FORTH Workstation for the 1802

DRAFT RELEASE 2025/09/09

Contents

Introduction

Why another Forth implementation for the RCA1802?

There are several very good published versions of Forth for the RCA1802. However, they mostly implement core Forth words from the figFORTH model. Which then essentially turns an 1802 system into a kind of RPN programmable calculator. Lacking disk storage or any means of writing, compiling, running and saving code in a reusable manner they tend to get implemented (a fun project) and then ignored.

Meanwhile. sometime back in the early 1980's I decided I wanted more. So I pulled together various publicly available bits of code, wrote some more of my own, and built a true multi-tasking Forth based 1802 workstation. And rather that relying on compiling basic functionality from source every time you want to use it, I integrated an editor, assembler, and multitasker into the base system.

From 1982 to 2001 my 1802 workstation functioned as my developments system and home automation system. Roll the clock forward to the 2020's and that old system is still running (albeit not doing HA anymore) and I'l updated the code to support Q/EF type serial I/O as well as the original CDP1854 UART. Which means I'm not using that system on the two Lee Hart Membership Card's I own – both the early version and later version with 7 segment LEDS.

So, in conclusion, I've released this code to capture my 40-year project and share. Hopefully somebody else might enjoy it, or at least pull ideas and code for their project.

Scope: What's Included?

Off the top of my head, I've added the following to the original fig-FORTH release:

1. Console code supporting **19200 baud Q&EF serial I/O**. With ?TERMINAL support integrated into the inner interpreter to allow "BREAK" functionality when words like VLIST or DUMP are outputting long listing.
2. Console code supporting an interrupt driven **CDP1854 UART**
3. **RAM disk** I/O - as many 1K FORTH screens as you have free memory - possibly battery backed / EPROM / EEPROM as required.
4. **Line Editor** - pre-assembled as words in a FORTH dictionary for editing RAM disk FORTH "screens" (see above)
5. **1802 Assembler** - pre-assembled as words in a FORTH dictionary to allow adding inline 1802 code to other Forth words
6. **ROM-able** – generic 1802 figFORTH will run from ROM but would not work with multiple vocabularies. That has been fixed.
7. Support for **loading and storing Forth screens** or random memory blocks over the serial port as either Forth formatted text files or Intel Hex format raw data files.
8. **Multitasking** - cooperative multitasking support for concurrent tasks - task 1 runs the console (works best in a fully interrupt driver system but work well on a simple ELF)
9. **Configurable build modes** for different hardware and software combinations.
10. **1802 specific words** for controlling Q, testing reading the EF lines, and doing N-line port I/O
11. Built in **ASCII text error messages** (from the original screen 4 in figFORTH).
12. **Pre-configured versions** can be built for a simple Q/EF serial I/O Elf, an 1802 system with a 1854 UART (interrupts supported), or both early and recent versions of Lee Hart's Membership Card (with or without interrupt support).
13. **RTC support** - interrupt driven Motorola MC146818
14. Support for a few other **hardware specific things** like an ADC0808 A/D converter, intel 8255 I/O port, eight digit 7-segment display, and a single line alpha-numeric LCD display.
15. **Example screens** showing words written in Forth assembler, and words using the multitasking
16. **Autoload** screens on boot option via dip switch selection.
17. .... and probably some **other goodies**.

History

In case you are interested, I started this project back in 1982. Step one was hand typing into PDP11 system a photocopy of a photocopy of the figFORTH reference implementation for the 1802. Then I created hex files to burn to EPROM using an 1802 assembler for the PDP11 written by Wayne Bowdisk (hi Wayne wherever you are). I also wrote an 1802 simulator in PDP-11 FORTRAN to facilitate debugging of the code. And then I wrote about all of this in Ipso Facto issue #29 (May 1982).

link > IPSO FACTO Issue #29

There were several follow-on articles (email me for copies if these links are dead):

link > IPSO FACTO Issue #31
link > IPSO FACTO Issue #32
link > IPSO FACTO Issue #34
link > IPSO FACTO Issue #35
link > IPSO FACTO Issue #37
link > IPSO FACTO Issue #38

At one time my system worked with the A.C.E. WD1771 controller board running dual 8" floppy disk drives. I converted to battery backed RAM in the mid-80's and sadly those drives, and the supporting code, are long gone.

In the meantime, while I've added a few things to the A.C.E. release of fig-FORTH over the years, the original version of the source code file is long lost. So back during the Covid19 lockdown, I recreated the current version recreated by reverse assembly of the ACE EPROM. Of course I used tools written in 1802 figFORTH to disassemble itself. And having played with two versions of Lee Hart's Membership Card (link> https://www.sunrise-ev.com/1802.htm) I naturally ported the code to those.

Which brings us to 2025 where I've collected all the bits and pieces and create a single source file with configurable build options for console and timer functionality, as well as several memory models and a few special features.

Implementation

Pretty much every version of Forth for the 1802 requires a console, which in turn requires the definition of three words

- KEY : a word to get a single character from the console,
- EMIT : a word to output a single character to the console,
- ?TERMINAL ; a word to indicate a console key has been pressed

In ACE1802FORTH the labels for these words are getKEY, CSEND and qTERM. Sample implementations have been provided for using software serial I/O via Q & EF3, as well as hardware UART I/O via an interrupt driven CDP1854 UART. Modify you implementation using those routines as a guide.

In addition to the console words, figFORTH has example words meant to be used for interface to two floppy disk drives. ACE1802FORTH originally supported two 8” drives via a WD1771 chip but has been modified to use RAM disk only. To make that useful, please see the Work Flow section of this document for information about how to load and store programs to and from the RAMDISK via the console (and a suitable terminal emulator program on your PC).

Finally, to support cooperative multitasking ACE1802FORTH needs a “tic timer” – a signal used to indicate when tasks should run and when they should sleep. A tic time is typically created from a periodic hardwire interrupt but ACE1802FORTH also simulates a tic timer in software if no hardware source is available. Naturally timing accuracy suffers when a software timer is used but it still allows cooperative multitasking. In any case, example 1802 interrrupt service routines are provided for a periodic hardware interrupt (like the later versions of Lee Hart’s Membership Card) and a real time clock chip (the Motorola Mc 46818).

Build Options

ACE1802FORTH source code is written to allow various build option so that it supports multiple hardware platforms and different memory configurations. Listed below are the build options, which are located near the start of the source code file.

Build Option	Choices	Description
uart_type	software / hardware	indicates whether the console is connected by a software UART (Q& EF) or a hardware based interrupt driven CDP1854 UART.
timer_type	software / hardware	tic timer implemented in software or hardware
extra_hardware	yes / no	no / yes = include code for extra hardware support ACE CPU systems
example_screens	yes / no	no / yes = include example Forth source screens at block -1
clock_mhz	4 or 1	CPU clock speed for software UART timing (1.8 Mhz or 4 Mhz)
uart_config	$3E	CDP1854 control register : Interrupts enabled, 8 data bits , 2 stop bits , even parity , parity enabled
load_address	0 / 1 / 2	Memory map & start address to be used (see below) 0= $0000 , 1 = $4000 , 2 = $8000
version	6	version will be 6,7, or 8 or depending on load address selected – only appears on the cold/warm start console message
editor	yes / no	1 = include line editor code
assembler	yes / no	include assembler code
stackptr_show	yes / no	show top of stack address as part of the OK prompt
zero_ram	yes / no	include code to set all RAM to $00 at start-up

Memory Map

ORG	$0000	code start
START_OF_RAM	$4000	start of RAM area - must be on a page bountry
END_OF_RAM	$8000	end of RAM block - first byte after
S0_START	END_OF_RAM -$0200	data stack for console task -grows up) variable name = S0
R0_START	END_OF_RAM-$0101	return stack for console task (grows down) variable name = R0
USER_START	END_OF_RAM-$0100	USER area - 64 variables max variable name = UP
TIB_START	END_OF_RAM-$0080	terminal input buffer variable name = TIB
tx_buffer	END_OF_RAM-$0400	256 bytes of RAM used as a hardware UART tx buffers NOTE : buffer must be on page boundaries
rx_buffer	END_OF_RAM-$0300	Reserve 256 bytes of RAM for hardware UART rx buffers NOTE : buffer must be on page boundaries
FIRSTB	END_OF_RAM-$0400	used for RAM disk - address of first disk screen (note #0 is unusable) variable name = FIRST
LIMITB	$FFFF	end of RAM disk area variable name = LIMIT
EXAMPLE_SCREENS	$3800	storage address of example screen must be on 1K boundary
task1stacks	END_OF_RAM-$0780	task 1 stacks
task2stacks	END_OF_RAM-$0700	task 2 stacks
task3stacks	END_OF_RAM-$0680	task 3 stacks
task4stacks	END_OF_RAM-$0600	task 4 stacks
task5stacks	END_OF_RAM-$0580	task 5 stacks
task6stacks	END_OF_RAM-$0500	task 6 stacks
task7stacks	END_OF_RAM-$0480	task 7 stacks

Multitasking

One fun aspect of the ACE1802FORTH workstation is the ability to run multiple tasks simultaneously with the console. Multitasking on the workstation is cooperative – tasks run until they release control to the next task in the queue. A simple "tic timer" mechanism exists to allow tasks to start and stop themselves for defined intervals. The timer runs from either a hardware interrupt or a software simulated interrupt (which is necessarily less accurate).

As implemented, the workstation supports up to eight tasks, but more can be added with a simple edit to the source code. Task 0 is by default the console task, leaving seven tasks for user functions. Each task has its own return stack and computation stack, along with a tic timer used to let it "sleep" periodically.

Multitasking requires only six new Forth words :

1. **START** - used to create a dormant task that will run a single Forth word when started. Usage is : **n START text** where n is the task number (1 – 7) and text is the Forth word for the task,
2. **RUN** – used to start a dormant task. Usage : **n RUN** where n is the task number.
3. **HALT** – used to stop a running task. Usage : **n HALT** where n is the task number
4. **PAUSE** – used within a task to release CPU control until the next pass through the task list.
5. **TIC** – used within a task to release CPU control for "n" tics
6. **TASK#** - used within a task to retrieve its task number. Useful for letting a task HALT itself.

An example task to blink the 1802 Q LED might look like :

```
: QTASK   BEGIN  QON 20 TIC QOFF 40 TIC AGAIN ;
2 START QTASK
2 RUN
```

A Brief Note about Software UARTs

Implementing a UART in software via discrete I/O pins (Q & EF3) creates some unique problems for a multitasking system. Proper timing requires masking interrupts during character transmission and reception. And detecting an incoming character can fail if other task in the system have control of the processor. To that end, for software UART the ACE1802FORTH uses a hybrid console that locks out multitasking while an input line is being received. Once the line has been fully received and is ready to process (i.e. the user pressed carriage return CR) multitasking is reenabled while the input line is processed and/or executed. In addition, the user can enable or disable multitasking by using the words **EI** (enable interrupts) and **DI** (disable interrupts). That can be especially useful when using the Editor when multi-tasking can be distracting. Note: none of this applies to systems using hardware interrupt drive UARTs, which will happily multitask while processing user input from the console.

RAM Disk

Storage of Forth source code “screens” in ACE1802FORTH workstation was originally implemented for two 8” floppy disk drives. Those drives were noisy, bulky, and slow so the system converted to using a “RAM disk”. A RAM disk is simply an area of the microprocessors 64K memory used as storage to simulate a real disk drive. In Forth, disk drives are divided up into “screens”, which were traditionally 1K blocks of storage that could be displayed as sixteen 64-byte lines of ASCII text. So, the ACE1802FORTH RAM drive was simply a mapping of the screen concept onto the processor’s memory. This allowed up to 64 screens, aligned to the start of free memory as screen 1, with some screens obviously pointing to unused memory regions or the code of ACE1802FORTH itself.

Of course, the problem with a RAM drive is that data stored in it is not persistent. When the power to the system is turned off, the data goes away. It is of course possible to use battery backed up memory, NVRAM, or even EPROM / EEPROM for all or part of a RAM drive. In fact, the Forth demo example screens included in ACE1802FORTH are just that – screen data stored with the ROMable Forth code. But for other development purposes, a better way to save and restore Forth source code from the RAM drive is needed.

With ACE1802FORTH, it is assumed that the “console” is typically a PC of some type with its own non-volatile storage that can be used to save & restore Forth screens and so words are available that support that storage. The user has the choice of storing screens as slightly compressed ASCII text files, or embedded in error checking Intel hex formatter file. To save or restore the appropriate word is typed on the console and ACE1802FORTH switches into a send or receive mode. The use then uses the text file transfer capability of the terminal emulator software on their PC to send or receive the selected type of file. (Note : recommended terminal emulator for Windows = TeraTerm and Linux = minicom)

File transfer words :

Word	Stack Diagram	Description
SCR>	n screen# ---	Transmits n screens, starting at screen#, to the console. Lines are truncated at the last non-space character and a CR inserted.
>SCR	screen# ---	Receives incoming characters from the console to screen#. Input text files should be delimited by a ~ (tilde). Lines are padded to 64 characters with spaces if necessary.
IH>	addr n ---	Transmits to the console n bytes of memory data in Intel Hex format from memory starting at addr.
>IH	---	Receives data from the console formatted as Intel Hex and stores in memory address from the intel hex file.

Editor

SCREENS

n	**LIST**	list screen *n*
n	**CLEAR**	clear screen *n*
n1 n2	**COPY**	copy screen *n1* to *n2*
-	**L**	list current screen

CURSOR

-	**TOP**	position cursor at start of screen
n	**M**	move cursor by signed amount *n*
-	**F** ***text***	move ***text*** to PAD and search forward
-	**N**	find next occurrence of ***text*** from by F
-	**B**	back up over ***text*** found by F

LINES

n	**P** ***text***	overwrite line *n* with ***text***
n	**E**	erase line n with blanks
n	**S**	spread at line n. Move lines down

PAD

n	**H**	hold line *n* at PAD
n	**D**	delete line *n* to PAD.
n	**T**	type line *n* and save in PAD
n	**R**	replace line with PAD
n	**I**	insert text from PAD at line *n*.

TEXT EDIT

-	**C** ***text***	spread and copy in ***text*** at cursor
-	**TILL** ***text***	delete from cursor to ***text***
-	**X** ***text***	find and delete ***text***
n	**DELETE**	delete *n* characters before cursor

Assembler

The ACE1802FORTH workstation includes a Forth vocabulary for an 1802 assembler that allows the creation of assembly code words.   Direct input of all 1802 opcodes is support but it's one-pass assembler, meaning forward branches have to be implemented using pseudo operator structures. Supported structures are :

```
BEGIN, …code… AGAIN,
BEGIN, …code… test UNTIL,
BEGIN, …code…  test WHILE,  …code…  REPEAT,
test IF, …code… ELSE, …code… ENDIF,
```

Forth words writing in assembler look like these examples :

```
CODE  QON  SEQ,  NEXT

CODE LEDS   9 INC, 9 LDN, STXD, IRX, 4 OUT, 9 DEC, 9 DEC, 9 DEC, 9 DEC, NEXT

CODE TEST1   Z IF,  SEQ,   ELSE,  REQ,   ENDIF,  NEXT

CODE TEST2   BEGIN,  SEQ, REQ, 0 LDN, Z UNTIL, NEXT

CODE TEST3   BEGIN,  SEQ,  AGAIN,  NEXT

CODE TEST4   BEGIN,  4 LDN,  Z WHILE,  REQ,  REPEAT,  NEXT
```

VOCABULARY WORDS : FORTH

Word	New Word	Type	Stack Diagram	Usage	Description
'	no	Dictionary	--- addr	' 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".
-	no	Math	n1 n2 --- diff	-	Leave 16 bit the difference of n1-n2
-->	no	Compile	---		Continue interpretation with the next disc screen. (pronounced next-screen).
!	no	Memory	n addr ---	-	Store 16 bits of n at address. Pronounced "store".
!CODE	no	Compile	---	-	Used internally to create new word headers
!CSP	no	System	---	-	Save the stack position in CSP. Used as part of the compiler security.
#	no	Compile	d1 --- d2	See #S	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 #>.
#>	no	Compile	d --- addr count	-	Terminates numeric output conversion by dropping d, leaving the text address and character count suitable for TYPE.
#S	no	Compile	d1 --- d2	-	Generates ascii text in the text output buffer, by the use of #, until a zero double number n2 results. Used between <# and #>.
(	no	Compile	---	(comment text)	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.

Word	New Word	Type	Stack Diagram	Usage	Description
(.")	no	Run-time	---	see ."'	The run-time procedure, compiled by ." which transmits the following in-line text to the selected output device.
(;CODE)	no	Run-time	---	see ;CODE	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.
(+LOOP)	no	Run-time	---	see +LOOP	The run-time procedure compiled by +LOOP, which increments the loop index by n and tests for loop completion.
(ABORT)	no	Run-time	---	-	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)	no	Run-time	---	see DO	The run-time procedure compiled by DO which moves the loop control parameters to the return stack.
(FIND)	no	Run-time	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)	no	Run-time	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)	no	Run-time	---	-	The run-time procedure compiled by LOOP which increments the loop index and tests for loop completion
(NUMBER)	no	Compile	d1 addr1 --- d2 addr2	see NUMBER	Convert the ascii text beginning at addr1+l 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.
*****	no	Math	n1 n2 --- prod	-	Leave the signed product of two signed 16 bit numbers

Word	New Word	Type	Stack Diagram	Usage	Description
***/**	no	Math	n1 n2 n3 --- n4	-	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**	no	Math	---	-	Leave the quotient n5 and remainder n4 of the operation : n1*n2/n3 A 31 bit intermediate product is used as for */
,	no	Memory	n --	-	Store n into the next available dictionary memory cell, advancing the dictionary pointer. (pronounced comma)
.	no	Console	n ---	-	Print a number from a signed l6 bit two's complement value, converted according to the numeric BASE. A trailing blanks follows. Pronounced "dot"
."	no	Console	---	." 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	no	Console	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	no	Console	n1 n2 ---	-	Print the number n1 right aligned in a field whose width is n2. No following blank is printed.
.S	yes	Console	---	-	dump contents for stack to console output formatted as 16 bit hex numbers
/	no	Math	n1 n2 --- quot	-	Leave the signed quotient of n1/n2
/MOD	no	Math	n1 n2 --- rem quot	-	Leave the remainder and signed quotient of n1/n2. The remainder has the sign of the dividend

Word	New Word	Type	Stack Diagram	Usage	Description
:	no	Compile	---	: 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.
;	no	Compile	---	-	Terminate a colon-definition and stop further compilation. Compiles the run-time ;S.
;CODE	no	Compile	---	: cccc .... ;CODE	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 procedure given 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 cc prior to ;CODE
;S	no	Compile	---	-	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 procedure
?	no	Console	addr ---	-	Print the value contained at the address in free format according to the current base.
?COMP	no	Console	---	-	Issue error message if not compiling
?CSP	no	Console	---	-	Issue error message if stack position differs from value saved in CSP.
?EF	yes	1802	n --- 0/1	-	Put the state of the 1802 EF pin defined by n on the stack
?ERROR	no	Console	f n --	-	Issue an error message number n, if the boolean flag is true
?EXEC	no	Console	---	-	Issue an error message if not executing
?LOADING	no	Console	---	-	Issue an error message if not loading

Word	New Word	Type	Stack Diagram	Usage	Description
?PAIRS	no	Console	n1 n2 --	-	Issue an error message if n1 does not equal n2. The message indicates that compiled conditionals do not match
?STACK	no	Console	---	-	Issue an error message is the stack is out of bounds. This definition may be installation dependent.
?TERMINAL	no	Console	---	-	Perform a test of the terminal keyboard for actuation of the break key. A true flag indicates actuation. This definition is installation dependent.
@	no	Stack	addr --- n	-	Leave the 16 bit contents of address
[	no	Compile	---	: 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]	no	Compile	---	: xxx [COMPILE] FORTH ;	[COMPILE] will force the compilation of an immediate definitions, that would otherwise execute during compilation. The above example will select the FORTH vocabulary then xxx executes, rather than at compile time.
]	no	Compile	---	See [	Resume compilation, to the completion of a colon-definition
+	no	Math	n1 n2 --- sum	-	Leave the 16 bit sum of n1+n2
+-	no	Math	n1 n2 --- n3	-	Apply the sign of n2 to n1, which is left as n3
+!	no	Math	n addr ---	-	Add n to the value at the address. Pronounced "plus-store".

Word	New Word	Type	Stack Diagram	Usage	Description
+LOOP	no	Execution Flow	n1 --- (run) addr n2 --- (compile)	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 tine error checking.
+ORIGIN	no	Memory	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.
<	no	Math	n1 n2 --- f	-	Leave a true flag if n1 is less than n2; otherwise leave a false flag.
<#	no	Console	---	-	Setup for pictured numeric output formatting using the words: <# # #S SIGN #> The conversion is done on a double number producing text at PAD.

Word	New Word	Type	Stack Diagram	Usage	Description
<BUILDS	no		---	: cccc <BUILDS ... DOES> ... ;	Each time cccc is executed, <BUILDS defines a new word with a high-level execution procedure. Executing cccc in the form sp? 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 runtime procedures to written in high-level rather than in assembler code (as required by ;CODE).
=	no	Math	n1 n2 --- f	-	Leave a true flag if n1=n2 ; otherwise leave a false flag
>	no	Math	n1 n2 --- f	-	Leave a true flag if n1 is greater than n2 ; otherwise a false flag
>IH	yes	Console	addr n ---	-	Receives data from the console formatted as Intel Hex and stores in memory address from the intel hex file.
>R	no	Stack	n ---	-	Remove a number from the computation stack and place as the most accessible on the return stack. Use should be balanced with R> in the same definition.
>SCR	yes	Editor	screen# ---	-	accepts text for Forth screen# from the console. A tilde (~) delimits the start and end of text . Lines on screen are padded out to 64 character with blanks when CR is detected in input stream..
0	no	Math	--- 0	-	puts a zero on top of stack - used so often that is attractive to define as a constant instead of a literal
0<	no	Math	n --- f	-	Leave a true flag if the number is less than zero (negative), otherwise leave a false flag.
0=	no	Math	n --- f	-	Leave a true flag is the number is equal to zero, otherwise leave a false flag

Word	New Word	Type	Stack Diagram	Usage	Description
0BRANCH	no	Execution Flow	f ---	-	The run-time procedure 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	no	Math	--- 1	-	puts a one on top of stack -- used so often that is attractive to define as a constant instead of a literal
1+	no	Math	n --- n+1	-	Increment n by 1
2	no	Math	--- 2	-	puts a two on top of stack -- used so often that is attractive to define as a constant instead of a literal
2.R	yes	Console	n ---	-	outputs lower byte of number on top of stack as two digit hex
2+	no	Math	n --- n+2	-	Increment n by 2
2DUP	yes	Stack	n1 n2 --- n1 n2 n1 n2	-	duplicates top two number on the stack
3	yes	Math	--- 3	-	puts a three on top of stack - used so often that is attractive to define as a constant instead of a literal
4.R	yes	Console	n ---	-	outputs number on top of stack as four digit hex
A2H1	yes	Console	n1 n2 --- b	-	combines two nibbles on TOS into one byte - used by Intel Hex Loader
ABORT	no	Execution Flow	---	-	Clear the stacks and enter the execution state. Return control to the operators terminal, printing a message appropriate to the installation.
ABS	no	Math	n --- u	-	Leave the absolute value of n as u.
AGAIN	no	Execution Flow	addr n --- (compiling)	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.

Word	New Word	Type	Stack Diagram	Usage	Description
ALLOT	no	Memory	n ---	-	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	no	Math	n1 n2 --- n2	-	Leave the bitwise logical and of n1 and n2 as n3.
B/BUF	no	Disk	--- n	-	This constant leaves the number of bytes per disc buffer, the byte count read from disc by BLOCK
B/SCR	no	Disk	--- n	-	This constant leaves the number of blocks per editing screen. By convention, an editing screen is 1O24 bytes organized as 16 lines of 64 characters each.
BACK	no	Compile	addr --	-	Calculate the backward branch offset from HERE to addr and compile into the next available dictionary memory address
BASE	no	Console	--- addr	-	A user variable containing the current number base used for input and output conversion

BEGIN	no	Compile	--- addr n (compiling)	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	no	Console	--- c	-	A constant that leaves the ascii value for "blank".
BLANKS	no	Console	addr count --	-	Fill an area of memory beginning at addr with blanks
BLK	no	Disk	--- addr	-	A user variable containing the block number being interpreted. If zero, input is being taken from the terminal input buffer.

Word	New Word	Type	Stack Diagram	Usage	Description
BLOCK	no	Disk	n --- addr	See also BUFFER, R/W UPDATE, FLUSH	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.
BRANCH	no	Execution Flow	---	-	The run-time procedure to unconditionally branch. An in-line offset is added to the interpretive pointer IP to branch ahead or back. BRANCH is compiled by ELSE, AGAIN, REPEAT.
BYE	no	Execution Flow	---	-	Exits FORTH to an installation dependent address
C!	no	Memory	b addr ---	-	Store 8 bits at address.
C,	no	Memory	b ---	-	Store 8 bits of b into the next available dictionary byte, advancing the dictionary pointer.
C/L	yes	Console	--- n	-	Store the number of characters available per line on the console
C@	no	Memory	addr --- b	-	Leave the 8 bit contents of memory address.
CAPS	yes	Console	--- addr	-	Leave address of caps lock variable
CFA	no	Compile	pfa --- cfa	-	Convert the parameter field address of a definition to its code field address.
CMOVE	no	Memory	from to count --	-	Move the specified quantity of bytes beginning at address from to address to.
COLD	no	Execution Flow	---	-	The cold start procedure to adjust the dictionary pointer to the minimum standard and restart via ABORT. May be called from the terminal to remove application programs and restart.

Word	New Word	Type	Stack Diagram	Usage	Description
COMPILE	no	Compile	---	-	When the word containing COMPILE executes, the execution address of the word following COMPILE is copied (compiled) into the dictionary. This allows specific compilation situations to be handled in addition to simply compiling an execution address (which the interpreter already does).
CONSTANT	no	Compile	n --	n CONSTANT cccc	creates word cccc, with its parameter field containing n. When cccc is later executed, it will push the value of n to the stack
CONTEXT	no	Compile	-- addr U,L0	-	A user variable containing a pointer to the vocabulary within which dictionary searches will first begin
COUNT	no	Console	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	no	Console	---	-	Transmit a carriage return and line feed to the selected output device.
CREATE	no	Compile	---	CREATE cccc	Used 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	no	Compile	---- addr	-	A user variable temporarily storing the stack pointer position, for compilation error checking.
CURRENT	no	Math	d1 d2 --- dsum	-	Leave the double number sum of two double numbers.
D.	no	Console	d ---	-	Print a signed double number from a 32 bit two's complement value. The high-order l6 bits are most accessible on the stack. Conversion is performed according to the current BASE. A blank follows. Pronounced D-dot

Word	New Word	Type	Stack Diagram	Usage	Description
D.R	no	Console	d n ---	-	Print a signed double number d right aligned in a field n characters wide.
D+	no	Math	d1 d2 --- dsum	-	Leave the double number sum of two double numbers
D+-	no	Math	d1 n --- d2	-	Apply the sign of n to the double number d1, leaving it as d2
DABS	no	Math	d --- ud	-	Leave the absolute value ud of a double number
DECIMAL	no	Console	---	-	Set the numeric conversion BASE for decimal input-output.
DEFINITIONS	no	Compile	---	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.
DI	Yes	1802	---	-	Disables interrupts
DIGIT	no	Console	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 isinvalid, leaves only a false flag.
DLITERAL	no	Compile	d --- d (executing) d --- (compiling)	-	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	no	Math	d1 --- d2	-	Convert d1 to its double number two's complement

Word	New Word	Type	Stack Diagram	Usage	Description
DO	no	Execution Flow	n1 n2 --- (execute) addr n --- (compile)	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 current 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>	no	Compile	---	-	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, multidimensional arrays, and compiler generation.
DP	no	Memory	---- addr	-	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	no	Console	---- addr	-	A user variable containing the number of digits to the right of the decimal on double integer input. It may also be used hold output column location of a decimal point, in user generated formatting. The default value is -1
DROP	no	Stack	n ---	-	Drop the number from the stack

Word	New Word	Type	Stack Diagram	Usage	Description
DUMP	no	Console	addr n ---	-	Print the contents of n memory locations beginning at addr. Both addresses and contents are shown in the current numeric base
DUP	no	Stack	n --- n n	-	Duplicate the value on the stack
-DUP	no	Stack	n1 -- n1 (if zero) n1 -- n1 n1 (non-zero)	-	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.
EI	yes	1802	---	-	Enable interrupt processing
ELSE	no	Execution Flow	addr1 n1 --- addr2 n2	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	no	Console	c ---	-	Transmit ascii character c to the selected output device. OUT is incremented for each character output.
ENCLOSE	no	Console	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	no	Execution Flow	---	-	This is an 'alias' or duplicate definition for UNTIL.

Word	New Word	Type	Stack Diagram	Usage	Description

ENDIF	no	Execution Flow	addr n ---	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	no	Memory	addr n --	-	Clear a region of memory to zero from addr over n addresses
ERROR	no		line --- in blk	-	Execute error notification and restart of system. WARNING is first examined. If 1, the text of line n, relative to screen 4 of drive O is printed. This line number may be positive or negative, and beyond just screen 4. If WARNING=O, n is just printed as a message number (non disc installation). If WARNING is -l, 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
ERRS	yes	Console	--- addr	-	Returns the address of the variable where a count of serial I/O communication errors are saved
EXECUTE	no	Execution Flow	addr ---	-	Execute the definition whose code field address is on the stack. The code field address is also called the compilation address.
EXPECT	no	Console	addr count ---	-	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	no	DIctionary	--- addr	-	A user variable containing an address below which FORGETting is trapped. To forget below this point the user must alter the contents of FENCE

Word	New Word	Type	Stack Diagram	Usage	Description
FILL	no	Memory	addr quan b -	-	Fill memory at the address with the specified quantity of bytes b.
-FIND	no	Dictionary	--- 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. Oherwise, only a boolean false is left
FIRST	no	Disk	--- n	-	A constant that leaves the address of the first (lowest) block buffer
FLD	no	Console	--- addr	-	A user variable for control of number output field width. Presently unused in fig-FORTH.
FORGET	no	Dictionary	---	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	no	Dictionary	---		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
GO	yes	Execution Flow	--- addr	-	Transfers execution control to addr
HALT	yes	Multitasking	---	n HALT	Cause task n to stopt executing and go dormant
HERE	no	Dictionary	--- addr	-	Leave the address of the next available dictionary location
HEX	no	Console	---	-	Set the numeric conversion base to sixteen (hexadecimal).
HLD	no	Console	--- addr	-	A user variable that holds the address of the latest character of text during numeric output conversion.

Word	New Word	Type	Stack Diagram	Usage	Description
HOLD	no	Console	c ---	-	Used between <# and #> to insert an ascii character into a pictured numeric output string. e.g. 2E HOLD will place a decimal point.
I	no	Math	--- n	See R	Used within a DO-LOOP to copy the loop index to the stack.
ID.	no	Console	addr --	-	Print a definition's name from its name field address

IF	no		--- addr n (run-time)
			--- P,C2,L0 (compile)
				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 0BRANCH and reserves space for an offset at addr. addr and n are used later for resolution of the offset and error testing.

IH>	yes	Console	addr count ---	-	outputs n bytes of memory data in Intel Hex format from memory starting at addr.
IMMEDIATE	no	Compile	---	-	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	no	Console	--- addr	-	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.
INP	yes	1802	N -- byte	-	inputs a byte of data from a port selected by the 1802's N line to stack. Top byte of resulting word set to zero.

Word	New Word	Type	Stack Diagram	Usage	Description
INTERPRET	no	Compile	---	See NUMBER	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.
J	yes	Math	---	-	Used within nested DO-LOOPs to copy the loop index of the outer loop to the stack.
KEY	no	Console	--- c	-	Leave the ascii value of the next terminal key struck
LATEST	no	Vocabulary	--- addr	-	Leave the name field address of the topmost word in the CURRENT vocabulary
LEAVE	no	Execution Flow	---	-	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	no	Execution Flow	pfa --- lfa	-	Convert the parameter field address of a dictionary definition to its link field address
LIMIT	no	Memory	---- n	-	A constant leaving the address just above the highest memory available for a (disc) buffer. Usually this is the highest system memory
LIST	no	Editor	---	-	Display the ascii text of screen n on the selected output device. SCR contains the screen number during and after this process.

Word	New Word	Type	Stack Diagram	Usage	Description
LIT	no	Math	--- n	-	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	no	Compile	n --- (compiling)	-	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 m value. Compilation is resumed and LITERAL compiles this value.
LOAD	no	Editor	n ---	See ;S and -->	Begin interpretation of screen n. Loading will terminate at the end of the screen or at ;S
LOOP	no	Execution Flow	addr n --- (compiling)	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*	no	Math	n1 n2 --- d	-	A mixed magnitude math operation which leaves the double number signed product of two signed number.
M/	no	Math	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	no	Math	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	no	Math	n1 n2 --- max	-	Leave the greater of two numbers

Word	New Word	Type	Stack Diagram	Usage	Description
MESSAGE	no	Console	n --	-	Print on the selected output device the text of line n relative to screen 4 of drive O. 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 unavailable)
MIN	no	Math	n1 n2 --- min	-	Leave the smaller of two numbers.
MINUS	no	Math	n1 --- n2	-	Leave the two's complement of a number
MOD	no	Math	n1 n2 --- mod	-	Leave the remainder of n1/n2, with the same sign as n1
NEXT	no	Execution Flow	---		This is the inner interpreter that uses the interpretive pointer IP to execute compiled Forth definitions. It is not directly executed but is ff 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
NEXTBYTE	yes	Console	---	-	reads in two ASCII Hex character and converts to binary byte
NFA	no	Compile	pfa --- nfa	-	Convert the parameter. field address of a definition to its name field
NIP	yes	Stack	n1 n2 --- n2	-	Removes the number second from the top of stack.
NUMBER	no	Console	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

Word	New Word	Type	Stack Diagram	Usage	Description
OFFSET	no	Disk	--- addr	-	A user variable which may contain a block offset to disc drives. The contents of OFFSET is added to the stack number by BLOCK. Messages by MESSAGE are independent of OFFSET.
OR	no	Math	n1 n2 -- or	-	Leave the bit-wise logical or of two l6 bit values
ORIGIN	no		--- addr	-	Constant value that returns the base address of USER variables
OUT	no	Console	--- addr	-	A user variable that contains a value incremented by EMIT. The user may alter and examine OUT to control display formatting
OUTP	yes	1802	n port# ---	-	outputs value n to 1802 N-line port#
OVER	no	Stack	n1 n2 --- n1 n2 n1	-	copy the second stack value, placing it as the new top
PAD	no	Console	--- addr	-	Leave the address of the text output buffer, which is a fixed offset above HERE
PAUSE	yes	Multitasking	---	-	used by a task to cooperatively release execution control for one pass of the multitasker
PFA	no	Compile	--- pfa	-	Convert the name field address of a compiled definition to its parameter field address
PICK	yes	Stack	index --- n	-	Places the value of stack element index onto the top of stack
QKEY	yes	Console	--- ch	-	Used by internal intel hex loader to get the next character from console and abort if it's an ESC ($1B)
QOFF	yes	1802	---	-	Set the state of the 1802 Q pin to zero (gnd)
QON	yes	1802	---	-	Set the state of the 1802 Q pin to zero (+5 V)
QUERY	no	Console	---	-	Input 80 characters of text (or until a "return") from the operators terminal. Text is positioned at the address contained in TIB with IN set to zero.
QUIT	no	Execution Flow	---	-	clear the return stack, stop compilation, and return control to the operators terminal. No message is given
R	no	Stack	--- n	-	Copy the top of the return stack to the computation stack.

Word	New Word	Type	Stack Diagram	Usage	Description
R#	no	Console	-- addr	-	A user variable which may contain the location of an editing cursor, or other file related function
R>	no	Stack	--- n	See >R and R.	Remove the top value from the return stack and leave it on the computation stack. See >R and R.
R0	no	Stack	--- addr	See >R and R	A user variable containing the initial location of the return stack. Pronounced R-zero.
REPEAT	no	Execution Flow	addr n --- (compiling)	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	no	Stack	n1 n2 n3 --- n2 n3 n1	-	Rotate the top three values on the stack, bringing the third to the top.
-ROT	yes	Stack	n1 n2 n3 --- n3 n1 n2	-	Counter rotate the top three values on the stack, bringing the second to the top
RP!	no	Stack	---	-	Initialize the return stack pointer from user variable R0.
RUN	yes	Multitasking	task# ---	-	Sets the state of task# to cause it to run when the multi-tasker next gets to it
S->D	no	Math	n --- d	-	Sign extend a single number to form a double number
S0	no	Stack	--- addr	See SP!	A user variable that contains the initial value for the stack pointer.
SCR	no	Editor	--- addr	-	A user variable containing the screen number most recently reference by LIST
SCR>	yes	Editor	N screen# ---	-	Transmits n screens, starting at screen#, to the console. Lines are truncated at the last non-space character and a CR inserted.
SIGN	no	Console	n d --- d	-	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 #>.

Word	New Word	Type	Stack Diagram	Usage	Description
SMUDGE	no	Compile	---	-	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!	no	Stack	---	-	initialize the stack pointer from SO.
SP@	no	Stack	--- addr	-	Return the address of the stack position to the top of the stack, as it was before SP@ was executed
SPACE	no	Console	---	-	Transmit an ascii blank to the output device.
SPACES	no	Console	n ---	-	Transmit n ascii blanks to the output device.
START	yes	Multitasking	n ---	n START taskname	Finds taskname in current dictionary and adds to the task control table as task n
STATE	no	Compile	--- addr	-	A user variable containing the compilation state. A non-zero value indicates compilation
SWAP	no	Stack	n1 n2 --- n2 n1	-	Exchange the top two values On the stack.
TASK	no	Compile	---	-	A no-operation word which can mark the boundary between applications
TASK#	yes	Multitasking	--- n	-	returns the current task's number (to itself)
THEN	no	Execution Flow	---	-	exAn alias for ENDIF
TIB	no	Console	--- addr	-	A user variable containing the address of the terminal input buffer.
TIC	yes	Multitasking	tics ---	-	pauses current task for tics intervals (actual time depends on system tic timer interrupt value)
TOGGLE	no	Math	addr b --	-	Complement the contents of addr by the bit pattern b
-TRAILING	no	Console	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.

Word	New Word	Type	Stack Diagram	Usage	Description
TRAVERSE	no	Compile	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=-l, the motion is toward low memory. The addr2 resulting is address of the other end of the name.
TUCK	yes	Stack	n1 n2 --- n2 n1 n 2	-	copies top of stack and inserts as third item on stack
TYPE	no	Console	addr count ---	-	Transmit count characters from addr to the selected output device
U*	no	Math	u1 u2 --- ud	-	Leave the unsigned double number product of two unsigned numbers
U.	no	Console	u --	-	Output the value on top of stack as an unsigned number
U/	no	Math	ud u1 --- u2 u3	-	Leave the unsigned remainder u2 and unsigned quotient u3 from the unsigned double dividend ud and unsigned divisor u1.
UNTIL	no	Execution Flow	f --- (run-time) addr n --- (compile)	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 (0BRANCH) and an offset from HERE to addr. n is used for error tests.
USER	no	Compile	n ---	n USER cccc	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	no	Compile	n ---	n VARIABLE cccc	creates the definition cccc with its parameter field initialized to n. When cccc is later executed, the address of its parameter field (containing n) is left on the stack, so that a fetch or store may access this location
VLIST	no	Console	---	-	List the names of the definitions in the context vocabulary. "Break" will terminate the listing.

Word	New Word	Type	Stack Diagram	Usage	Description
VOCABULARY	no	Compile	---	VOCABULARY cccc	creates 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 vocabularies ultimately chain to Forth. By convention, vocabulary names are to be declared IMMEDIATE. See VOC-LINK.
VOC-LINK	no	Compile	--- 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 FORGET'ing thru multiple vocabularies.
WARM	no	Execution Flow	---	-	Warm starts the system while maintaining words added to the base dictionary
WARNING	no	Console	--- addr	See MESSAGE, ERROR	A user variable containing a value controlling messages. If = 1 disc is present, and screen 4 of drive 0 is the base location for messages. If = 0, no disc is present and messages will be presented by number. If = -1, execute (ABORT) for a user specified procedure.
WHILE	no	Execution Flow	f --- (run-time) ad1 nl --- ad1 n1 ad2 n2 (compile)	BEGIN ... WHILE ... 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 (0BRANCH) and leaves ad2 of the reserved offset. The stack values will be resolved by REPEAT.

Word	New Word	Type	Stack Diagram	Usage	Description
WIDTH	no	Compile	--- addr	-	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	no	Console	c ---	-	Read the next text characters from the input stream being interpreted, until a delimiter c is found, storing the packed character string beginning at the dictionary buffer HERE. WORD leaves the character count in the first byte, the characters, and ends with two or more blanks. Leading occurrences of c are ignored. If BLK is zero text is taken from the terminal input buffer, otherwise from the disc block stored in BLK.
X.	no	Console	n ---	-	Outputs top of stack formatted in hexadecimal
XOR	no	Math	n1 n2 --- xor	-	Leave the bitwise logical exclusive or of two values

...

VOCABULARY WORDS : EDITOR

Word	New Word	Type	Stack Diagram	Usage	Description
#LAG	Yes	Editor	---		editor internal use
#LEAD	Yes	Editor	---		editor internal use
#LOCATE	Yes	Editor	---		editor internal use
1LINE	Yes	Editor	---		editor internal use
B	Yes	Editor	---		back up over text found by F
b	Yes	Editor	---		editor internal use
C	Yes	Editor	---	C text	spead and copy in text at cursor
CLEAR	Yes	Editor	n ---		clear screen n to blanks
COPY	Yes	Editor	n1 n2 ---		copy screen n1 to n2
D	Yes	Editor	n ---		delete line n to PAD (i.e. cut)
DELETE	Yes	Editor	n ---		deleted n characters before the current cursor position
E	Yes	Editor	n ---		erase line n with blanks
F	Yes	Editor	---	F text	move text to PAD and search forward
f	Yes	Editor	---		editor internal use
find	Yes	Editor	---		editor internal use
H	Yes	Editor	n ---		hold line N at PAD (i.e. copy)
I	Yes	Editor	n ---		insert text from PAD at line n
L	Yes	Editor	---		lists the current screen
LINE	Yes	Editor	---		editor internal use
LIST	No	Forth	n --		lists screen n on the console
M	Yes	Editor	n ---		moves the cursos by the signed about n
MATCH	Yes	Editor	---		editor internal use
-MOVE	Yes	Editor	---		editor internal use
N	Yes	Editor	---		find next occurance of text moved to PAD by F
P	Yes	Editor	n ---	P text	overwrite line n with text
R	Yes	Editor	---		editor internal use
S	Yes	Editor	n ---		spread at line n, moving subsequent lines down
T	Yes	Editor	n ---		type line n and save in PAD
TEXT	Yes	Editor	---		editor internal use
TILL	Yes	Editor	---	TILL text	deleted text from cursos to text
TOP	Yes	Editor	---		positions the cursor at the top of the current screen
X	Yes	Editor	---	X text	find and delete text

VOCABULARY WORDS : ASSEMBLER

Word	New Word	Type	Stack Diagram	Usage:	Description
?FAULT	yes	Compiler	addr1 addr2 ---		used by compiler to detect off page short branches
ADC,	yes	Mnemonic	---		1802 op code
ADCI,	yes	Mnemonic	---		1802 op code
ADD,	yes	Mnemonic	---		1802 op code
ADI,	yes	Mnemonic	---		1802 op code
AGAIN,	yes	Execution control	---	BEGIN, code AGAIN,	inserts a BR instruction back to where the IF, statement was
AND,	yes	Mnemonic	---		1802 op code
ANI,	yes	Mnemonic	---		1802 op code
BEGIN,	yes	Mnemonic	---		1802 op code
BR,	yes	Mnemonic	---		1802 op code
CODE	no	Compiler	---		1802 op code
DEC,	yes	Mnemonic	---		1802 op code
DF	yes	Test	---		inserts a BNF instruction
DIS,	yes	Mnemonic	---		1802 op code
EF1	yes	Test	---		inserts a BN1 instruction
EF2	yes	Test	---		inserts a BN2 instruction
EF3	yes	Test	---		inserts a BN3 instruction
EF4	yes	Test	---		inserts a BN4 instruction
ELSE,	yes	Execution control	---		modifies the previous IF, instructions false code target address to current location
ENDIF,	yes	Execution control	---		modifies the previous IF, instructions true code target address to current location
GHI,	yes	Mnemonic	---		1802 op code
GLO,	yes	Mnemonic	---		1802 op code
IDL,	yes	Mnemonic	---		1802 op code

Word	New Word	Type	Stack Diagram	Usage:	Description
IF,	yes	Execution control	---	test IF, code ELSE, code ENDIF,	1802 op code
INC,	yes	Mnemonic	---		1802 op code
INP,	yes	Mnemonic	---		1802 op code
IRX,	yes	Mnemonic	---		1802 op code
LBR,	yes	Mnemonic	---		1802 op code
LDA,	yes	Mnemonic	---		1802 op code
LDI,	yes	Mnemonic	---		1802 op code
LDN,	yes	Mnemonic	---		1802 op code
LDX,	yes	Mnemonic	---		1802 op code
LDXA,	yes	Mnemonic	---		1802 op code
MARK,	yes	Mnemonic	---		1802 op code
NEXT	yes	Compiler	---		1802 op code
NOP,	yes	Mnemonic	---		1802 op code
NOT	yes	Test	---		converts previous branch if false instruction to branch if true
OR,	yes	Mnemonic	---		1802 op code
ORI,	yes	Mnemonic	---		1802 op code
OUT,	yes	Mnemonic	---		1802 op code
PHI,	yes	Mnemonic	---		1802 op code
PLO,	yes	Mnemonic	---		1802 op code
Q	yes	Test	---		inserts a BNQ instruction
REPEAT,	yes	Execution control	---		inserts a BR instruction back to where the BEGIN, statement was
REQ,	yes	Mnemonic	---		1802 op code
RET,	yes	Mnemonic	---		1802 op code
SAV,	yes	Mnemonic	---		1802 op code
SD,	yes	Mnemonic	---		1802 op code
SDB,	yes	Mnemonic	---		1802 op code
SDBI,	yes	Mnemonic	---		1802 op code
SDI,	yes	Mnemonic	---		1802 op code
SEP,	yes	Mnemonic	---		1802 op code

Word	New Word	Type	Stack Diagram	Usage:	Description
SEQ,	yes	Mnemonic	---		1802 op code
SEX,	yes	Mnemonic	---		1802 op code
SHL,	yes	Mnemonic	---		1802 op code
SHLC,	yes	Mnemonic	---		1802 op code
SHR,	yes	Mnemonic	---		1802 op code
SHRC,	yes	Mnemonic	---		1802 op code
SM,	yes	Mnemonic	---		1802 op code
SMB,	yes	Mnemonic	---		1802 op code
SMBI,	yes	Mnemonic	---		1802 op code
SMI,	yes	Mnemonic	---		1802 op code
STR,	yes	Mnemonic	---		1802 op code
STXD,	yes	Mnemonic	---		1802 op code
UNTIL,	yes	Execution control	---	BEGIN code test UNTIL,	inserts a branch address back to BEGIN, for previous conditional instruction
WHILE,	yes	Execution control	---	BEGIN test WHILE, code REPEAT,	inserts a branch address for following UNTIL,
XOR,	yes	Mnemonic	---		1802 op code
XRI,	yes	Mnemonic	---		1802 op code
Z	yes	Test	---		1802 op code