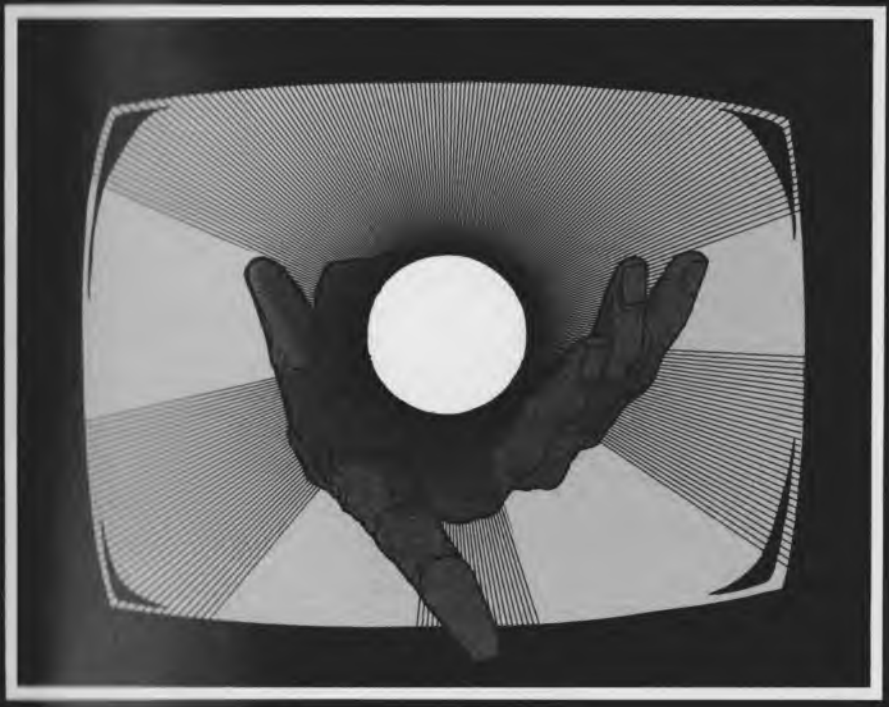
ACCELERATED SOFTWARE INC.

enhanced FORTH

THE POWER OF MACHINE CODE ... THE EASE OF BASIC

Features: An assembler, screen editor, floating point instructions, and an easy to use manual.



A FULL IMPLEMENTATION OF STANDARD FORTH

TABLE OF CONTENTS

	Page
I. INTRODUCTION	2
II. OVERVIEW	3
- The Stack	3
- Reverse Polish Notation	4
- Definitions	5
- Dictionary and Vocabularies	6
- Screens	6
- Floating and Fixed Point	7
III. ASI-FORTH	8
- Standard Definitions	9
- Editor	34
- Assembler Implementation	35
- Floating Point Definitions	37
- Graphics Definitions	40
- Music Definitions	44
APPENDIX A - STARTING UP	46
APPENDIX B - DISK NOTES	48

COPYRIGHT NOTICE © 1984

The ASI-FORTH Computer program is copyrighted and all rights are reserved by ACCELERATED SOFTWARE INC. Only you, as an original purchaser, may use the ASI-FORTH program and only on one computer system. Except for the limited purpose of system back-up, any copying, duplicating, selling, or otherwise distributing the ASI-FORTH computer program, is a violation of the law.

I. INTRODUCTION

Welcome to the world of FORTH!!

By choosing ASI-FORTH you have given yourself the advantages of a fast and efficient FORTH interpreter/compiler without losing the ease of BASIC.

This version of ASI-FORTH comes with a full screen editor, an assembler, floating point routines, and routines that allow you to use the full sound and graphics capabilities of your COMMODORE 64. Also included on the distribution disk are utilities that allow you to back-up FORTH data disks, decode word definitions and create application versions of ASI-FORTH. Please refer to APPENDIX A for further information on these utilities.

This manual is not intended to be an introduction to the FORTH language, but a reference manual that lists all the definitions available in this FORTH. If you are unfamiliar with the FORTH language, we would suggest "Starting Forth" by Leo Brodie, FORTH INC., published by Prentice-Hall. This is an excellent guide for the beginning FORTH user.

ACKNOWLEDGEMENT

This publication and the ASI-FORTH computer program are based in part on the fig-FORTH model made available by

THE FORTH INTEREST GROUP
P.O. BOX 1105
SAN CARLOS
CALIFORNIA 94070

II. OVERVIEW

Forth is a stack based, reverse Polish notational, self defining language. DON'T PANIC, in this chapter we will explain all the terms used above, as well as other important concepts in FORTH.

THE STACK

A stack is like a pile of cards from which you can only add or remove one card at a time to/from the top of the pile.

That is, if you place, in the following order, the nine-of-hearts, queen-of-diamonds and the ace-of-spades on the pile, the only way to retrieve them is in the opposite order in which they were placed, i.e. Ace-of-spades, queen-of-diamonds, nine-of-spades.

Forth has two different stacks, the parameter and return stacks. The most important to the novice Forth user is the parameter stack. Whenever you type a number in Forth, the value is placed on the parameter stack and most functions use the parameter stack for both input and output of values.

FUNCTION	YOU TYPE	COMPUTER RESPONDS	STACK
(number)	2 <CR>	OK	2
(number)	3 <CR>	OK	3 2
(number)	5 <CR>	OK	5 3 2
add top two number on stack	+ <CR>	OK	8 2
subtract top two numbers on stack	- <CR>	OK	-6
Print top of stack	. <CR>	-6 OK	Emptv

NOTE: The stack is read from left to right; thus the top of stack is the leftmost value.

There are many functions that play with the stack. Here are some examples:

FUNCTION	YOU TYPE	COMPUTER RESPONDS	STACK
	1 <CR>	OK	1
	2 <CR>	OK	2 1
	3 <CR>	OK	3 2 1
Swap top two values in stack	SWAP <CR>	OK	2 3 1
Duplicate first value in stack	DUP <CR>	OK	2 2 3 1
Remove first value in stack	DROP <CR>	OK	2 3 1
Rotate top 3 values in stack	ROT <CR>	OK	1 2 3
Bring 2nd value to top of stack	OVER <CR>	OK	2 1 2 3

If you want to learn about the return stack, you should acquire a more advanced book on Forth.

REVERSED POLISH NOTATION:

Reversed Polish notation describes the order in which you enter a calculation.

For example on a regular calculator you would enter the sequence:
5 * 6 + 3 - 7 = to get an answer, but in reverse Polish you would enter:

(1) 5 6 * 3 + 7 -
or (2) 5 6 * 3 7 - +

In reversed Polish notation the operators (+,-,*,/ ...) perform their function on that last two values on the stack.

EXAMPLE:

(1)	STACK		(2)	STACK		
5	5		5	5		
6	6	5	6	6	5	
*	30		*	30		
3	3	30	3	3		
+	33		7	7	3	30
7	7	33	-	-4	30	
-	26		+	26		

Here are some examples of regular equations and their reversed Polish notation equivalences:

Regular	Reversed Polish
4 + 6	4 6 +
5 / 9	5 9 /
(4 + 5) / (6 + 7)	4 5 + 6 7 + /
6 * (3 - 5 / (7 - 2))	6 3 5 7 2 - / - *

Lets solve the last equation.
The answer is 12. (Right?)

(6 * (3 - 5 / 5) --> 6 * (3 - 1) --> 6 * 2 = 12)

	Stack				
6	6				
3	3	6			
5	5	3	6		
7	7	5	3	6	
2	2	7	5	3	6
-	5	5	3	6	(7 - 2)
/	1	3	6		(5 / 5)
-	2	6			(3 - 1)
*	12				(2 * 6)

DEFINITIONS:

A self-defining language is a language where new definitions are based upon the existing language and they themselves become part of the language.

In Forth the symbol ":" (Colon) means - Create a new definition, and the symbol ";" (Semi-colon) means end the definition. Thus if we wanted to create a definition that found the square of a number, called SQUARE, the Forth code would like like this:

```
:       SQUARE        DUP              *              ;
```

Create Definition	Call it SQUARE	Duplicate top number on stack	Multiply top two numbers on stack	End Definition

To find the square of 7, we would now merely type 7 SQUARE <CR>. This would then leave 49 on the stack.

Let us see what the computer actually does when we type 7 SQUARE:

Typed in	Machine executes	Stack
7	7	7
SQUARE	:	
	DUP	7 7
	*	49

Now if we wanted a definition to sum the squares of two numbers we can use the already defined routine SQUARE within this new definition. Let's call this SUM-SQUARE

```
:       SUM-SQUARE      SQUARE          SWAP        SQUARE  +  ;
```

Create Definition	Call it SUM-SQUARE	Square Number on top of stack	Swap top 2 numbers on stack	Square Number on top of stack and add top two values

Therefore 5 7 SUM-SQUARE would leave 74 on the stack. Let us see what the computer actually does:

Typed in	Computer executes	Stack
5	5	5
7	7	5 7
SUM-SQUARE	:	
	SQUARE	
	:	
	DUP	5 5 7
	*	25 7
	;	
	SWAP	7 25
	SQUARE	
	:	
	DUP	7 7 25
	*	49 25
	;	
	+	74
	;	

A definition that you create or one that already exists in Forth can be used by any number of other definitions. Since any definition can be based upon any existing definition, there is no practical limit to the number of levels of definitions a new definition can be built on.

DICTIONARY AND VOCABULARIES:

The dictionary is the list of the defined "words" in the language, including system definitions, user definitions, code, variables, and other types of definitions. The dictionary resides in memory, in compiled form.

A "word" in FORTH is any string of up to 31 characters bounded by spaces. Any printable character, other than space, backspace and carriage return, may be used in a word.

Words are added to the dictionary by "defining words", examples of which are: VARIABLE, CONSTANT, : ("colon"), CREATE, etc.

A vocabulary is an independent sub-list of the dictionary. You can have several different vocabularies in your dictionary.

SCREENS:

Forth code written by a user is stored on disk in blocks of 1024 characters. These blocks are called screens. In this Forth screen text is arranged in 32 lines of 32 characters.

You refer to a specific screen by its number, thus 4 LIST <CR> pulls the 4th block of 1024 characters into a memory buffer and lists it on the current output device.

Here are some commands that work with screens:

Command	Effect
n LIST	Load screen n into memory and list screen on output device.
n LOAD	Load screen n into memory and compile code in screen into the Forth dictionary. (Dictionary - name of the collection of all known definitions).
n EDIT	Load screen n into memory and allow you to edit it using the full screen editor.
n1 n2 INDEX	Does NOT load any screens into memory, but lists the first line of screens n1 to n2 on the current output device. Can be terminated by hitting the return key.
n BLOCK	Loads screen n into memory and returns, on the stack, the address in memory of the buffer that holds screen n. (If screen n is already in memory it just returns the address of the buffer).

Note: The first line of a screen should be a comment line describing the definitions on the screen. Comment lines start with a left bracket separated by spaces " (" and end with a right bracket followed by a space ") ".

FLOATING AND FIXED POINT:

A fixed point number is an integer value, that is, there is NO fractional part. A floating point number can have a fractional part.

Fixed	Floating
0	0.1
-31767	-32767.0
234	2.34552E+12

In Forth there are four types of fixed numbers:
unsigned-single; unsigned-double; signed-single and signed-double.

Type	Lower Bound	Upper Bound	Memory requirements
Unsigned-Single	0	65535	2 Bytes (Characters)
Unsigned-Double	0	4,294,967,295	4 Bytes
Signed-Single	-32768	32767	2 Bytes
Double-Signed	-2,147,483,648	2,147,483,647	4 Bytes

In most cases in Forth, fixed-point double precision numbers are not used.

Floating point numbers are not usually available in Forth, but by loading screen £5 this Forth allows you to use them. To use the floating point facility all floating point numbers and functions have to be preceeded by a pound sign " £ ".

NOTE: Floating point numbers take 6 bytes of memory.

Examples:

£ 3.4 Enter floating point number 3.4 on stack. (Note the space between "£" and 3.4).

£+ Add the top two floating point numbers on the stack. (Note: No space between "£" and "+").

£. Print-out top floating point value on stack.

IMPORTANT NOTE: In standard Forth and in this Forth the method for entering a double precision number is to place a decimal point anywhere within the number. Do NOT confuse this with floating point numbers as they are preceeded by a pound sign (£).

Number	Type of Number
34	Single (Signed or Unsigned)
-56	Single Signed
7.8	Double (Signed or Unsigned)
-78.	Double Signed
£ -6.7	Floating
£ 0.1E-34	Floating

III. ASI-FORTH

In this chapter we present a list of all words available in ASI-FORTH.

The chapter is divided into six sections:

1. Standard definitions
2. The Editor
3. Assembler implementation
4. Floating point routines
5. Graphics definitions
6. Music definitions.

Within each section, the words are arranged in alphanumeric order.

KEY:

stack notation: before —— after; top of stack on RIGHT

n, n1, ...	16-bit signed numbers
d, d1, ...	32-bit signed numbers
u, u1, ...	16-bit unsigned numbers
ud, ud1, ...	32-bit unsigned numbers
b, ...	8-bit signed numbers
f, ...	boolean flag
c ...	ascii character value
addr, addr1 ...	address
f1, f2, ...	floating-point number 48-bit.
sp ...	16-bit number in the range 0-7 indicates sprite number.

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

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

d1 --- d2
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 #>.

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

#BUF --- n
A constant returning the number of disk buffers allocated.(3)

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

' --- addr
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".

(
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.

(.")
The run-time procedure, compiled by ." which transmits the following inline text to the selected output device. 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. See ;CODE.

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

.CPU Prints the processor name (ie. 6502) from $22 +ORIGIN encoded as a 32 bit, base 36 integer.

(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) 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 32 indicates the full line text length.

(LOOP)

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 unconvertible digit. Used by NUMBER.

(OPEN) n1 n2 n3 ---
The run-time procedure, compiled by OPEN, that opens the file contained in the in-line text on logical unit n1, device n2 and channel n3. See OPEN.

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

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

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

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

+!

n addr ---
Add n to contents of location addr.

+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)
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 greaer 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 bytes to the origin parameter area. This definition is used to access or modify the boot-up parameters at the origin area.

,

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

-

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

-->

Continue interpretation with the next disc screen. (pronounced next-screen).

-DUP

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.

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

."
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 form ".

.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
Leave the signed quotient of n1/n2.

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

Ø 1 2 3 --- n
These small numbers are used so often that is attractive to define them by name in the dictionary as constants.

Ø< n --- f
Leave a true flag if the number is less than zero (negative), otherwise leave a false flag.

Ø= n --- f
Leave a true flag if the number is equal to zero, otherwise leave a false flag.

ØBRANCH 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+ n1 --- n2
Increment n1 by 1.

2+ n1 --- n2
Leave n1 incremented by 2.

2! nlow nhigh addr ---
32 bit store. nhigh is stored at addr; nlow is stored at addr+2.

2@ addr --- nlow nhigh
32 bit fetch. nhigh is fetched from addr; nlow is fetched from addr+2.

2DUP n2 n1 --- n2 n1 n2 n1
Duplicate the top two values on the stack. Equivalent to OVER OVER.

2LIST n ---
Displays on the selected output device screens n & n+1. Output is suitable for source text record, and includes a reference line at the bottom taken from line 31 of screen 4.

:

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.

;

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

;CODE

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 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

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.

<

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

<#

Setup for pictured numeric output formatting using the words:
<# # #S SIGN #>
The conversion is done on a double number producing text at PAD.

<BUILDS

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 procedures to written in highlevel rather than in assembler code (as required by ;CODE).

=

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

>

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

>R

n ---
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 --
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 if the stack is out of bounds.

?TERMINAL --- f
Perform a test of the terminal keyboard for actuation any key. A true flag indicates actuation.

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

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

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

AGAIN addr n --- (compiling)
Used in a colon-definition in the form:
BEGIN ... AGAIN
At run-time, AGAIN forces execution to return to corresponding BEGIN. There is no effect on the stack. Execution cannot leave this loop (unless R> DROP is executed one level below).

At compile time, AGAIN compiles BRANCH with an offset from HERE to addr. n is used for compile-time error checking.

ALLOT n ---
Add the signed number to the dictionary pointer DP. May be used to reserve dictionary space or re-origin memory. n is in bytes.

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

B/BUF --- n
This constant leaves the number of bytes per disk buffer, the byte count read from disk 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 32 lines of 32 characters each.

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

BASE --- addr

A user variable containing the current number base used for input and output conversion.

BEGIN --- addr n (compiling)

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 beginning at addr with blanks.

BLK --- addr

A user variable containing the block number being interpreted. If zero, input is being taken from the terminal input buffer.

BLOCK n --- addr

Leave the memory address of the block buffer containing block n. If the block is not already in memory, it is transferred from disk to which ever buffer was least recently written. If the block occupying that buffer has been marked as updated, it is rewritten to disk before block n is read into the buffer. See also BUFFER, R/W UPDATE FLUSH.

BRANCH

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

BUFF-COMMAND ---

Prints out the disk command "B-P: 9 Ø" on the current output device.

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 disk. The block is not read from the disc. The address left is the first cell within the buffer for data storage.

BYE Returns control back to Basic.

C! b addr ---

Store 8 bits at address.

C, b ---
Store 8 bits of b into the next available dictionary byte, advancing the dictionary pointer.

C/L --- n
Constant leaving the number of characters per line; used by the editor. (32)

C@ addr --- b
Leave the 8 bit contents of memory address.

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

CKEY --- n
Leaves the ascii value of the next terminal key struck if default input is the keyboard. If input is something else KEY is called. Note: A cursor IS displayed.

CLOSE n ---
Close logical file n. Note: an error is given if file is NOT open.

CLOSEALL ---
Perform the Kernel routine Close all files.

CMD n ---
Set logical file n as default output device. The screen is considered as logical file 0. Thus to return to the screen as being the default output device the command would be: Ø CMD.

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

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

CONSTANT n ---
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 vale of n to the stack.

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

COUNT addr1 --- addr2 n

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

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

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

CURRENT --- addr

A user variable that contains a link to the dictionary which definitions will be linked to.

CURSOR ---

Displays a cursor at current screen location as determined by the Kernel.

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 ---

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 doubled number d right aligned in a field n characters wide.

DABS d --- ud

Leave the abolute value ud of a double number.

DECIMAL

Set the numeric conversion BASE for decimal input-output.

DEFINITIONS

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.

DEVIN

--- addr

A variable that contains the current default input device. See INP.

DEVOUT

--- addr

A variable that contains the current default output device. See CMD.

DIGIT

c nl --- n2 tf (ok)
c nl --- ff (bad)

Convert the ascii character c (using base nl) 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)

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

dl --- d2

Convert dl to its double number two's complement.

DO

nl 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 nl 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 nl 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>

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.

DOOPEN

Call the Kernel routine OPEN.

DP ---- 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 ---- addr

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

DROP n ---

Drop the number from the stack.

DUP n --- n n

Duplicate the value on the stack.

ELSE addr1 n1 --- addr2 n2
(compiling)

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 ---

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

EMPTY-BUFFERS

Mark all block-buffers as empty. Updated blocks are not written to the disk. This is also an initialization procedure before first use of the disk.

ENCLOSE addrl c ----
adrl n1 n2 n3
The text scanning primitive used by WORD. From the text address addrl 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
This is an 'alias' or duplicate definition for UNTIL.

ENDIF addr n --- (compile)
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 ASI-FORTH. See also IF and ELSE.

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

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

ERROR line --- in blk
Execute error notification and restart of system. WARNING is first examined. If 1, the text of line n, relative to screen 4 of drive Ø is printed. This line number may be positive or negative, and beyond just screen 4. If WARNING = Ø, n is just printed as a message number (non disk 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). ASI-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 ---
Transfer characters from the terminal to address, until a "return" or the count of characters have been received. One or more nulls are added or the end of the text.

FENCE --- 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.

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

FLUSH
Write all UPDATEd disk buffers to disk. Should be used after editing, before removing a disk, or before exiting FORTH.

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

FORTH
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
Leave the address of the next available dictionary location.

HEX
Set the numeric conversion base to sixteen (hexadecimal).

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

HOLD 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 --- n
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)

Occurs is a colon-definition in form:

IF (tp) ... ENDIF
IF (tp) ... ELSE (fp) ... ENDIF

At run-time, IF selects execution based on a boolean flag. If f is true (non-zero), execution continues ahead thru the true part. If f is false (zero), execution skips till just after ELSE to execute the false part. After either part, execution resumes after ENDIF. ELSE and its false part are optional.; if missing, false execution skips to just after ENDIF.

At compile-time IF compiles ØBRANCH 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 preceding it with [COMPILE].

IN

--- addr

A user variable containing the byte offset within the current input text buffer (terminal or disk) 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 disk screens.

INP

n ---

Set logical file n as default input device. The keyboard is considered as logical file Ø. Thus to return to the keyboard as being the default input device the command would be: Ø INP.

INTERPRET

The outer text interpreter which sequentially executes or compiles text from the input stream (terminal or disk) 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
Leaves the ascii value of the character in the input stream. If default input device is the keyboard KEY will wait for a terminal key to be struck and return with that value. Note: No cursor is displayed.

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

LEAVE
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 ---- addr
A constant leaving the address just above the highest memory available for a disk buffer. Usually this is the highest system memory.

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

LIT --- 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 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 a value. Compilation is resumed and LITERAL compiles this value.

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

LOOP addr n --- (compiling)
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
Leave the greater of the 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
Leave the smaller of two numbers.

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

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

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

NOOP
A Forth 'no operation'.

NOPRINTER ---
Prints a trailing <CR>, set output to the screen and closes the printer channel.

NUMBER addr --- d
Convert a character using 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 efect occurs. If numeric conversion is not possible, an error message will be given.

OFFSET --- addr
A user variable that currently is not used in this version of ASI-Forth.

OPEN n1 n2 n3 ---
Use: n1 n2 n3 OPEN "file-spec". Open the logical file n1 for device n2 using channel n3. The file name appears after the OPEN enclosed by quotes. Note: " " - A single blank filename is convert to a NULL file specification.

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

OUT --- adr
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
Copy the second stack value, placing it as the new top.

PAD --- addr
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.

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

PRINTER ---
Open the printer logical channel 4 and set output the that device. Note: Printer is opened for upper-case only. See OPEN.

QUERY
Input 80 characters of text (or until a "return") from the operators terminal. Text is positioned at the address contained in TIB with IN set to zero.

QUIT
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
A user variable which is unused in ASI-forth.

R/W addr blk f ---
The ASI-FORTH standard disk 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=Ø write and f=1 read. R/W determines the location on mass storage, performs the read-write and performs any error checking.

R> --- n
Remove the top value from the return stack and leave it on the computation stack. See <R and R.

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

REPEAT addr n --- (compiling)
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
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 RØ.

RP@ --- addr
Leaves the current value in the return stack pointer register.

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

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

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

SEC/BLK --- n
A constant that contains the number of sectors used per block.

SEC --- addr
A variable used by the disk interface, containing the sector number last read or written.

SEC-READ
Reads one sector into memory. All parameters must have been set by SET-DRIVE and SET-IO. The status on completion is stored in DISK-ERROR.

SEC-WRITE
Writes one sector from memory. All parameters must have been set by SET-DRIVE and SET-IO. The status on completion is stored in DISK-ERROR.

SET-DRIVE ---
Updates the DISK-COMMAND, that is sent to the drive, with the value stored in DISK-DRIVE and Initializes correct drive to power-on state.

SET-IO ---
Updates the DISK-COMMAND, that is sent to the drive, with track and sector stored in variables TRACK & SEC.

SETLFS n1 n2 n3 ---
Calls the Kernel routine SETLFS with the logical unit n1, device n2 and channel n3. See OPEN.

SETNAM addr count ---
Calls the Kernel routine SETNAM with the name defined by addr and count. See OPEN.

SIGN 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 betweeen <# 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! Initialize the stack pointer from SØ.

SP@ --- addr
Return the address of the stack position to the top of the stack, as it was before SP@ was executed.

SPACE
Transmit an ascii blank to the output device.

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

STATE --- addr
A user variable containing the compilation state. A non-zero value indicates compilation.

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

T&SCAL n ---
Track & Sector calculation for disk IO. n is the total sector count derived by:
n = block# * SEC/BLK
The corresponding track and sector are calculated.
The track number is stored in SEC. T&SCALC is usually executed before SET-IO.

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
An alias for ENDIF.

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

TOGGLE addr b ---
Complement the contents of addr by the bit person b.

TRACK --- addr
A variable used by disk IO. Contains the track number last read or written.

TRAVERSE addr1 n --- addr2
Move across the name field of a ASI-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.

TYPE addr count ---
Transmit count characters from addr to the selected output device.

U< u1 u2 --- f
Leave the boolean value of an unsigned less-than comparison. Leaves f = 1 for u1 < u2; otherwise it leaves Ø. This function must be used when comparing memory addresses. u1 and u2 are unsigned 16 bit integers.

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

U. ul ---
Print a number from an unsigned 16 bit value, converted according to the numeric BASE. A trailing blank follows.

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

UNTIL f --- (run-time)
addr n --- (compile)
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 (ØBRANCH) and an offset from HERE to addr. n is used for error tests.

UPDATE
Marks the most recently referenced block (pointed to by PREV) as altered. The block will subsequently be transferred automatically to disk 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 ---
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
A defining word used in the form;
n VARIABLE cccc
When VARIABLE is executed, it creates the definition cccc with its parameter field initialized to n. When cccc is later executed, the address of its parameter field (containing n) is left on the stack, so that a fetch or store may access this location.

VOC-LINK --- addr
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

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 "cccc DEFINITIONS" will also make cccc the CURRENT vocabulary into which new definitions are placed.

In ASI-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.

VIC-3040
VIC-4040
VIC-B050

--- addr

Track and sector tables for each type of disk drive. (VIC-1541 is the same as VIC-4040).
These are used in conunction with DISK-TYPE.

VLIST

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

WARM

Warm start routine. Can be called from Basic by doing a SYS to the start of Forth plus 4. Take a look at the Basic part of Forth and add 4 to the SYS address.

WARNING

--- addr

A user variable containing a value controlling messages. If = 1 disk is present, and screen 4 of drive Ø is the base location for messages. If = Ø, no disk is present and messages will be presented by number. If = -1, execute (ABORT) for a user specified procedure. See MESSAGE, ERROR.

WHILE

f --- (run-time)
adl nl --- adl nl ad2 n2

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, 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 (BRANCH) and leaves ad2 of the reserved offset. The stack values will be resolved by REPEAT.

WIDTH --- addr

In ASI-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 ---

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 disk block stored in BLK. See BLK, IN.

X

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

XOR n1 n2 --- xor

Leave the bitwise logical exclusive or of two values.

[

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]

Used in a colon-definition in form:

```
: xxx [COMPILE] FORTH ;
```

[COMPILE] will force the compilation of an immediate definition, that would otherwise execute during compilation. The above example will select the FORTH vocabulary when xxx executes, rather then at compile time.

]

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

EDITOR

The editor is a 32 line by 32 characters per line full screen editor. There are three ways of invoking it (See following definitions). The arrow , home , insert and delete keys all work correctly. Insert and delete work with one line at a time. The function keys are defined as follows:

F1 - Exit editor. (Remember to flush and/or update).

F2 - Not Used.

F3 - Edit the next screen. (Remember to update current screen if required).

F4 - Edit the previous screen. (Remember to update current screen if required).

F5 - Set screen as updated.

F6 - Set screen as NOT updated.

F7 - Insert line at current cursor position. (Can't be used on last line of screen).

(Last line in screen MUST be all blanks).

F8 - Delete from cursor to end of line and pull in next line. (Can't be used on last line of screen) (Next line must be able to fit in deleted space).

EDITOR DEFINITIONS

EDITOR ---

Set EDITOR as the context vocabulary. This vocabulary contains lower level code which the main editor routines use. These routines are not made for general use and thus are not documented.

EDIT n ---

Place you into the editor with screen n.

ED ---

Place you into the editor with the last screen edited or loaded.

ERR in blk ---

Place you into the editor with the screen that when just loaded, had an error. The cursor will be placed on the last letter of the offending word. See ERROR.

ASSEMBLER IMPLEMENTATION

The assembler is accessed by loading screen no. 9. The assembler is based upon 'reverse Polish' notation. This means that the operand comes before the operator. Each 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,). Therefore:

```
        34 ,X LDA,          in FORTH would be:

        LDA 34,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 generation of the equivalent of macros.

GLOSSARY OF FORTH ASSEMBLER

IP	address of the Interpretive pointer in zero-page.
W	address of the code field pointer in zero-page.
N	address of 2 byte scratch area in zero-page.
UP	address of User Pointer.
.A	specify accumulator addressing mode.
#	specify immediate mode for machine byte literals.
,X ,Y	specify memory indexed addressing mode.
X))Y	specify indirect memory addressing mode.
)	indirect addressing for JMP, & JSR,
MEM	specify direct addressing mode.
???,	assemble instruction ???.
PUSHN	address of routine to push N on stack and goto NEXT.
PUSHWN	address of routine to push W on stack and goto PUSHN.
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 pointer machine code.

BEGIN, save address for branch back at UNTIL,

UNTIL, branch to last use of BEGIN, i.e BEGIN, ... 0= UNTIL,

IF, execute if condition, set-up branch ie. 0= IF, ... ELSE, ... THEN,

ELSE, set-up jump to THEN, and place offset in branch at IF,

THEN, terminate IF, place address in jump at ELSE,

0= Used before IF, and BEGIN, as BEQ

CS Used before IF, and BEGIN, as BCS

0< Used before IF, and BEGIN, as BMI

>= Used before IF, and BEGIN, as alais CS

NOT Used after (0=,CS,0<,>=) for (BNE,BCC,BPL,BCC)

IX2 INX, INX,

DX2 DEX, DEX,

L0: - L8: Used to store label addresses. (See LABEL)

LABEL Used to define L0:-L8: ie. LABEL L0: .... L0: MEM JMP,

BYTE.IN Used to calculate indexes into FORTH variables. 2 BYTE.IN INDEX # LDA,

REPLACE.BY Change old definition to point to next one. CODE NEW. LABEL L0: ... NEXT MEM JMP, L0: REPLACE.BY . END-CODE

CODE Start assembler definition. CODE cccc

END-CODE End assembler definition and SMUDGE cccc

SUBROUTINE Start assembler definition that is to be used as a subroutine.

END-SUBROUTINE End assembler definition.

FLOATING POINT ROUTINES

Floating Point routines can be accessed by loading screen 5. All floating point operations are preceeded by a pound sign (£). Floating point numbers are 6 bytes long.

For more description of floating point functions see the Basic reference manual.

FLOAT

Sets FLOAT to the context vocabulary. This vocabulary contains lower level code which the main floating point routines use. These routines are not made for general use and thus are not documented.

£ string --- f1

Convert string to floating point number. The number is either placed on the stack or compiled into the dictionary, depending on whether you are compiling or not.

£. f1 ---

Print out floating point number using normal Basic format.

£+ f1 f2 --- f3

Add f1 to f2 giving f3.

£- f1 f2 --- f3

Subtract f2 from f1 giving f3.

£/ f1 f2 --- f3

Divide f1 by f2 giving f3.

£* f1 f2 --- f3

Multiply f1 by f2 giving f3.

£DROP f1 ---

Drop a floating point number from the stack.

£SWAP f1 f2 --- f2 f1

Exchange the top two floating point values on the stak.

£DUP f1 --- f1 f1

Duplicate the floating point value on the stack.

£OVER f1 f2 --- f1 f2 f1

Copy the second floating point number, placing it as the new top.

£ROT f1 f2 f3 --- f2 f3 f1

Rotate the top three floating point numbers on the stack, bringing the third to the top.

£! f1 addr ---

Store floating point number at address addr. Requires 6 bytes of storage. See £VARIABLE.

£@ addr --- f1
Leave the floating point contents of address addr on the stack.

£VARIABLE --- addr
A defining word of the form:
f1 £VARIABLE cccc
When £VARIABLE is executed, it creates the definition cccc with its parameter field initialized to f1. When cccc is later executed the address of its parameter field (containing f1) is left on the stack, so that a fetch or store may access this location.

£CONSTANT --- f1
A defining word used in the form:
f1 £CONSTANT cccc
to create word cccc, with its parameter field containing f1. When cccc is later executed, it will push the floating point value of f1 to the stack.

£LOG f1 --- f2
Perform the Basic function LOG(f1).

£SGN f1 --- f2 (-1 , 0 , 1)
Perform the Basic function SGN(f1).

£ABS f1 --- f2
Perform the Basic function ABS(f1).

£INT f1 --- f2
Perform the Basic function INT(f1).

£SQR f1 --- f2
Perform the Basic function SQR(f1).

£↑ f1 f2 -- f3
Perform the Basic function f1↑f2.

£EXP f1 --- f2
Perform the Basic function EXP(f1).

£RND f1 --- f2 (f1 = {-1 0 1})
Perform the Basic function RND(f1).

£COS f1 --- f2
Perform the Basic function COS(f1).

£SIN f1 --- f2
Perform the Basic function SIN(f1).

£TAN f1 --- f2
Perform the Basic function TAN(f1).

£ATN f1 --- f2
Perform the Basic function ATN(f1).

S->f n --- fl
Convert 16 bit signed integer to floating point number.

f->S fl --- n
Convert floating point number to 16 bit signed integer.

f< fl f2 --- f
Leave a true flag if fl is less than f2; otherwise leave a false flag.

f> fl f2 --- f
Leave a true flag if fl is greater than f2; otherwise leave a false flag.

f= fl f2 --- f
Leave a true flag if fl is equal to f2; otherwise leave a false flag.

GRAPHICS DEFINITIONS

Graphics definitions can be accessed by loading screen no. 37 on your ASI-FORTH disk.

GRAPHICS ---
Vocabulary that contain primitive graphic definitions.

GON ---
Turn on high resolution graphics.

GOFF ---
Turn off high resolution graphics.

CLEAR ---
Clear graphics screen.

BORDER cl ---
Set border to colour cl.

HUE clb clf ---
Set Background colour to clb and foreground colour to clf.

MODE n ---
Sets mode of plotting for subsequent uses of LINE and PLT.

-1 -- Compliment
0 -- Clear
1 -- Set

ON x y ---
Turn pixel at graphics location x,y on.

OFF x y ---
Turn pixel at graphics location x,y off.

?ON x y --- f
Return a boolean value indicating if pixel at location x,y is on.
On = True
Off = False

COMP x y ---
Compliment pixel at graphics location x,y

PLT x y ---
Set pixel at graphics location x,y depending of current mode of plotting. (See MODE)

LINE x1 y1 x2 y2 ---
Draw a line of graphics location x1,y1 to x2,y2 using the current mode of plotting. (See MODE)

CSET x y clb clf ---
Set background colour to clb and foreground colour to clf for the 8 byte area that the graphics pixel x,y falls into.

SPON sp ---
Turn sprite number sp on.

SPOFF sp ---
Turn sprite number sp off.

CCSET x y clb clf ---
Set background colour to clb and foreground colour to clf for the 8 byte area defined by CHARACTER location x,y.

SPMCON sp ---

Set sprite sp to multi-colour.

SPMCOFF sp ---
Set sprite sp to non-multi-colour.

XEXPON sp ---
Turn on sprite sp X-expand.

XEXPOFF sp ---
Turn off sprite sp X-expand.

YEXPON sp ---
Turn on sprite sp Y-expand.

YEXPOFF sp ---
Turn off sprite sp Y-expand.

SPCOL cl sp ---
Set sprite sp to colour cl.

SPCOL0 cl ---
Set multi-colour sprite colour-0 to cl.

SPCOL1 cl ---
Set multi-colour sprite colour-1 to cl.

SPFBKGD sp ---
Set sprite sp's priority to in-front of background.

SPBBKGD sp ---
Set sprite sp's priority to behind background.

SP-SPCOL ---
Read sprite to sprite collision detect register. Used BEFORE ?SP-SPCOL.

?SP-SPCOL sp --- f
Leaves a boolean flag indicating if sprite sp has collided with any other sprites. Used AFTER SP-SPCOL

SP-BCOL ---

Read sprite to background collision detect register. Used BEFORE ?SP-BCOL.

?SP-BCOL sp --- f

Leaves a boolean flag indicating if sprite sp has collided with the background. Used AFTER SP-BCOL

SPPOS x y sp ---

Set sprite sp to position x,y.

XINC , YINC --- addr

Arrays that hold direction vectors to interpret the joystick.

MOVL n sp ---
MOVR
MOVU Move sprite sp n units in the appropriate direction.
MOVD

MOVL - Left
MOVR - Right
MOVU - Up
MOVD - Down

SPADR n --- addr

Returns the address of sprite bank n. n runs from 0 to 239.

SPLOAD addr n ---

Load sprite bank n with data starting at location addr.
Typical use:

```
0 VARIABLE SPRITE1 45 , 45 , ... 23 ,

SPRITE1 1 SPLOAD
```

SPTAKE n sp ---

Set bank that sprite sp takes it's information from to n.

?FIRE --- f

Returns a boolean flag indicating if the fire button on control port 2 is pressed.

JOYSTICK --- yinx xinc

Returns the direction of the joystick on port 2 by increments in the x and y direction.

```
 0  0  = no motion      0  1 = move right
 0 -1  = move left      1  0 = move down
-1  0  = move up        ...
```

PCHAR x y char ---

Place character char in character position x,y on the graphics screen.

STRING addr c x y ---
Place string starting at location addr for c characters on graphics screen starting at character position x,y.

MUSIC DEFINITIONS

Music definitions can be accessed by loading screen no. 16 on your ASCI-FORTH disk.

There are five main definitions required to create music:

(1) RESET - Resets sound registers. (Turns off sound.) After you reset the computer (RUN/STOP - RESTORE) you must type RESET before music will work.

(2) COMPOSE name - Start a music definition called "name".

(3) FINI - End music definition.

(4) SPEED - A variable that contains the speed of play.

(5) n VOICE - Start definition of music for voice # n. (1 <= n <= 3)

The general structure of a music composition is as follows:

```
          (1)
COMPOSE name (2)  VOICE {note octave duration} {note octave duration}
          (3)

                                (1)
{note octave duration} ... (2) VOICE {note octave duration}
                                (3)

                                                        (1)
 note octave duration} {note octave duration} ... (2) VOICE
                                                        (3)

 note octave duration}  note octave duration} ... FINI
```

NOTE: All three voices are NOT required.

POSSIBLE NOTES

```
C
C# (Same as D flat)
D
D# (Same as E flat)
E
F
F# (Same as G flat)
G
G# (Same as A flat)
A
A# (Same as B flat)
B
```

OCTAVES : 0,1,2,3,4,5,6,7 middle C is octave 4

POSSIBLE DURATIONS

```
  Q  - quarter note
  ET - eighth note
  Q. - dotted quarter note
  H  - half note
  W  - whole note
  S  - sixteenth note
n ETS - the sum of n eighth notes (0 < n < 32)
n  SS - the sum of n sixteenth notes (0 < n < 63)
```

If a rest is required you can use the definition REST inplace of note and octave}.

APPENDIX A - STARTING UP

FORTH PROGRAMS:

On the disk you received you will find two programs FORTH and FORTH4. FORTH is the standard Forth language that the normal user uses, FORTH4 is for use when BASIC-4 is in use.

To start FORTH, take the following steps:

1. Turn off the computer
2. Insert the "key" that comes with the disk in the cassette part at the back of the Commodore-64
3. Turn on the computer
4. type: LOAD "FORTH",8
   (instead of FORTH you might want FORTH4).
5. After program has located type RUN.

To get a list of the Forth screens on your ASI-FORTH disk type:

```
4 46 INDEX
```

The program disk comes with the following screens.

4 LIST Lists the error messages. (You cannot load this screen)

5 LOAD Loads the Floating Point screen. (See instructions)

9 LOAD Loads 6502 assembler for Forth. (See instructions)

14 LOAD Loads utilities.

(1) n1 n2 n3 BACKUP

Back's up screens form one disk to another, or to the same disk but on a different screen. One should always keep a backup copy of important work in case you erase it.

The command would be: 4 46 4 BACKUP

n1 - Starting screen# of source.

n2 - Ending screen# of source.

n3 - Starting screen# of destination.

IMPORTANT: The first thing you should do when you get ASI-FORTH, is to backup the source programs (by loading and saving) and then load this screen and backup the screens on this disk.

(2) SHOW forth-word

Tries to display a "forth-word's" definition. This does not work with words that are defined in machine language.

16 LOAD Loads Music definitions. (For a description of them see appendix B)

19 LOAD Loads the music for ODE TO JOY. (Load after 16) Type ODE-TO-JOY to play it.

21 LOAD Loads the music for GREENSLEEVES. (Load after 16). Type GREENSLEEVES to play it.

23 LOAD Loads the music for JESU, JOY OF MANS DESIRING. Type JOY to play it. (Load after 16)

29 LOAD Loads the music for ALSO SPRACH ZARATHUSTRA. Type SPRACH to play it. (Load after 16)

31 LOAD Loads the music for MOZART'S #40. (Load after 16) Type MOZART-#40 to play it.

37 LOAD Loads the graphics definitions.

See page 40.

42 LOAD Loads a game using the graphic definitions. Screen 37 has to be loaded first.

To play just type GAME and use a joystick in control port #2 to destroy as many ships as possible.

46 LOAD By loading this screen you make all the definitions currently in Forth as part of the Forth program that can be saved from Basic by SAVE "prog",8

Example:

To make Forth always have floating point definitions upon boot-up, the following sequence is performed:

```
5 LOAD      ( Load floating point definitions)
46 LOAD     ( Fudge memory locations)
BYE         ( Return to Basic)
SAVE "float",8 ( Save Forth under new name)
```

When you load "float" and run it, Forth will boot-up with the floating point definitions already available.

APPENDIX B - DISK NOTES

Logical unit numbers 8 and 9 are used for disk I/O and should not be used for regular programming. SET-DRIVE must be executed after changing which drive you are using. If that disk seems to hang, break Forth by hitting RUN/STOP RESTORE and try executing SET-DRIVE, if this doesn't work you have a problem with your drive or you have over written the Forth program.

To open a channel with no command or file name enter:

log Dev Chan OPEN " "

When you get your ASI-FORTH you should copy screens 4 to 46 from the disk to a fresh disk, using the BACKUP utility contained on screen 14.

FORTH DATA DISK:

Disks that are used for storing FORTH screens must be initialized before use. NEVER store regular programs on a FORTH data disk.

DISCLAIMER NOTICE

Much care has been taken in preparing this manual, and the software programs. However, ACCELERATED SOFTWARE INC. makes no expressed or implied warranty of any kind with regard to these programs nor the documentation in these manuals. In no event shall ACCELERATED SOFTWARE INC. be liable for incidental or consequential damages in connection with or arising out of the furnishing, performance or use of any of these programs or manuals. ACCELERATED SOFTWARE INC. reserves the right to alter or update any program, publication or manual from time to time without obligation to notify any person of such changes.

Other ASI® products for the Commodore 64®

SPIDER AND THE FLY : The world of insects is a hard world, where one has to move fast to stay alive! Weave a web around your elusive prey — but beware of the dangers that stalk you!
MULTI-LEVEL, MULTI-PLAYER CAPABILITY.

Q-BOPPER : 3D ACTION on a cubic structure. Bop the cubes to change their colour. Beware! Killer spheres abound. Race the clock and hit magic squares for extra points.
MULTI-LEVEL, ARCADE GRAPHICS.

CHOMPER : Chomp your way through varieties of fruit while being chased by killer mushrooms, preying birds and free roaming deadly pumpkins. Enter and exit through ever changing passages that lead to new plentiful gardens.
MULTI-LEVEL, MULTI-MAZE.

EXCITING NEW RELEASES For the Commodore 64®

SPACE WARS : All new, 3D ACTION in the far reaches of OUTER SPACE. Three completely different battle scenes. Battle around planets, through outer space and finally in the death tunnel! It's the closest feeling to being in space itself!!
MULTI-LEVEL, ARCADE ACTION.

BALLS : A revolution in Arcade games! Over 20 different game concepts involving a cube and a sphere. Each is multi-level, each is a complete challenge. Why buy just one game — buy an arcade!
MULTI-CONCEPT, MULTI-LEVEL, ARCADE ACTION.

THE DUNGEONS OF BA : First there were word adventure games. Then there were graphics adventure games. Now, for the first time, ASI presents an action adventure game with arcade graphics and arcade speed!
MULTI-SCREENS WITH CONTINUOUS ACTION. THE CHALLENGE OF A PUZZLE COMBINED WITH THE ACTION OF AN ARCADE!

CASTLE OF JAZOOM : A game in the spirit of THE DUNGEONS OF BA. A computer experience which combines the mystery of an adventure game, with the action of an arcade. A completely new puzzle, filled with new dangers and new challenges!

FINANCIAL FORECASTER : 6 programs in 1 package! A complete homeuser's aid. Balance your budget, analyse cash flow, keep track of your banking, evaluate different house purchase scenarios, obtain payment schedules, amortization tables and more! All for one low price.

ACCELERATED SOFTWARE INC., P.O. Box 129, Station "A", Scarborough, Ontario M1K 5B9
ASI® logo is a registered trademark of ACCELERATED SOFTWARE INC./ASI® est une marque déposée d'Accelerated Software Inc.
©1984 All Rights reserved/Tous droits réservés.
Printed in Canada/Imprimé au Canada.