ES FORTH

ATARI 400/800 & XL MODELS 32K

SPECIAL FEATURES OF ES FORTH:

- Basic dictionary compatible with FIG-FORTH.
- Many general purpose extensions.
- Full screen editor.
- Complete set of I/O extensions.
- Built-in editor definitions.
- Can also be used with ATARI ASSEMBLER EDITOR for debugging and machine code linking.
- Built-in sound commands.
- ATARI BASIC compatible graphics commands.
- Automatic control of ATARI PLAYER-MISSILE graphics.
- Complete set of controller commands.
- User selectable number of editing screens.

ES FORTH

ATARI 400/800 & XL MODELS 32K

ES-FORTH

USERS MANUAL

CASSETTE VERSION C1.2

FOR ATARI 32K COMPUTER SYSTEMS

THE ENGLISH SOFTWARE COMPANY.

BY

A. KOZAKEWYCZ

Original documentation provided through the courtesy of the FORTH INTEREST GROUP, P.O. Box 1105, San Carlos, CA 94070.

THE ENGLISH SOFTWARE COMPANY, P.O. BOX 43, MANCHESTER M60 3AD.
Telephone 061-835-1358

THE POWER OF EXCITEMENT

PREFACE

This manual does not set out to teach the user how to program in FORTH. It assumes the user has already read an introductory book and is familiar with FORTH's structure and in particular the nature of FORTH'S "STACK" should be well understood. The user should also be familiar with the ATARI personal computer system's SCREEN EDITOR and control functions. Note at least 32K of RAM is required to operate ES-FORTH.

The primary function of this manual is to provide a glossary of all the definitions within ES-FORTH and as such is divided into three main sections.

SECTION 1 deals with all the definitions in ES-FORTH which are compatible with fig-FORTH.

SECTION 2 deals with the general purpose utility words which are not available in fig-FORTH.

SECTION 3 deals with all the special sound, graphics and controller commands.

INTRODUCTION

ES-FORTH allows the user to write fast programmes without resorting to machine code. FORTH is typically 10 times faster than BASIC and sometimes up to 100 times faster. This speed coupled with the large range of graphics commands available in ES-FORTH allows the user to create fast games and utility programmes relatively quickly.

Although not essential the user will find it an advantage to know about the structure of memory within a computer as most of FORTH's definitions are concerned with accessing and storing bytes in memory locations.

ES-FORTH is fully compatible with fig-FORTH as it is derived from the same model. Although some words in the dictionary are not internally identical they will have the same result as the equivalent fig definition.

ES-FORTH has a number of advantages over fig-FORTH. One of these being the ability to use both upper and lower case text in definitions names. In ES-FORTH any combination of ASCII characters up to a length of 31 can be used in a name. It should be noted though that reverse video characters should not be used in names as this may cause a crash. Reverse video is freely available otherwise.

The full screen editor is available in ES-FORTH and text can be processed anywhere on the screen. The full fig editor is not provided as the screen editor on the ATARI makes many of the definitions redundant.

In order to allow cassette operation on what is basically a disc dependant system, a user selectable portion of RAM is reserved to simulate disc-drives.

N.B. There are two versions of ES-FORTH, one is supplied on disc and one on cassette. Although both versions will run the same programme, each cannot directly load programmes from another MEDIUM, i.e. the cassette version cannot access the disc.

It is possible to run ES-FORTH with the ATARI ASSEMBLER-EDITOR CARTRIDGE installed. This leads to the possibility of linking machine code programmes through ES-FORTH's "CALL" command. A possible use for this is the generation of Display list interrupts to create more colours on screen.

To use the cartridge make sure it is already installed on power-up and follow the normal loading procedure. When FORTH is loaded typing "DOS" will take the user into FORTH. Use FORTH's "*ALLOT*" command to reserve memory for machine code.

LOADING FORTH INTO THE COMPUTER

ES-FORTH is supplied on a self booting cassette tape. It is not necessary to have any cartridge installed.

The tape should be inserted in the programme recorder and rewound with the computer switched off. The computer should then be turned on with the start button depressed. A single audible tone will sound. The user should then press the return key (make sure the play button on the programme recorder is depressed). This will initiate tape loading. The programme is approximately 10K bytes long and should take about 5 minutes to load. If the computer shows a boot error or returns to memo-pad during loading, try cueing the tape on a normal cassette player (rewind and listen for the leader tone).

LOADING: Wind cassette forward one inch past the leader tape!
Cassette: Empty cartridge slot. Switch computer on whilst holding down the **START** button. On the XL models, hold down the **OPTION** key also. Press play, then press **RETURN**.
Disk: Empty cartridge slot. Switch computer on. On the XL models, hold down the **OPTION** key also. Disk will load automatically.

ALLOCATING EDITOR TEXT MEMORY (EDM)

When ES-FORTH is fully loaded the greeting message will appear and a prompt for the number of editing screens the user requires. The user will have to balance how much programme memory against how much editing memory he needs. Each editing screen requires 1024 bytes of memory (16 lines x 64 CHARs). It is possible to gain more system memory if the full modes 8-11 graphics are not required (8-15 on XL's). (See Appendix A for memory assignment).

A useful figure is 5 editing screens to start with (user should type 5 then return). This leaves 7K in a typical 32K system for compiled programmes. This is quite a lot of memory as FORTH programmes tend to be much more compact than their BASIC equivalent.

After entering the required number the screen will clear and FORTH's "OK" prompt will appear. If for any reason the number is out of range FORTH will restart.

N.B. FORTH will restart if the number of editing screens leaves less than 1000 bytes of dictionary space.

USING SYSTEM RESET

Using the system reset button has no detrimental effect on FORTH and is a useful way of exiting infinite loops. If a cartridge is installed use the "DOS" command to re-enter ES-FORTH.

RESET does not clear Edit memory or erase newly created definitions from the dictionary.

SYSTEM CRASHES

All RAM based languages are susceptible to crashes and FORTH is no exception.

Crashes are nearly always caused by errors in the user's programme and they are normally fatal. If the system does not respond to the reset then the only option is to turn off the computer and re-boot.

The most common cause of crashes in FORTH is stack overrun and underrun. The stack is checked when the keyboard has control of the system but not when compiled definitions are executing.

To avoid this the user must ensure that definitions do not remove or leave parameters on the stack which are not going to be used.

Another common error is the use of the >R definition. This must be balanced with R> within the same definition to avoid crashing.

The lack of error checking at run time in FORTH may seem a hindrance, but it is a necessary sacrifice in order to achieve the extremely high execution speed.

THE "USER" COMMAND

It is not recommended that the programmer utilise the "USER" command as most of the "USER" variables available have already been allocated to system variables. It is just as easy and much safer to use "VARIABLE" where variables are necessary.

MATHEMATICS IN ES-FORTH

Unless otherwise stated all numbers are 16 bit signed integers. The low byte is on top of the stack, the high byte is second on the stack with the sign in the leftmost bit. This allows numbers in the range +32767 to -32768 to be handled. In applications where a memory address is required the sign bit is ignored and the number treated as a 16 bit unsigned integer.

Some definitions support 32 bit signed double numbers. In these cases the most significant part (with the sign) is on top of the stack.

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

SECTION 1 (GLOSSARY)

The following Glossary contains all the word definitions in ES-FORTH which are compatible with fig-FORTH. The Glossary is presented in ALPHABETICAL order.

The first line of each entry shows a symbolic description of the action of the procedure on the parameter stack. The symbols indicate the order in which input parameters have been placed on the stack. Three dashes "---" indicates the execution point (substitute the word name where the dashes appear). Any parameters left on the stack are listed. In this notation the top of the stack is to the RIGHT.

SYMBOL TABLE

addr	16 bit memory address
b	8 bit byte (high 8 bits zero)
c	8 bit ASCII character
d	32 bit signed double integer, most significant portion with sign on top of stack.
f	boolean flag. 0=false, none-zero=true
ff	boolean false flag, flag=0
tf	bolean true flag, flag=non-zero
n	16 bit signed integer number
u	16 bit unsigned integer

The capital letters on the right show definition characteristics:

C	may only be used within a colon definition. A digit shows number of memory address used, if other than one.
E	Intended for execution only
L0	Level zero definition of FORTH-78
L1	Level one definition of FORTH-79
P	Has precedence bit set. Will execute even when compiling
U	A user variable

! 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 procedure, compiled by ." which transmits the following in-line text to the selected output device. See ."

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

(+LOOP) n --- C2
The run-time procedure 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 procedure.

(DO) C
The run-time procedure 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 procedure 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 addr1 --- 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 stream to HERE, and searches the CONTEXT and then CURRENT vocabularies for a matching entry. If found, the dictionary entry's parameter field address, its length byte, and a boolean true is left. Otherwise, only a boolean false is left.

-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 procedure 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 LO

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

0= n --- f LO

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

0BRANCH f --- C1

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,LO

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,LO

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

;CODE P,C,LO

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 ASSEMBLER, 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,LO

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 LO

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

<# LO

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,LO

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 LO

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

> n1 n2 --- f LO

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

>R n --- C,LO

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 -- LO

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) F,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 --- n3 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 containing the current number base used for input and output conversion.

BEGIN --- addr n (compiling) F,LO
Occurs in a colon-definition in form:
BEGIN ... UNTIL
BEGIN ... AGAIN
BEGIN ... WHILE ... REPEAT
At run-time, BEGIN marks the start of a sequence that may be repetitively executed. It serves as a return point from the corresponding UNTIL, AGAIN or REPEAT. When executing UNTIL, a return to BEGIN will occur if the top of the stack is false; for AGAIN and REPEAT a return to BEGIN always occurs.

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

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 vocabulary.

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 LO
Set the numeric conversion BASE for decimal input-output.

DEFINITIONS LI
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.

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,LO
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> LO
A word which defines the run-time action within a high-level defining word. DOES> alters the code field and first parameter of the new word to execute the sequence of compiled word addresses following DOES>. Used in combination with <BUILDS. When the DOES> part executes it begins with the address of the first parameter of the new word on the stack. This allows interpretation using this area or its contents. Typical uses include the Forth assembler, multi-dimensional arrays, and compiler generation.

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,LO
A user variable containing the number of digits to the right of the decimal on double integer input. It may also be used hold output column location of a decimal point, in user generated formating. The default value on single number input is -1.

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

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

DUMP addr n --- LO
Print the contents of n memory locations beginning at addr. Both addresses and contents are shown in the current numeric base.

DUP n --- n n LO
Duplicate the value on the stack.

ELSE addr1 n1 --- addr2 n2
(compiling) P,C2,LO
Occurs within a colon-definition in the form:
IF ... ELSE ... ENDIF
At run-time, ELSE executes after the true part following IF. ELSE forces execution to skip over the following false part and resumes execution after the ENDIF. It has no stack effect.

At compile-time ELSE emplaces BRANCH reserving a branch offset, leaves the address addr2 and n2 for error testing. ELSE also resolves the pending forward branch from IF by calculating the offset from addr1 to HERE and storing at addr1.

EMIT c --- LO
Transmit ascii character c to the selected output device. OUT is incremented for each character output.

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

ENCLOSE addr1 c ---
addr1 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 procedure will not process past an ascii 'null', treating it as an unconditional delimiter.

END P,C2,LO
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 re-start 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 --- LO
Transfer characters from the terminal to address, until a "return" or the count of characters have been received. One or more nulls are added at the end of the text.

FENCE --- addr 0
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,LI

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 in 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 recently made definition so that when encountered at compile time, it will be executed rather than being compiled. i.e. the precedence bit in its header is set. This method allows definitions to handle unusual compiling situations, rather than build them into the fundamental compiler. The user may force compilation of an immediate definition by 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 LO
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,LO
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 proceeds 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 --- LO
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,LO
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,LO
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 resumed and LITERAL compiles this value.

LOAD n --- LO
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,LO
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 LO
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 LO
Leave the smaller of two numbers.

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

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

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 procedures. 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 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 LO
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 corresponding 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 LO
Rotate the top three values on the stack, bringing the third to the top.

RP!
A computer dependent procedure 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 referenced by LIST.

SIGN n d --- d LO
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 uncompleted definition from being found during dictionary searches, until compiling is completed without error.

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

SP@ --- addr
A computer dependent procedure to return the address of the stack position to the top of the stack, as it was before SP@ was executed. (e.g. 1 2 SP@ @ . . . would type 2 2 1)

SPACE LO
Transmit an ascii blank to the output device.

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

STATE --- addr LO,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 LO
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,LO
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,Lu

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 @ 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 ultimately 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)

adl n1 --- adl 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 continues execution of the true part thru to REPEAT, which then branches back to BEGIN. If f is false (zero), execution skips to just after REPEAT, exiting the structure.

At compile time, WHILE emplaces (OBRANCH) and leaves ad2 of the reserved offset. The stack values will be resolved by REPEAT.

WIDTH --- addr U

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

WORD c --- LO

Read the next text characters from the input stream being interpreted, until a delimiter c is found, storing the packed character string 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 definintion, 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 [.

SECTION II

The following Section deals with the commands not generally available in fig-FORTH and is divided into three parts. Part 1 covers general purpose definitions. Part 2 details editor definitions and Part 3 covers cassette handling and I/O.

GENERAL PURPOSE DEFINITIONS

1- n_1 — n_2
Decrements n1 by 1

2- n_1 — n_2
Decrements n1 by 2

2* n_1 — n_2
fast Multiply by 2

CALL A Y X addr — A Y X
Loads the processor registers from the stack and performs a JSR to address. Register values are saved on the stack before returning to FORTH. (Useful for accessing OS routines).

CLIT — n
When encountered within a colon definition causes the 8 bit contents of the next dictionary location to be pushed to the stack. There is no high level word which compiles CLIT and so is not useful to the programmer.

C/L — n
Constant (set to 64) which leaves the number of characters/line.

FREE — n
Leaves the amount of RAM left in the dictionary for compiling new definitions.

HIED --- addr

Variable which holds the highest memory address available to the text editor.

INPUT --- n

Waits for the user to enter a number from the keyboard. Numbers are converted with reference to "BASE" and if unconvertible an appropriate error message is given.

J --- n

Used within a double nested "DO-LOOP" to copy the index of the outer loop to the computation stack.

LINE n --- addr

Leaves the current address of line n. This address will be in the buffer area. This command is used by the editor and is not particularly useful to the user.

MOB addr1 addr2 addr3 ---

"MOVE-OBJECT-BLOCK" is primarily intended for use by the PM-GRAPHICS commands. The definition performs the following functions. Erase 16 bytes at addr1, move 16 bytes of data from addr2 to addr3.

R2 --- n

Copies the second value on the return stack to the computation stack.

R3 --- n

Copies the third value on the return stack to the computation stack.

RND --- n

Leaves a 16 bit random number.

TOP — n

Variable, holds the highest memory address available for dictionary expansion. The user should not alter this value as it is automatically calculated from HIED and the number of editing screens selected.

U. u —

Prints the number on the computation stack as an unsigned integer.

WAIT n —

Halts processing for n units of time. 1 unit is 20ms.

EDITING PROGRAMMES

Editing commands are built into ES-FORTH as standard FORTH definitions. Editing screens follow the standard fig layout.

Each SCR is defined as 16 lines of 64 characters. There is no saving in memory by not using the full 64 char line but programmes are tidier if kept to the ATARI 40 char/LINE layout. Any char's after the 64 count are ignored.

ES-FORTH does not edit memory directly but transfers the current SCR selected to a special set of buffers before doing any work on the text. These buffers are transparent to the user as when any other action by FORTH requires the use of them, any data which has been updated is automatically transferred back to its source.

To select a screen for editing one of two commands can be used.

LIST n —

SCR n is listed to the display and selected for editing.

CLEAR n —

SCR n is erased and selected for editing.

The following commands are used to edit text within the SCR

P n —— ccc ccc ccc RETURN

Put text following P onto line no. n. Up to 64 char's are accepted and is terminated with the return key.

E n ——

Erase line n with blanks.

D n ——

Delete line n and move all lines below up 1 line. (Line 15 erased).

S n ——

Spread at line n. Line n-14 is moved down 1 line and then line n is erased. (Line 15 is lost).

COPY n_1 n_2 ——

Copy SCR n_1 to SCR n_2

L ——

Relist current editing screen.

INDEX n_1 n_2 ——

Lists line 0 of screens n_1 to n_2.
It is convention that line 0 of each screen be a comment LINE to indicate the contents of the SCR.

LLIST n ——

Lists SCR n to the line printer.

CORRECTING MISTAKES

It is not necessary to re-type a whole line to correct a mistake. Simply re-list the screen using "L", move the cursor to the beginning of the line in error and insert the "P" command after the line number, then move the cursor along the line and correct the mistake. Press the return key and the screen editor will sort everything out.

ERRORS DURING COMPILATION

When FORTH is compiling from a SCR with the "LOAD" command and an error is met, an error message is printed and loading is aborted. This can be most unhelpful as the error message does not tell the user on what screen and whereabouts the error occurred.

If the user types "WHERE" after one of these errors occur the command will print the SCR and a picture of exactly what it had trouble compiling. Thus making error correction simpler.

LOADING AND SAVING PROGRAMMES

The user should refer to Section 4 of the BASIC reference manual if unsure about recorder operation.

Two simple commands are used to load and save programmes to tape.

CLOAD n ——

Loads text memory starting at SCR n from the programme recorder. Tape loading continues until an EOF is reached or until all the available memory is full.

CSAVE n_1 n_2 ——

Saves SCR nos. n_1 to n_2 inclusive to the programme recorder. Whole screens are saved. CSAVE will not save part of a screen.

Both commands follow the normal ATARI cassette handling procedure.

ES-FORTH I/O WORDS

CIO IOB COM IBA IBL b_1 —— b_2

FORTH's main interface to the ATARI operating system I/O sub-system. Using the IOCB , command, buffer address and buffer length do a JSR to the OS CIO vector.

If the command is a "PUT CHARACTER" operation and IBL is zero then the single byte number b_1 is used as the character to output. For all other commands b_1 is ignored.

If the command was a "GET CHARACTER" operation and IBL is zero then the single byte number b2 corresponds to the character "GOT". For all other operations b_2 should be ignored by using the "DROP" command.

If the OS detects an error during I/O then an error message will automatically be reported and "QUIT" executed.

P: S: K: E: C: --- addr

All of these commands leaves the address in memory of the corresponding character strings suitable for use in an "OPEN" command.

OPEN DEV AUX1 AUX2 IOB ---

Opens IOCB IOB. Device names can be any of above. E.g. K: 4 0 1 OPEN , will open the keyboard for input.

CLOSE IOB ---

Close IOCB

PUT IOB char ---

Puts a single character to IOCB

GET IOB --- char

Gets a single character from IOCB

PUT-CHAR IOB addr len ---

Transmits len characters from address to IOCB

GET-CHAR IOB addr len ---

Receives len characters to address from IOCB

PUT-RECORD IOB addr len -- Count

Transmit len characters from address to IOCB until all characters have been sent or until an EOL character is met. Count indicates the actual number of characters sent.

GET-RECORD IOB addr len -- count

Receive len characters from IOCB to address until all characters have been received or until an EOL character is met. Count indicates the actual number of characters received.

PDEV -- addr

Variable that holds the current IOCB no. that output text is being sent to. Used by "TYPE" and "EMIT" and USER alterable with the "IOCB" command. Default for PDEV is zero (screen-editor).

IOCB n --

Selects IOCB n for the output of text. Should be set back to zero (screen editor) when output is complete. E.g. 6 IOCB ."HELLO" 0 IOCB. prints on the graphics screen.

SECTION III (GLOSSARY)

The following Section deals with the special sound, controller and graphics commands.

SOUND

PLAY CH FREQ DIST VOL TIME ---

This command has the same action as the SOUND command except that the time that the sound is to be played can be specified. The command does not halt processing during play time. TIME can be 1 to 256 units, 1 unit being 20mS.

SOUND CH FREQ DIST VOL ---

Action of the sound command is identical to the equivalent ATARI-BASIC command.

CONTROLLER COMMANDS

All of the following commands have the same action as the equivalent command in ATARI-BASIC.

PADDLE n_1 --- n_2

Reads paddle n1, n2 is the value.

PTRIG n_1 --- n_2

Reads trigger button off paddle n1. n2=0 when depressed.

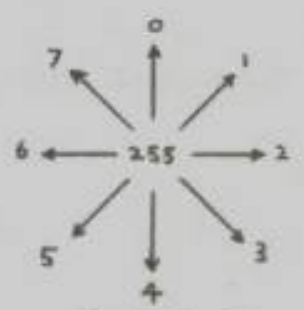
STRIG n_1 --- n

Reads trigger button of stick n1. n2=0 when depressed.

STICK n_1 --- n_2

Reads joystick n1. Result n2 is the same as in ATARI-BASIC. If 4 is added to the joystick number n_1 then result n_2 is compatible with the PMOVE/MMOVE commands.

$n_1 = 0-3$



$n_1 = 4-7$

GRAPHICS

The following commands operate on "playfield" graphics. The action of each word is identical to the equivalent ATARI BASIC command.

GRAPHICS n_1 ---

Set up for graphics mode nl. This command will access any graphics mode that the ATARI OS will support, i.e. XL machines can access graphics modes 12-15. Unlike ATARI BASIC, changing graphics modes does not interfere with PM-GRAPHICS.

COLOR n_1 ---

Selects color n_1 as the current color for "PLOT" or "DRAW" command.

COL --- addr

Variable, holds current color set by COLOR command.

SETCOLOR n_1 n_2 n_3 ---

n_1 selects color register, n_2 selects colour and n_3 selects luminence.

POSITION X Y ---

Positions cursor at X,Y co-ordinates.

PLOT X Y ---

Plots a single point at X,Y co-ordinates in the colour stored in COL.

DRAW X Y ---

Same as BASIC's DRAWTO command. Draws a simulated straight line from the current cursor position to X,Y co-ordinates. Colour is taken from COL.

LOCATE X Y --- n_1

Reads pixel value at co-ordinate X,Y.

USER-DEFINED CHARACTERS

The following commands enables the user to create user-defined characters without worrying about memory allocation.

USER-CHAR ---

Command to enable the user-defined character option. When executed automatically allocates memory for the new character set within the dictionary and downloads the ATARI char-set from ROM. This command can only be executed once from power-up.

DOWN LOAD ---

Downloads ATARI char-set from ROM. If USER-CHAR is not enabled an error message will be given.

ATARI ---

Tells hardware to use standard character set.

CUSTOM ---

Tells hardware to use custom character set.

DEF-CHAR n_1 ,,, n_8 C ---

Data bytes n_1 to n_8 are used to redefine character number C in the custom character set.

N.B. Words defined before USER-CHAR was enabled should not be forgotten as this may overwrite memory allocated to the custom character set.

PM-GRAPHICS COMMANDS

The following set of definitions removes the tedium normally associated with writing programmes to control the "player-missile" capabilities of the ATARI hardware.

Each player can be up to 8 bits wide by 16 bytes deep (missiles 2 bits x 16 bytes) and "shapes" are stored in shape tables within the dictionary. There are 3 shape tables which are accessible by the user, these are designated as follows.

PSHAPE - primary shape table for players. Whenever a player's shape is moved on the screen it is from this table that the data is taken. There are 4 shapes held in this table, one for each player (numbered 0 to 3). PSHAPE can be loaded directly using the PLOAD definition or from SSHAPE using the S>P definition.

SSHAPE - secondary shape table. This table holds 8 predefined shapes which can be transferred to any primary shape at high speed for animation effects. SSHAPE is loaded by the user with the SLOAD definition.

MSHAPE - missile shape table. Holds data for the 4 missiles (numbered 0 to 3) and is loaded using the MLOAD definition.

When enabling the PM-GRAPHICS option the definition remembers the TV-LINE resolution that the user supplies. This value is used in further definitions to mask the vertical value of players' (or missiles) positions. This is necessary in 2 line resolution to stop players overwriting into other players' screen memory.

All parameters used with PM-GRAPHICS definitions are masked, i.e. selecting player 4 does not cause an error. it simply masks to player 0. Similarly, selecting SSHAPE No. 8 is automatically masked to No. 0. This enables the use of the RND command without any further masking.

PM-GRAPHICS DEFINITIONS

PM-GRAPHICS n_1 ——

Set up player missile graphics with TV-line resolution of n_1 (1 or 2). This command enables PM-GRAPHICS by setting all the relevant hardware registers initialising the variables and allotting the screen memory within the dictionary.

MBASE —— addr

Returns the address of the first memory location of the missile screen RAM.

PBASE n —— addr

Returns the address of the first memory location of player no. n's screen RAM.

MSHAPE —— addr

Returns the address of the missile shape table.

PSHAPE n —— adr

Returns the address of player no. n's shape table.

SSHAPE n —— adr

Returns the address of secondary shape no. n.

CPMR ——

Clears Player-Missile screen memory. Removes all objects from screen but leaves shape tables intact. Objects can be redrawn using PDRAW or MDRAW. Objects will automatically be redrawn when the next movement is executed.

CLRP n ——

Removes player no. n from screen.

CLRM n ——

Removes missile no. n from screen.

PLOAD n_1 ,,, nx A P ---

Clears player no. P's primary shape and reloads with data bytes n_1 to nx. A is the number of arguments on the stack and can be any number up to 16. PLOAD will update screen memory with the new shape.

PDRAW n ---

Draws player no. n on the screen at the current X,Y co-ordinates. Shape data is taken from PSHAPE.

MLOAD n_1 ,,, nx A M ---

Clears missile no. M's shape table and reloads with data bytes n_1 to nx. A is the number of arguments on the stack (up to 16). MLOAD automatically updates screen memory with the new shape. Missiles may only be 2 bits wide.

MDRAW n ---

Draws missile no. n on the screen at the current X,Y co-ordinates. Shape data is taken from MSHAPE.

SLOAD n_1 ,,, nx A S ---

Clears secondary shape no. S and reloads with data bytes n_1 to nx. A is the number of arguments on the stack (up to 16). SLOAD stores the data into SSHAPE. This table holds up to 8 shapes (S can be numbered 0 to 7).

S>P S P ---

Move secondary shape S (0 to 7) to primary shape P (0 to 3). This command does not update screen memory. The screen will automatically be updated during the next PMOVE or PPLOT command or the user may update immediately with the PDRAW command.

PSIZE P n —

Set player no. P's size to n. Values for n are:
0 = normal, 1 = 2 x width, 2 = normal, 3 = 4 x width.

MSIZE M n —

Same as PSIZE but for missiles.

PCOL P C L —

Set player no P's color to C and luminence L. Color values are the same as SETCOLOR.

PPH X P —

Plot player P at horizontal position X. This command moves the player on the screen but does not update the shape from PSHAPE.

PPV Y P —

Plot player P at vertical position Y. This command clears the shape from the screen and redraws using data from PSHAPE.

MPH P n —

Move player P, n pixels to the right (left if n is negative). This command does not update screen memory.

MPV P n —

Move player P, n pixels up the screen (down if n is negative). This command will update screen memory from PSHAPE.

PMH X M —

Same as PPH but for missiles.

PMV Y M —

Same as PPV but for missiles. Screen is updated from MSHAPE.

MMH M X —

Same as MPH but for missiles.

MMV M Y —

Same as MPV but for missiles.

PPLOT P X Y ---

Plots player no. P at X,Y co-ordinates. Screen memory is automatically updated from PSHAPE.

MPLOT M X Y —

Plots missile no. M at X,Y co-ordinates. Screen memory is automatically updated from MSHAPE.

PMOVE P O D —

Moves player no. P, O pixels in direction D. Screen memory is updated from PSHAPE.

MMOVE M O D —

Moves missiles no. M, O pixels in direction D. Screen memory is updated from MSHAPE.

For both of the MOVE commands the directions are shown below.

MX@ n_1 — n_2

Leaves current X position of missile no. n_1.

MY@ n_1 — n_2

Leaves current Y position of missile no. n_1.

PX@ n_1 — n_2

Leaves current X position of player no. n_1.

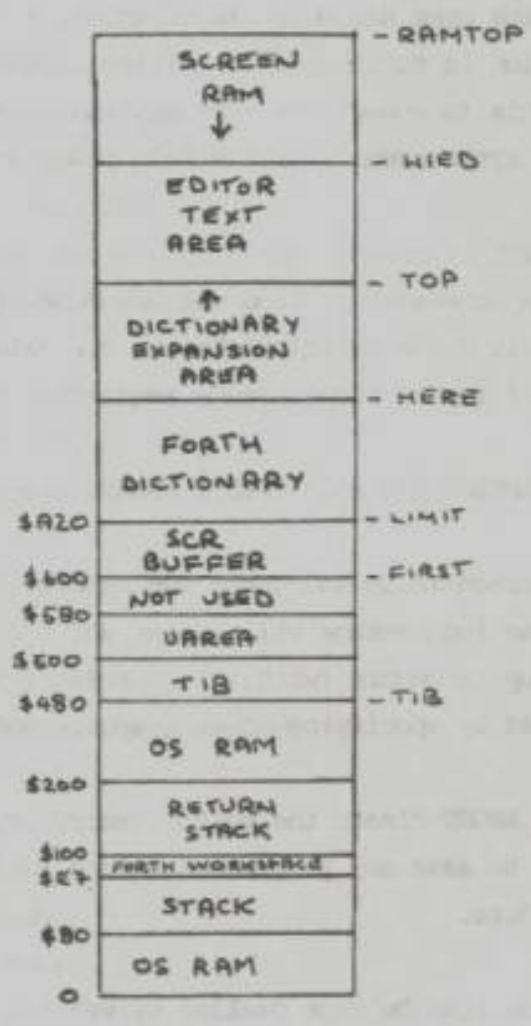
PY@ n_1 — n_2

Leaves current Y position of player no. n_1.

APPENDIX A

MEMORY UTILISATION

ES-FORTH MEMORY MAP



MEMORY CONSERVATION

At power up ES-FORTH checks how much memory is available in the system. 8K bytes is subtracted from this value and stored in the variable HIED. This memory is reserved in order to allow full use of all the standard graphics modes. The remaining memory has to be shared between dictionary expansion space and editor text memory.

In 48K systems 28K of memory is left to the user, so it is practical to allocate 10 editing screens and still have 18K of dictionary memory.

In 32K systems the user has only 12K of memory to share out. The normal mode of operation is to allocate 5 editing screens leaving 7K of dictionary memory. This is enough for most applications but it is possible to gain more system memory if the full modes 8-11 graphics are not required.

To do this it is necessary to find out how much graphics memory is required and use this value to calculate the new value for HIED. E.g. if graphics mode 7 is the highest mode required:

```
HEX 7 GRAPHICS 230 @ 2- HIED ! ABORT
```

will calculate the correct value and store the result in HIED. When ABORT executes the greeting message will appear and the user can carry out the normal power up procedure (with an extra 4K of memory). Further savings can be achieved by specifying lower graphics modes.

N.B. Executing ABORT clears the editor memory, so it is necessary to save any programmes before following the procedure.

Editor memory can also be kept smaller by writing programmes in smaller parts and saving them to tape. Each part can then be loaded in and compiled separately.

A further 2K of memory can be gained if the Player-missile graphics commands are not required. This can be achieved by typing:

```
FORGET MBASE
```

All PM-GRAPHICS commands will then be erased from the dictionary. The system will not allow commands below MBASE to be forgotten.

APPENDIX B

ERROR MESSAGES

0	syntax error
1	empty stack
2	insufficient memory for operation
3	incorrect address mode
6	screen number out of range
7	full stack
9	option not enabled
10	option already enabled
17	can only be used in a colon definition
18	can only be executed
19	conditionals not paired. e.g. IF without ENDIF
20	definition not finished
21	in protected dictionary (see FENCE)
22	use only when loading
23	off current editing screen
24	declare vocabulary

Errors greater than 100 refer to the ATARI operating system.

129	IOCB already open
130	non-existent device specified
131	IOCB write only
132	invalid command
133	device not open
134	bad IOCB number
135	IOCB read only
137	truncated record
138	device timeout
139	device not acknowledge
140	serial bus data framing error
141	cursor out of range
142	serial bus data frame overrun
143	serial bus data frame checksum error
145	read after write compare error
146	function not implemented
147	insufficient RAM for selected graphics mode

APPENDIX C

REFERENCES

ATARI 400 Operator's Manual

ATARI 800 Operator's Manual

ATARI 810 Operator's Manual

Starting FORTH by brodie

SYSTEMS GUIDE TO fig-FORTH by ting

BOOT UP LITERALS

The boot up literals are identical to fig-FORTH and the following offsets should be used with the "+ORIGIN" command for access.

+8	N/A
+10	N/A
+12	Name address of top dictionary entry
+14	N/A
+16	Address of user Variable area
+18	Initial top of stack
+20	Initial top of return stack
+22	Terminal I/P buffer address
+24	Name field width
+26	Initial value for WARNING
+28	Initial FENCE address
+30	Initial top of dictionary
+32	Initial Vocabulary link pointer

ADDENDUM

The following general purpose commands are included in ES-FORTH.

MOB addr1 addr2 addr3 ---
"MOVE OBJECT BLOCK" is primarily intended for use with the PM-GRAPHICS commands. When executed the following actions are carried out, erase 16 bytes at address 1, move 16 bytes of data from address 2 to address 3.

U> u1 u2 --- f
Leaves a true flag if the unsigned number u1 is greater than the unsigned number u2. Otherwise a false flag is left.

U< u1 u2 --- f
Leaves a true flag if the signed result of u1-u2 is less than zero. Otherwise a false flag is left.

ES-FORTH

SYSTEM GUIDE

CASSETTE VERSION C1.1

FOR ATARI 32K COMPUTER SYSTEMS

BY

A. KOZAKIEWICZ

Original documentation provided through the courtesy of the FORTH INTEREST GROUP, P.O. Box 1105, San Carlos, CA 94070.

THE ENGLISH SOFTWARE COMPANY, P.O. BOX 43, MANCHESTER M60 3AD.
Telephone 061-835-1358

ES-FORTH

SYSTEM GUIDE

CASSETTE VERSION C1.1

FOR ATARI 32K COMPUTER SYSTEMS

THE ENGLISH SOFTWARE COMPANY

BY

A.KOZAKEWYCZ

Original documentation provided through the courtesy of the FORTH INTEREST GROUP, P.O. BOX 1105, San Carlos, CA 94070.

THE ENGLISH SOFTWARE COMPANY, P.O. BOX 43,MANCHESTER M60 3AD.
Telephone 061-835-1358.

Anyone who has programmed in FORTH should have no trouble with ES-FORTH. The basic dictionary is compatible with fig-FORTH so all the information needed to write programmes is available from the number of books and articles available on the subject.

The following set of notes are relevant to ES-FORTH.

STACK PARAMETERS

Inspection of the glossary will show that most commands expect to find numbers on the computation stack (also known as parameter stack). All of these commands when executed will remove the specified numbers leaving the parameters that are shown to the right of the execution point (three dashes).

The computation stack is capable of holding up to 52 numbers consecutively. There is a 4 number under-run capability, if this is exceeded then a crash is probable.

INTERPRETATION OF TEXT

When text is entered (from either keyboard or editing screen) a search of the dictionary is first made to find a matching entry. If a match is found then the text is a command and the word is executed. Unless the executed word determines otherwise, FORTH will continue interpretation with the next item of text.

If no match is found then FORTH tries to convert the text into a number. If this is possible then the number is pushed to the stack, control is then passed back for interpretation of the next item of text. If convertion is not possible then the text is rubbish and an appropriate error message is given.

After any error control is passed through "QUIT" to the keyboard.

KEYBOARD ENTRY

Commands typed in from the keyboard will execute immediately when the return key is pressed. Keyboard entry can execute any number of commands on the same logical line provided each command is seperated by a space. A logical line is defined as 3 TV lines of text (up to 120 characters). It is up to the user to make sure that the correct parameters are on the stack prior to executing each command.

DEFERRED ENTRY

Like BASIC, commands can be saved in memory to be executed at a later time. These commands are saved into editing screens (SCR's). To do this the SCR must first be selected using a "LIST" or "CLEAR" command. Text can be saved into any line within the selected SCR using the "P" command.

Note although a logical line is 120 char's in length, only the first 64 char's after the P command will be stored into the selected line. The rest of the line is ignored.

LOADING FROM A SCREEN

FORTH can be ordered to execute a series of commands stored in a SCR using the "LOAD" command. Commands are interpreted in exactly the same way as if entered from the keyboard.

Loading can be stopped anywhere on the SCR by including the "¡S" command at the desired point. This returns control to the keyboard. Loading automatically stops when the end of the SCR is met.

If more than one SCR is used for a programme then they can be linked together using the "-->" command. When this is encountered during loading, control is passed to the next sequential SCR.

USING "LOAD"

The "LOAD" command can be stored into a SCR to create nested loads. This can be useful as it gives a kind of macro capability. Standard sets of commands can be stored on tape or disc and re-used in different applications. Multiple loads can be nested as deep as the return stack of the microprocessor will allow.

CREATING NEW DICTIONARY ENTRIES

There are three main kinds of dictionary entry available to the user, variables, constants and colon definitions.

Variables and constants are created as follows,

```
0 VARIABLE cccc
0 CONSTANT cccc
```

The number before VARIABLE is the initial value that the variable will be set to when created. In some versions of FORTH this value is not specified, but it is necessary to include it in ES-FORTH. The number before CONSTANT is the value that will be pushed to the stack whenever the constant name is executed.

When creating colon definitions you are not limited to 1 logical line of text. Hitting return will compile the complete line and then wait for more text to be input. Compilation will continue until the ";" command is met. This will terminate the definition. During this time the "OK" prompt is suppressed.

If an error occurs during compilation then an error message will be printed and control passed back to the keyboard. This will leave the definition unterminated and "smudged" (the definition is present but not executable). Obviously this is a waste of memory so the best action is to forget the definition. To do this first type "SMUDGE", this will toggle the smudge bit in the definition header enabling the entry to be found during a search. Then use the "FORGET" command to remove the entry from the dictionary.

SOUND

The "SOUND" command is included for compatibility with BASIC.

The "PLAY" command has been added to allow sounds to be cancelled after a period of time. The time element is user selectable and does not stop processing during play time.

NB. this command is interrupt driven using the VBI deferred vector. This should be taken into account when linking machine code.

The following example shows the play command driving all 4 channels.

```
HEX
: NOTES
    BEGIN
       BEGIN 2FC C@ FF < UNTIL
       RND 2FC C@ A 8 15 PLAY
       FF 2FC C!
    ?TERMINAL UNTIL ;
DECIMAL
```

When "NOTES" is executed pressing any key will generated a tone from one of the channels selected randomly. Each tone will play for 300 mS unless another key selects that channel.

CONTROLLERS

All the controller commands available in BASIC are also available in ES-FORTH. They all work the same except "STICK", this has an additional option that allows direct compatibility with the PM-GRAPHICS move commands.

USER DEFINED CHARACTERS

User defined characters are obtained very easily in ES-FORTH with the "USER-CHAR" command. No calculations are necessary as the commands automatically decide where in memory the new set will reside. The old set is downloaded and the hardware switched to use the new set. If a large number of characters are to be redefined then it would be a good idea to change "DEF-CHAR" to something a bit easier to type.

e.g. `: DC DEF-CHAR ;`

When using DEF-CHAR make sure the data bytes are entered in reverse order (the byte on top of the stack is the top of the character).

The following example redefines the !, " and # char's.

```
USER-CHAR
1 2 3 4 5 6 7 8 1 DEF-CHAR
66 102 60 189 255 90 124 24 2 DEF-CHAR
2 4 255 196 159 4 31 0 3 DEF-CHAR
```

To switch to the standard set type "ATARI".
To switch back to the custom set type "CUSTOM"

PLAYER-MISSILE GRAPHICS

The player missile graphics commands are designed to allow easy control of the sprite capabilities of the hardware.

The "PM-GRAPHICS" command sets up the variables, hardware and screen memory within the dictionary. Both 1 and 2 line resolution are catered for and this is automatically taken into account whenever plotting or moving objects around the screen.

All parameters used with the commands are masked. So using silly parameters will not cause a crash (as long as there are the correct number of parameters).

If the "PM-GRAPHICS" command is used more than once then it will allocate a completely new area of screen memory within the dictionary. So it is advisable to compile a null definition before the command is executed (e.g. : TASK ;) to enable the old screen memory to be forgotten. When using "PLOAD", "SLOAD" or "MLOAD" the data bytes should be entered in reverse order (the top data byte on the stack is the top of the shape formed).

SPEED CONSIDERATION

The PM-GRAPHICS plot and move commands are nested in levels of complexity. This means that the higher level commands tend to use the lower level commands. Obviously this means that some commands will operate faster than others. The following table shows the commands in order of level (speed of operation),

PLAYERS :- PPH , MPH , PPV , MPV , PPLOT , PMOVE
MISSILES:- PMH , MMH , PMV , MMV , MPLOT , MMOVE

Due to the extra calculations necessary (4 times as many) the missile commands will operate slower than the player's.

Shapes can appear to move faster by specifying larger offsets in the move commands however this may also give a jittery effect if the offset is to large.

PM EXTENSIONS

The following extensions can be added to provide collision detection for the PM-GRAPHICS. All of the words are called in the form,

P(orX) --- b

Result b is a bit pattern dependent on the type of collision. Only the lower 4 bits are used, the least significant bit refering to player, missile or playfield #0, bit 1 to player #1 etc.

e.g. if result=1 when "CMPF" is executed then a collision has occured between the selected missile and playfield 1.

The command "PRIOR" enables access to the priority register. The user should refer to the ATARI technical notes for priority selection as the actual values to be stored can be quite complicated.

The command "HITCLR" should be executed after detection of collisions to reset the flags.

```
SCR# 1
 0 ( PM-GRAPHICS COLLISION DETECTION )
 1 FORTH DEFINITIONS HEX
 2 : CMPF ( missile to playfield )
 3       D000 + C@ ;
 4 : CPPF ( player to playfield )
 5       D004 + C@ ;
 6 : CPPL ( player to player )
 7       D008 + C@ ;
 9 : CMPL ( missile to player )
10      D00C + C@ ;
11 : HITCLR ( clears all collision flags )
12      0 D01E C! ;
13 : PRIOR ( stores into priority register )
14      26F C! ;
15 DECIMAL ;S

SCR# 2
 0 ( PM-EXTENSIONS #2 )
 1 FORTH DEFINITONS DECIMAL
 2 0 VARIABLE ESHAPE 126 ALLOT
 3
 4 : ELOAD 7 AND 16 * ESHAPE +
 5         >R R 16 ERASE
 6         R + R> DO I C! LOOP ;
 7
 8 : E>P PSHAPE SWAP 7 AND
 9       16 * ESHAPE +
10       OVER MOB ;
11
12
13
14
15
```

SCR# 2 will build another secondary shape table within the dictionary which can be used in exactly the same way as "SSHAPE". The table "ESHAPE" will hold another 8 predifined shapes and can be loaded using "ELOAD" as follows,

```
n1 , , , nx A E ELOAD
```

where n1 to nx are the data bytes, A is the number of arguments and E is the shape number. Any shape can be loaded to any primary shape using the "E>P" definition in exactly the same way as "S>P".

PM-EXAMPLES

The following examples demonstrate the use of the player-missile commands. Each SCR will load and execute immediately and can be stopped using the break key. After "breaking" type "FORGET TASK" before loading the next SCR (avoids reallocation of screen memory).

```
SCR# 1
 0 ( PM-GRAPHICS DEMO #1 )
 1 FORTH DEFINITIONS DECIMAL
 2 : TASK ;
 3 : DEMO1
 4   1 2 4 8 16 32 64 7 0 SLOAD
 5   8 8 8 8 8 8 8 7 1 SLOAD
 6   64 32 16 8 4 2 1 7 2 SLOAD
 7   0 0 0 127 0 0 0 7 3 SLOAD
 8   0 5 5 PCOL 0 100 60 PPLOT
 9   0 BEGIN 1+ DUP 3 AND 0 S>P
10     0 PDRAW 0 2 MPH 4 WAIT
11    ?TERMINAL UNTIL ;
12 1 GRAPHICS 1 PM-GRAPHICS DEMO1
13  ;S
14
15


SCR# 2
 0 ( PM-GRAPHICS DEMO # 2 )
 1 FORTH DEFINITIONS DECIMAL
 2 : TASK ;
 3 : PDAT 1 2 3 4 5 5 ;
 4 : MDAT 1 2 1 2 1 5 ;
 5 : DEMO2 4 0 DO PDAT I PLOAD
 6       MDAT I MLOAD
 7       I 120 120 PPLOT I 120 120 MPLOT
 8       I I 3 * 4 PCOL LOOP
 9    BEGIN 4 0 DO
10    I 2 I PMOVE
11    I 2 I 4 + MMOVE LOOP
12   ?TERMINAL UNTIL ;
13
14  1 GRAPHICS 1 PM-GRAPHICS
15  DEMO2
```

```
SCR #3
 0 ( PM-GRAPHICS DEMO # 3 )
 1 FORTH DEFINITIONS DECIMAL
 2 : TASK ;
 3 : DEMO3 1 2 3 4 5 5 1 PLOAD
 4    1 5 5 PCOL 1 100 70 PPLOT
 5    BEGIN 4 STICK DUP 255 <
 6     IF 1 1 ROT PMOVE
 7      ELSE DROP
 8     ENDIF
 9    ?TERMINAL UNTIL ;
10
11  1 GRAPHICS 1 PM-GRAPHICS
12  DEMO3
13 ;S
14
15
```

SCR #1 when loaded shows the use of the secondary shape table to create the effect of a rod walking across the screen. The animation is considerably slowed down with the "WAIT" command in order to show each shape appearing.

SCR #2 moves all four players and missiles in different directions at the same time.

SCR #3 demonstrates the use of the "STICK" command in conjunction with the move commands. The player on the screen is controlled with the joystick in port 1.

SIMPLE UNCOMPILER

An uncompiler is often a useful aid when debugging applications. The following screens provide the word,

```
UNCOMPILE cccc
```

where " cccc " is the dictionary entry to be uncompiled.

When executed the definition reports whether the word is a variable,constant, USER-variable or colon definition. If the word is a variable or constant then the current value is displayed.If the word is a colon defintion then the parameter field is uncompiled

```
SCR# 1
 0 ( UNCOMPILER 1)
 1 FORTH DEFINITIONS HEX
 2 : GC DUP 2- @ ;
 3 : SHOW BEGIN DUP @ 2+
 4   DUP NFA ID. DUP ' CLIT = IF >R
 5   2+ DUP C@ . 1- ELSE DUP ' (LOOP)
 6   = OVER ' (+LOOP) = OR OVER ' LIT
 7   = OR OVER ' BRANCH = OR OVER
 8   ' OBRANCH = OR IF >R 2+ DUP ?
 9   ELSE DUP ' (.") =
10 IF >R DUP 2+ DUP 1+ SWAP C@ 1F AND
11 DUP ROT ROT TYPE 22 EMIT SPACE +
12 1+ ELSE >R
13 THEN THEN THEN 2+ R> DUP ' (;CODE)
14 = SWAP ' ;S = OR UNTIL DROP ;
15 -->
```

```
SCR# 2
 0 ( UNCOMPILER 2)
 1 : UNCOMPILE CR [COMPILE] ' DUP
 2   NFA C@ 40 AND >R DUP 6C = IF
 3   ." code " DROP ELSE
 4   6C [ -FIND C/L DROP DROP 2- @ ]
 5   LITERAL = IF ." constant " ? ELSE
 6   6C [ -FIND TOP DROP DROP 2- @ ]
 7   LITERAL = IF ." variable " ? ELSE
 8   6C [ -FIND BLOCK DROP DROP 2- @ ]
 9   LITERAL = IF SHOW ELSE
10  6C [ -FIND SCR DROP DROP 2- @ ]
11  LITERAL = IF ." USER-variable "
12  DROP ELSE ." can't uncompile "
13  DROP THEN THEN THEN THEN THEN
14  R> @ > IF ." IMMEDIATE" THEN CR ;
15 DECIMAL ;S
```

USEFUL VECTORS

Advanced users may wish to create their own machine code definitions. These definitions must terminate by jumping to one of the following vectors,

```
NEXT   =$0A86 -Address of the inner interpreter. All definitions must
               pass through this vector.
PUSH0A =$0DC7 -Pushes the value in the accumulator to the computation
               stack (high byte set to zero) then jumps to NEXT.
PUSH   =$0A7F -Pushes a number to the computation stack and then exec-
               utes NEXT (high byte in accumulator, low byte on return
               stack).
POP    =$0B98 -Drops one number from the computation stack then exec-
               utes NEXT.
POPTWO =$0B96 -Drops two numbers from the computation stack then exec-
               utes NEXT.
PUT    =$0A81 -Replaces last number on stack then executes NEXT (high
               byte in accumulator, low byte on return stack).
```

The following vectors will be useful.

```
XSAVE =$FE    -Temporary storage location for X register. As X is used
               as the computation stack pointer, it must be saved
               before using it for other purposes and then restored
               before vectoring to NEXT.
IP    =$F7    -Address of the interpretation pointer.
W     =$FA    -Address of the code field pointer.
N     =$EF    -Address of 8 byte scratch area in zero page ram.
UP    =$FC    -Address of user pointer.
SETUP =$0AAE  -Subroutine that moves 16 bit items on the computation
               stack to scratch area (N). Number of items to move is
               in accumulator.
```

THE fig-FORTH MODEL

A copy of the fig-FORTH model is included in this manual. By looking at each definition the user should be able to see how FORTH operates and what actions are carried out during normal operation.

In general ES-FORTH will operate the same as fig-FORTH however the internal composition of words are not necessarily the same. This is because ES-FORTH contains a lot more machine code than fig-FORTH and is optimised for both speed and efficiency of code. Also many more functions and enhancements have been added to take direct advantage of the unique abilities of the ATARI computer systems.

```
 0 # 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
```

```
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 0BRANCH            ( 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 = 1 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.

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


SCR # 7
  0 (  INPUT-OUTPUT,  APPLE                                  WFR-780730 )
  1 CODE HOME   FC58 JSR,  NEXT JMP,
  2 CODE SCROLL   FC70 JSR,  NEXT JMP,
  3
  4 HERE  '  KEY  2  -  !    ( POINT KEY TO HERE )
  5    FDOC JSR,  7F # AND,  PUSHOA JMP,
  6 HERE  ' EMIT  2  -  !   (  POINT EMIT TO HERE  )
  7    BOT 1+ LDA,  80 # ORA,  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,  PUSHOA JMP,
 14
 15 DECIMAL     ;S


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

FORTH INTEREST GROUP                                  MAY 1, 1979
```

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

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

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

FORTH INTEREST GROUP                                  MAY 1, 1979
```

```
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 CHART )  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 WC )
 10            TYA,  BOT 4 + CMP,  0=
 11          IF, ( T-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 WC )  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 DIS 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 -->

FORTH INTEREST GROUP                                  MAY 1, 1979
```

```
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,                  -->

FORTH INTEREST GROUP                                  MAY 1, 1979
```

```
SCR # 30
  0 (  STACK MANIPULATION                                    WFR-79MAR2( )
  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

FORTH INTEREST GROUP                                  MAY 1, 1979
```

```
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

FORTH INTEREST GROUP                                    MAY 1, 1979
```

```
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  OE  +ORIGIN ( BS )  @  -
  4     IF  DROP  OR  OVER  1  -  DUP  R>  2  -  +  >R  -
  5        ELSE ( NOT BS )  DUP  OD  -
  6             IF ( RET ) LEAVE  DROP  BL  0  ELSE  DUP  ENDIF
  7             I  CI  0  I  1+  !
  8        ENDIF EMIT  LOOP  DROP  ;
  9 : QUERY     TIB  @  50  EXPECT  0  IN  !  ;
 10 FORGET  HERE
 11 : X   BLK @                                  ( END-OF-TEXT IS NULL *)
 12       IF ( DISC ) 1 BLK +!  0 IN !  BLK @  7  AND  0=
 13          IF ( SCR END )  TEXEC  R>  DROP  ENDIF ( disc dependent )
 14        ELSE  ( TERMINAL )    R>  DROP
 15          ENDIF  ;   I   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  CI  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  @  CI  ;
 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  CI       ( LENGTH STORED FIRST )
  9    +  HERE  1+
 10    R>  CMOVE  ;      ( MOVE STRING FROM BUFFER TO HERE+1 )
 11
 12
 13
 14
 15 -->

FORTH INTEREST GROUP                                    MAY 1, 1979
```

```
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  <  ?  ?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                DR0
  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                                  DJE-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 LARGER OF TWO NUMBERS *)
 15       OVER  OVER  <  IF  SWAP  ENDIF  DROP  ; -->

FORTH INTEREST GROUP                                    MAY 1, 1979
```

```
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                                                    WFR-79APR02 )
  0 (  BLOCK
  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

FORTH INTEREST GROUP                                    MAY 1, 1979
```

```
SCR # 63
  0 (  INSTALLATION DEPENDENT TE) .INAL 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       XS/ /E LDX,   PUSHOA JMP,      -->

SCR # 64
  0 (  INSTALLATION DEPENDENT TERMINAL I-O,  TIM      WFR-79APR02 )
  1
  2 ( ?TERMINAL )
  3      HERE  -2  BYTE.IN  ?TERMINAL  !    ( VECTOR LIKEWISE )
  4      1 # LDA, 6E02 BIT, 0= NOT  IF,
  5      BEGIN,  731D JSR,  6E02 BIT,  0= END,  INY,  THEN,
  6       TYA,    PUSHOA JMP,
  7
  8 ( CR )
  9     HERE  -2  BYTE.IN  CR  !  ( VECTOR CRS' CF TO HERE )
 10     XSAVE STX,  728A JSR,  XSAVE LDX, NEXT JMP,
 11
 12 -->
 13
 14
 15

SCR # 65
  0 (  INSTALLATION DEPENDENT DISC                  WFR-79APR02 )
  1 6900    CONSTANT  DATA                    ( CONTROLLER PORT *)
  2 6901    CONSTANT  STATUS                  ( CONTROLLER PORT *)
  3
  4
  5 : #HL             ( CONVERT DECIMAL DIGIT FOR DISC CONTROLLER *)
  6       0  0A  U/  SWAP  30  +  HOLD  ;
  7
  8 -->
  9
 10
 11
 12
 13
 14
 15

FORTH INTEREST GROUP                              MAY 1, 1979
```

```
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 - 1 TO READ, 0 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 ( 1-READ)  ELSE  4F ( 0-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  DRO
 13 2C  BYTE.IN  BUFFER      REPLACED.BY  R/W
 14 30  BYTE.IN  BLOCK       REPLACED.BY  R/W      DECIMAL  ;S
 15
```

```
SCR # 72                                                DJK-WFR-79DEC02 )
  0 (  '  FORGET,
  1 :  '          ( FIND NEXT WORDS PFA; COMPILE IT, IF COMPILING *)
  2     -FIND 0= 0 ?ERROR DROP [COMPILE] LITERAL ;
  3                                  IMMEDIATE
  4 HEX
  5 : FORGET                        ( Dave Kilbridge's Smart Forget )
  6     [COMPILE] ' NFA   DUP FENCE @ U< 15 ?ERROR
  7     >R VOC-LINK @ ( start with latest vocabulary )
  8   BEGIN R OVER U< WHILE [COMPILE] FORTH DEFINITIONS
  9       @ DUP VOC-LINK ! REPEAT ( unlink from voc list )
 10   BEGIN DUP 4 -     ( start with phantom nfa )
 11       BEGIN PFA LFA @ DUP R U< UNTIL
 12       OVER 2 - 1 @ -DUP 0= UNTIL ( end of list ? )
 13     R> DP ! ;    -->
 14
 15

SCR # 73
  0 (  CONDITIONAL COMPILER, PER SHIRA                  WFR-79APR01 )
  1 : BACK     HERE - , ;              ( RESOLVE BACKWARD BRANCH *)
  2
  3 : BEGIN    ?COMP  HERE  1  ;                    IMMEDIATE
  4
  5 : ENDIF    ?COMP 2 ?PAIRS  HERE  OVER  -  SWAP  !  ;  IMMEDIATE
  6
  7 : THEN     [COMPILE]  ENDIF  ;    IMMEDIATE
  8
  9 : DO       COMPILE  (DO)  HERE  3  ;              IMMEDIATE
 10
 11 : LOOP     3  ?PAIRS  COMPILE  (LOOP)  BACK  ;  IMMEDIATE
 12
 13 : +LOOP    3  ?PAIRS  COMPILE  (+LOOP)  BACK  ;      IMMEDIATE
 14
 15 : UNTIL    1  ?PAIRS  COMPILE  OBRANCH  BACK  ; IMMEDIATE  -->

SCR # 74
  0 (  CONDITIONAL COMPILER                             WFR-79APR01 )
  1 : END      [COMPILE]  UNTIL  ;  IMMEDIATE
  2
  3 : AGAIN    1  ?PAIRS  COMPILE  BRANCH   BACK  ;   IMMEDIATE
  4
  5 : REPEAT   >R  >R  [COMPILE]  AGAIN
  6              R>  R>  2  -  [COMPILE]  ENDIF  ;  IMMEDIATE
  7
  8 : IF       COMPILE  OBRANCH   HERE  0  ,  2  ;  IMMEDIATE
  9
 10 : ELSE     2  ?PAIRS  COMPILE  BRANCH  HERE  0  ,
 11            SWAP  2  [COMPILE]  ENDIF  2  ;       IMMEDIATE
 12
 13 : WHILE    [COMPILE]  IF  2+  ;    IMMEDIATE
 14
 15 -->

FORTH INTEREST GROUP                                    MAY 1, 1979
```

```
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

FORTH INTEREST GROUP                                    MAY 1, 1979
```

E S FORTH

32K Cassette/Disk

Instructions for ATARI version:

LOADING:

Wind cassette forward one inch past the leader tape!

Cassette: Empty cartridge slot. Switch computer on whilst holding down the **START** button. On the XL models, hold down the **OPTION** key also. Press play, then press **RETURN.**

Disk: Empty cartridge slot. Switch computer on. On the XL models, hold down the **OPTION** key also. Disk will load automatically.

SPECIAL FEATURES OF E S FORTH:

Basic dictionary compatible with **FIG-FORTH.**

Many general purpose extensions.

Full screen editor.

Complete set of I/O extensions.

Built-in editor definitions.

Can also be used with **ATARI ASSEMBLER EDITOR** for debugging and machine code linking.

Built-in sound commands.

ATARI BASIC compatible graphics commands.

Automatic control of **ATARI PLAYER-MISSILE** graphics.

Complete set of controller commands.

User selectable number of editing screens.

FOR FULL DETAILS OF E S FORTH, REFER TO THE DETAILED USER MANUAL SUPPLIED WITH THIS PROGRAM.