCH f --- C2

The run-time proceedure to conditionally branch. If f is false (zero), the following in-line parameter is added to the interpretive pointer to branch ahead or back. Compiled by IF, UNTIL, and WHILE.

1+ n1 --- n2 L1

Increment n1 by 1.

2+ n1 --- n2

Leave n1 incremented by 2.

: P,E,L0

Used in the form called a colon-definition:

```
: cccc ... ;
```

Creates a dictionary entry defining cccc as equivalent to the following sequence of Forth word definitions '...' until the next ';' or ';CODE'. The compiling process is done by the text interpreter as long as STATE is non-zero. Other details are that the CONTEXT vocabulary is set to the CURRENT vocabulary and that words with the precedence bit set (P) are executed rather than being compiled.

; P,C,L0

Terminate a colon-definition and stop further compilation. Compiles the run-time ;S.

;CODE P,C,L0

Used in the form:

```
: cccc .... ;CODE
    assembly mnemonics
```

Stop compilation and terminate a new defining word cccc by compiling (;CODE). Set the CONTEXT vocabulary to ASSEMBER, assembling to machine code the following mnemonics.

When cccc later executes in the form:

```
cccc nnnn
```

the word nnnn will be created with its execution proceedure given by by the machine code following cccc. That is, when nnnn is executed, it does so by jumping to the code after nnnn. An existing defining word must exist in cccc prior to ;CODE.

;S P,L0

Stop interpretation of a screen. ;S is also the run-time word compiled at the end of a colon-definition which returns execution to the calling proceedure.

< n1 n2 --- f L0

Leave a true flag if n1 is less than n2; otherwise leave a false flag.

<# L0

Setup for pictured numeric output formatting using the words:

```
<# # #S SIGN #>
```

The conversion is done on a double number producing text at PAD.

<BUILDS C,L0

Used within a colon-definition:

```
: cccc <BUILDS ...
       DOES> ... ;
```

Each time cccc is executed, <BUILDS defines a new word with a high-level execution proceedure. Executing cccc in the form:

```
cccc nnnn
```

uses <BUILDS to create a dictionary entry for nnnn with a call to the DOES> part for nnnn. When nnnn is later executed, it has the address of its parameter area on the stack and executes the words after DOES> in cccc. <BUILDS and DOES> allow run-time proceedures to written in high-level rather than in assembler code (as required by ;CODE).

= n1 n2 --- f L0

Leave a true flag if n1=n2; otherwise leave a false flag.

> n1 n2 --- f L0

Leave a true flag if n1 is greater than n2; otherwise a false flag.

>R n --- C,L0

Remove a number from the computation stack and place as the most accessable on the return stack. Use should be balanced with R> in the same definition.

? addr -- L0

Print the value contained at the address in free format according to the current base.

?COMP

Issue error message if not compiling.

?CSP

Issue error message if stack position differs from value saved in CSP.

?ERROR f n ---
Issue an error message number n, if the boolean flag is true.

?EXEC
Issue an error message if not executing.

?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.

?STACK
Issue an error message is the stack is out of bounds. This definition may be installation dependent.

?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.

@ addr --- n LO
Leave the 16 bit contents of address.

ABORT LO
Clear the stacks and enter the execution state. Return control to the operators terminal, printing a message appropriate to the installation.

ABS n --- u LO
Leave the absolute value of n as u.

AGAIN addr n --- (compiling) P,C2,LO
Used in a colon-definion 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.

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).

AND n1 n2 --- n2 LO
Leave the bitwise logical and of n1 and n2 as n3.

B/BUF --- n
This constant leaves the number of bytes per disc buffer, the byte count read from disc by BLOCK.

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.

BACK addr ---
Calculate the backward branch offset from HERE to addr and compile into the next available dictionary memory address.

BASE --- addr U,LO
A user variable contaning the current number base used for input and output conversion.

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 correspoinding 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.

At compile time BEGIN leaves its return address and n for compiler error checking.

BL --- c
A constant that leaves the ascii value for "blank".

BLANKS addr count ---
Fill an area of memory begining at addr with blanks.

BLK --- addr U,LO
A user variable containing the block number being interpreted. If zero, input is being taken from the terminal input buffer.

BLOCK 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 rewritten 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 proceedure 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 minicomputers.

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 proceedure 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 C2
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 additon to simply compling 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.

CR L0
Transmit a carriage return and line feed to the selected output device.

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 vocablary.

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

D+ d1 d2 --- dsum
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 accessable on the stack. Conversion is performed according to the current BASE. A blank follows. Pronounced D-dot.

D.R d n ---
Print a signed double number d right aligned in a field n characters wide.

DABS d --- ud
Leave the absolute value ud of a double number.

DECIMAL L0
Set the numeric conversion BASE for decimal input-output.

DEFINITIONS L1
Used in the form:
cccc DEFINITIONS
Set the CURRENT vocabulary to the CONTEXT vocabulary. In the example, executing vocabulary name cccc made it the CONTEXT vocabulary and executing DEFINITIONS made both specify vocabulary cccc.

DIGIT c n1 --- n2 tf (ok)
c n1 --- ff (bad)
Converts the ascii character c (using base n1) to its binary equivalent n2, accompanied by a true flag. If the conversion is invalid, leaves only a false flag.

DLIST
List the names of the dictionary entries in the CONTEXT vocabulary.

DLITERAL d --- d (executing)
d --- (compiling) P
If compiling, compile a stack double number into a literal. Later execution of the definition containing the literal will push it to the stack. If executing, the number will remain on the stack.

DMINUS d1 --- d2
Convert d1 to its double number two's complement.

DO n1 n2 --- (execute)
addr n --- (compile) P,C2,L0
Occurs in a colon-definition in form:
DO ... LOOP
DO ... +LOOP
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

urrent value of the index to the stack. See I, LOOP, +LOOP, LEAVE.

When compiling within the colon-definition, DO compiles (DO), leaves the following address addr and n for later error checking.

DOES> L0
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-diminsional arrays, and compiler generation.

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.

DPL ---- addr U,L0
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 formating. The default value on single number input is -1.

DR0
DR1 Installation dependent commands to select disc drives, by preseting OFFSET. The contents of OFFSET is added to the block number in BLOCK to allow for this selection. Offset is supressed for error text so that is may always originate from drive 0.

DROP n --- L0
Drop the number from the stack.

DUMP addr n --- L0
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 L0
Duplicate the value on the stack.

ELSE addr1 n1 --- addr2 n2
(compiling) P,C2,L0
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 --- L0
Transmit ascii character c to the selected output device. OUT is incremented for each character output.

EMPTY-BUFFERS L0
Mark all block-buffers as empty, not necessarily affecting the contents. Updated blocks are not written to the disc. This is also an initialization proceedure before first use of the disc.

ENCLOSE addr1 c ---
ddr1 n1 n2 n3
The text scanning primitive used by WORD. From the text address addr1 and an ascii delimiting character c, is determined the byte offset to the first non-delimiter character n1, the offset to the first delimiter after the text n2, and the offset to the first character not included. This proceedure will not process past an ascii 'null', treating it as an unconditional delimiter.

END P,C2,L0
This is an 'alias' or duplicate definition for UNTIL.

ENDIF addr n --- (compile) P,C0,L0
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.

At compile-time, ENDIF computes the forward branch offset from addr to HERE and stores it at addr. n is used for error tests.

ERASE addr n ---
Clear a region of memory to zero from addr over n addresses.

ERROR line --- in blk
Execute error notification and restart of system. WARNING is first examined. If 1, the text of line n, relative to screen 4 of drive 0 is printed. This line number may be positive or negative, and beyond just screen 4. If WARNING=0, n is just printed as a message number (non disc installation). If WARNING is -1, the definition (ABORT) is executed, which executes the system ABORT. The user may cautiously modify this execution by altering (ABORT). fig-FORTH saves the contents of IN and BLK to assist in determining the location of the error. Final action is execution of QUIT.

EXECUTE addr --
Execute the definition whose code field address is on the stack. The code field address is also called the compilation address.

EXPECT addr count --- L0
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.

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.

FILL addr quan b ---
Fill memory at the address with the specified quantity of bytes b.

FIRST --- n
A constant that leaves the address of the first (lowest) block buffer.

FLD --- addr U

A user variable for control of number output field width. Presently unused in fig-FORTH.

FORGET E,LO

Executed in the form:
FORGET cccc
Deletes definition named cccc from the dictionary with all entries physically following it. In fig-FORTH, an error message will occur if the CURRENT and CONTEXT vocabularies are not currently the same.

FORTH P,L1

The name of the primary vocabulary. Execution makes FORTH the CONTEXT vocabulary. Until additional user vocabularies are defined, new user definitions become a part of FORTH. FORTH is immediate, so it will execute during the creation of a colon-definition, to select this vocabulary at compile time.

HERE --- addr LO

Leave the address of the next available dictionary location.

HEX LO

Set the numeric conversion base to sixteen (hexadecimal).

HLD --- addr LO

A user variable that holds the address of the latest character of text during numeric output conversion.

HOLD c --- LO

Used between <# and #> to insert an ascii character into a pictured numeric output string.
e.g. 2E HOLD will place a decimal point.

I --- n C,LO

Used within a DO-LOOP to copy the loop index to the stack. Other use is implementation dependent. See R.

ID. addr ---

Print a definition's name from its name field address.

IF f --- (run-time)
--- addr n (compile) P,C2,LO

Occurs is a colon-definition in form:
IF (tp) ... ENDIF
IF (tp) ... ELSE (fp) ... ENDIF
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.

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.

IMMEDIATE

Mark the most resently 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 preceeding it with [COMPILE].

IN --- addr LO

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.

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.

INTERPRET

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.

KEY --- c L0
Leave the ascii value of the next terminal key struck.

LATEST --- addr
Leave the name field address of the topmost word in the CURRENT vocabulary.

LEAVE C,L0
Force termination of a DO-LOOP at the next opportunity by setting the loop limit equal to the current value of the index. The index itself remains unchanged, and execution prodeeds normally until LOOP or +LOOP is encountered.

LFA pfa --- lfa
Convert the parameter field address of a dictionary definition to its link field address.

LIMIT ---- n
A constant leaving the address just above the highest memory available for a disc buffer. Usually this is the highest system memory.

LIST n --- L0
Display the ascii text of screen n on the selected output device. SCR contains the screen number during and after this process.

LIT --- n C2,L0
Within a colon-definition, LIT is automatically compiled before each 16 bit literal number encountered in input text. Later execution of LIT causes the contents of the next dictionary address to be pushed to the stack.

LITERAL n --- (compiling) P,C2,L0
If compiling, then compile the stack value n as a 16 bit literal. This definition is immediate so that it will execute during a colon definition. The intended use is:
: xxx [calculate] LITERAL ;
Compilation is suspended for the compile time calculation of a value. Compilation is reusumed and LITERAL compiles this value.

LOAD n --- L0
Begin interpretation of screen n. Loading will terminate at the end of the screen or at ;S. See ;S and -->.

LOOP addr n --- (compiling) P,C2,L0
Occurs in a colon-definition in form:
DO ... LOOP
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.

At compile-time, LOOP compiles (LOOP) and uses addr to calculate an offset to DO. n is used for error testing.

M* n1 n2 --- d
A mixed magnitude math operation which leaves the double number signed product of two signed number.

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.

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.

MAX n1 n2 --- max L0
Leave the greater of two numbers.

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).

MIN n1 n2 --- min L0
Leave the smaller of two numbers.

MINUS n1 --- n2 L0
Leave the two's complement of a number.

MOD n1 n2 --- mod L0
Leave the remainder of n1/n2, with the same sign as n1.

MON
Exit to the system monitor, leaving a re-entry to Forth, if possible.

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.

NEXT
This is the inner interpreter that uses the interpretive pointer IP to execute compiled Forth definitions. It is not directly executed but is the return point for all code proceedures. It acts by fetching the address pointed by IP, storing this value in register W. It then jumps to the address pointed to by 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.

NFA pfa --- nfa
Convert the parameter field address of a definition to its name field.

NUMBER addr --- d
Convert a character string left at addr with a preceeding 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.

OFFSET --- addr U
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, DR0, DR1, MESSAGE.

OR n1 n2 -- or L0
Leave the bit-wise logical or of two 16 bit values.

OUT --- addr U
A user variable that contains a value incremented by EMIT. The user may alter and examine OUT to control display formating.

OVER n1 n2 --- n1 n2 n1 L0
Copy the second stack value, placing it as the new top.

PAD --- addr L0
Leave the address of the text output buffer, which is a fixed offset above HERE.

PFA nfa --- pfa
Convert the name field address of a compiled definition to its parameter field address.

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.

PREV ---- addr
A variable containing the address of the disc buffer most recently referenced. The UPDATE command marks this buffer to be later written to disc.

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.

PUT
This code sequence stores machine register contents over the topmost computation stack value and returns to NEXT. It is not directly executable, but is a Forth re-entry point after machine code.

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.

QUIT L1
Clear the return stack, stop compilation, and return control to the operators terminal. No message is given.

R --- n
Copy the top of the return stack to the computation stack.

R# --- addr U
A user variable which may contain the location of an editing cursor, or other file related function.

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.

R0 --- addr U
A user variable containing the initial location of the return stack. Pronounced R-zero. See RP!

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 correspoinding BEGIN.

At compile-time, REPEAT compiles BRANCH and the offset from HERE to addr. n is used for error testing.

ROT n1 n2 n3 --- n2 n3 n1 L0
Rotate the top three values on the stack, bringing the third to the top.

RP!
A computer dependent proceedure to initialize the return stack pointer from user variable R0.

S->D n --- d
Sign extend a single number to form a double number.

S0 --- addr U
A user variable that contains the initial value for the stack pointer. Pronounced S-zero. See SP!

SCR --- addr U
A user variable containing the screen number most recently reference by LIST.

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 #>.

SMUDGE
Used during word definition to toggle the "smudge bit" in a definitions' name field. This prevents an un-completed definition from being found during dictionary searches, until compiling is completed without error.

SP!
A computer dependent proceedure to initialize the stack pointer from S0.

SP@ --- addr
A computer dependent proceedure 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)

SPACE L0
Transmit an ascii blank to the output device.

SPACES n --- L0
Transmit n ascii blanks to the output device.

STATE --- addr L0,U
A user variable containg the compilation state. A non-zero value indicates compilation. The value itself may be implementation dependent.

SWAP n1 n2 --- n2 n1 L0
Exchange the top two values on the stack.

TASK
A no-operation word which can mark the boundary between applications. By forgetting TASK and re-compiling, an application can be discarded in its entirety.

THEN P,C0,L0
An alias for ENDIF.

TIB --- addr U
A user variable containing the address of the terminal input buffer.

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.

TRIAD scr ---
Display on the selected output device the three screens which include that numbered scr, begining 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.

U* u1 u2 --- ud
Leave the unsigned double number product of two unsigned numbers.

U/ ud u1 --- u2 u3
Leave the unsigned remainder u2 and unsigned quotient u3 from the unsigned double dividend ud and unsigned divisor u1.

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.

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.

USE --- addr
A variable containing the address of the block buffer to use next, as the least recently written.

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.

VARIABLE E,LO
A defining word used in the form:
n VARIABLE cccc
When VARIABLE is executed, it creates the definition cccc with its parameter field initialized to n. When cccc is later executed, the address of its parameter field (containing n) is left on the stack, so that a fetch or store may access this location.

VOC-LINK --- addr U
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.

VOCABULARY E,L
A defining word used in the form:
VOCABULARY cccc
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.

In fig-FORTH, cccc will be so chained as to include all definitions of the vocabulary in which cccc is itself defined. All vocabularys ulitmately chain to Forth. By convention, vocabulary names are to be declared IMMEDIATE. See VOC-LINK.

VLIST
List the names of the definitions in the context vocabulary. "Break" will terminate the listing.

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 proceedure. See MESSAGE, ERROR.

WHILE f --- (run-time)
ad1 n1 --- ad1 n1 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 contintues 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 --- L0

Read the next text characters from the input stream being interpreted, until a delimiter c is found, storing the packed character string begining at the dictionary buffer HERE. WORD leaves the character count in the first byte, the characters, and ends with two or more blanks. Leading occurances 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 proceedure 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 defininition, that would otherwise execute during compilation. The above example will select the FORTH vocabulary when xxx executes, rather than at compile time.

] L1

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

```
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 'S0' *)_ 5 26
CODE RP!                                  ( LOAD RP FROM R0 *)_ 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
3BE0    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 $00A0 *)_ 1 36
( 06 USER  S0 )              ( TOP OF EMPTY COMPUTATION STACK *)_ 2 36
( 08 USER  R0 )                   ( 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
: ]    C0  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 LARGET 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 RATION OF THREE NUMBERS, LEAVING *)_ 11 57
         >R  M*  R>  M/  ;                ( REM-2, QUOTIENT-1 *)_ 12 57
: */     */MOD  SWAP  DROP  ;   ( LEAVE RATIO OF THREE NUMBS *)_ 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
: DR0      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, 0 TO WRITE *)_ 3 69
: R/W                               ( READ/WRITE DISC BLOCK *)_ 4 69
              ( BUFFER ADDRESS-3, BLOCK #-2, 1=READ 0=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~~. 6502

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.

.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 low 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 high 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 # 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 COMPILATION ONLY, USE IN DEFINITION
  2 EXECUTION ONLY
  3 CONDITIONALS NOT PAIRED
  4 DEFINITON 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
```

```
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,  PUSH0A 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,  PUSH0A  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    FD0C JSR,  7F # AND,  PUSH0A JMP,
  6 HERE  ' EMIT  2  -  !   (  POINT EMIT TO HERE  )
  7    BOT 1+ LDA,  80 # ORA,  FDED 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    C000 BIT,  0<
 12       IF,  BEGIN,  C010 BIT,  C000 BIT,  0< NOT END,  INY,
 13         THEN,  TYA,  PUSH0A JMP,
 14
 15 DECIMAL     ;S


SCR # 8
  0 (  INPUT-OUTPUT,  SYM-1                                  WFR-781015 )
  1 HEX
  2 CODE KEY    8A58 JSR,  7F # AND,  PUSH0A 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,  PUSH0A  JMP,
 12
 13
 14
 15 DECIMAL     ;S
```

```
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,  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
  0 (  DIGIT                                                  WFR-781202 )
  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
  0 (  FIND FOR VARIABLE LENGTH NAMES                         WFR-790225 )
  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,  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, ( 0 LINK ? )
 14      0= END, ( LOOP FOR ANOTHER NAME )
 15      XSAVE LDX,  0 # LDA, PHA,  PUSH JMP, ( FALSE )    -->


SCR # 20
  0 (  ENCLOSE                                                WFR-780926 )
  1 CODE ENCLOSE    ( 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       0= 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          RS-WFR-80AUG16 )
   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 multiplier )
   8          CS IF, ( set ) CLC,
   9             ( add multiplicand to partial product 32 bits )
  10               N    LDA,  BOT 2 + ADC,  BOT 2 + STA,
  11               N 1+ LDA,  BOT 3 + ADC,  BOT 3 + STA,
  12             CS IF,  BOT INC,  0= IF,  BOT 1+ INC, ENDIF, ENDIF,
  13           ENDIF,  DEY,  0=    ( corrected for carry bug )
  14       UNTIL,      NEXT JMP,   C;
  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 'S0' *)
  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,  0= 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,  0= 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,  0= 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 3BE0    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 $00A0 *)
  2 ( 06 USER  S0 )                ( TOP OF EMPTY COMPUTATION STACK *)
  3 ( 08 USER  R0 )                     ( 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 : ]     C0  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                                              WFR-79APR20 )
  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 : ."     22  STATE  @       ( COMPILE OR PRINT QUOTED STRING *)
 13    IF  COMPILE  (.")        WORD    HERE  C@  1+  ALLOT
 14        ELSE        WORD    HERE   COUNT  TYPE  ENDIF  ;
 15                IMMEDIATE     -->

FORTH INTEREST GROUP                                  MAY 1, 1979
```

```
SCR # 45
  0 (  TERMINAL INPUT                                        WFR-79APR29 )
  1
  2 : EXPECT              ( TERMINAL INPUT MEMORY-2,  CHAR LIMIT-1 *)
  3     OVER  +  OVER  DO  KEY  DUP  0E  +ORIGIN ( BS )  @  =
  4     IF  DROP  08  OVER  I  =  DUP  R>  2  -  +  >R  -
  5        ELSE ( NOT BS )  DUP  0D  =
  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 ( disc dependent )
 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  0A0  +  <  2  ?ERROR  ( 6502 only )
  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@  0FD  =  ALLOT    ( 6502 only )
 10       DUP  A0  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 COMPILATION 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                                          WFR-79MAR30 )
  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                                            WFR-79APR29 )
  1 CODE COLD                ( COLD START, INITIALIZING USER AREA *)
  2    HERE  02  +ORIGIN  !  ( POINT COLD ENTRY TO HERE )
  3          0C +ORIGIN LDA,  'T FORTH 4 + STA,  ( FORTH VOCAB. )
  4          0D +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       0F # LDY,  ( INDEX TO WARNING )   THEN, ( FROM IF, )
  8       10 +ORIGIN LDA,   UP    STA,  ( LOAD UP )
  9       11 +ORIGIN LDA,   UP 1+ STA,
 10        BEGIN,  0C +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                                    DJK-WFR-79APR29 )
  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                          ( LEAVE SMALLER OF TWO NUMBERS *)
 13         OVER  OVER  >  IF  SWAP  ENDIF  DROP  ;
 14 : MAX                           ( LEAVE LARGET OF TWO NUMBERS *)
 15         OVER  OVER  <  IF  SWAP  ENDIF  DROP  ; -->
```

```
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  0  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 : DR0       0  OFFSET  !  ;                    ( SELECT DRIVE #0 *)
 15 : DR1    07D0  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  0  IN  !  B/SCR  *  BLK !
  4     INTERPRET  R>  IN  !  R>  BLK  !  ;
  5
  6 : -->               ( CONTINUE INTERPRETATION ON NEXT SCREEN *)
  7      ?LOADING  0  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,  PUSH0A 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,    PUSH0A 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  0A  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,  C0 # 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,  C0 # 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

FORTH INTEREST GROUP                                    MAY 1, 1979
```

```
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, 1=READ 0=WRITE *)
  6    LITERAL  HLD  ! ( JUST AFTER WORKSPACE )   SWAP
  7    0  OVER  >  OVER  0F9F  >  OR  6  ?ERROR
  8    07D0  ( 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  @  0A  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  DR0
 13 2C  BYTE.IN  BUFFER       REPLACED.BY  R/W
 14 30  BYTE.IN  BLOCK        REPLACED.BY  R/W       DECIMAL  ;S
 15
```

```
SCR # 72
  0 (   ',   FORGET,   \                                    WFR-79APR28 )
  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   0BRANCH   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   0BRANCH    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           0C  EMIT ( FORM FEED )  CR  1+  SWAP
  9           DO  CR  I  3  .R  SPACE
 10               0  I  .LINE
 11               ?TERMINAL  IF  LEAVE  ENDIF  LOOP  ;
 12 : TRIAD      ( PRINT 3 SCREENS ON PAGE, CONTAINING # ON STACK *)
 13           0C  EMIT ( FF )  3  /  3  *  3  OVER  +  SWAP
 14           DO  CR  I  LIST  LOOP  CR
 15           0F  MESSAGE  CR  ;     DECIMAL   -->
```

```
SCR # 78
  0 (  TOOLS                                                  WFR-79APR20 )
  1 HEX
  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                                                  WFR-79MAY03 )
  1 HEX
  2
  3 CREATE  MON          ( CALL MONITOR, SAVING RE-ENTRY TO FORTH *)
  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
```

This is a sample editor, compatable 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  1  -  ( LIMIT )  0E ( 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  0F  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  0  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 : 1LINE        ( 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          1LINE  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                              WFR-79MAR24 )
  1 : N      ( FIND NEXT OCCURANCE OF PREVIOUS TEXT *)
  2       FIND  0  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  0  M  ;
 12
 13 : TILL   ( DELETE ON CURSOR LINE, FROM CURSOR TO TEXT END *)
 14       #LEAD  +  1  TEXT  1LINE  0=  0  ?ERROR
 15       #LEAD  +  SWAP  -  DELETE  0  M  ;      -->


SCR # 97
  0 (  STRING EDITOR COMMANDS                              WFR-79MAR23 )
  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     0  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


SCR # 98
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```