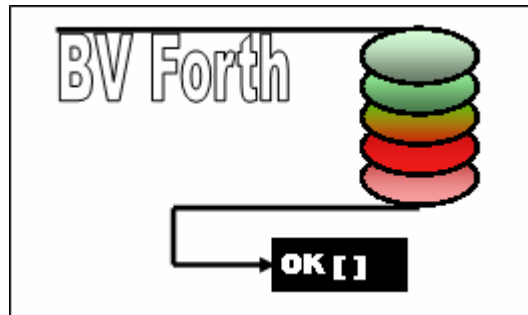


BV Forth (ARM) Core Glossary



ByVac

Contents

1. GLOSSARY 3
2. FUNCTIONAL CROSS REFERENCE..... 4
3. REVISIONS 43
4. APPENDIX B RESOURCES..... 44

Copyright in this work is vested in ByVac and the document is issued in confidence for the purpose only for which it is supplied. It must not be introduced in whole or in part or used for tendering or manufacturing purposes except under an agreement or with the consent in writing of ByVac and then only on condition that this notice is included in any such reproduction.

© Copyright ByVac 2007

No warranty

THE WORK IS PROVIDED "AS IS," AND COMES WITH ABSOLUTELY NO WARRANTY, EXPRESS OR IMPLIED, TO THE EXTENT PERMITTED BY APPLICABLE LAW, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Disclaimer of liability

IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS WORK, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1. Glossary

The glossary is the main 'instruction set' for Forth. This glossary lists all of the words in the core. For the purposes of this section the stack notation as laid down in the ANSI Forth standard will be followed, however for any other words outside of the ANSI standard the following notation will be used.

1.1. Stack Notation

The stack is critical to the operation of Forth and this is annotated in the glossary by showing the stack before the word is executed and after it has been executed thus:

```
( n1,n2 --- n3)          +      // adds two values
```

The two items, in this case n1 and n2 are on the stack before the word '+' is executed, after it is executed, just one value (n3) remains on the stack which is the result of n1 + n2, so the dashes '--' separate the before and after states of the stack.

The ANSI standard, that can be found on the net describes the values on the stack in terms of what they are, rather than what they do. The notation is given in the table at 1.2.1.

Some BV-Forth words are more descriptive the philosophy being to use something in the stack that best communicates what the word requires and what it does to the stack after, reinforce this with the word description and wherever possible an example of how to use the word.

```
//      ( day --- )
: DayOfTheWeek
  DOW 3 +
  DUP C@ EMIT 1+ DUP C@ EMIT C@ EMIT
;
```

1.2. CORE Glossary

The following is a glossary of words in the core Forth system, as most of the words follow the ANSI standard, this is the format used. Any references used in this section of the document refer to the ANSI Standard.

The standard can be found on the Fig Forth site: www.forth.org Reference numbers used outside of the section numbers of this text refer to the standard.

*** 1 Cell = 4 bytes.

1.2.1.Symbol	Data type	Size on stack
flag	flag	1 cell
true	true flag	1 cell
false	false flag	1 cell
char	character	1 cell
n	signed number	1 cell
+n	non-negative number	1 cell
u	unsigned number	1 cell
n u	(1) number	1 cell
x	unspecified cell	1 cell
xt	execution token	1 cell
addr	address	1 cell
a-addr	aligned address	1 cell
c-addr	character-aligned address	1 cell
d	double-cell signed number	2 cells

+d	double-cell non-negative number	2 cells
ud	double-cell unsigned number	2 cells
xd	unspecified cell pair	2 cells
colon-sys	definition compilation	implementation dependent
do-sys	do-loop structures	implementation dependent
case-sys	CASE structures	implementation dependent
of-sys	OF structures	implementation dependent
orig	control-flow origins	implementation dependent
dest	control-flow destinations	implementation dependent
loop-sys	loop-control parameters	implementation dependent
nest-sys	definition calls	implementation dependent
i*x, j*x, k*x(3)	any data type	0 or more cells

2. Functional Cross Reference

This is a summary of words in the core, grouped by function. The full definition of the word is contained within the glossary. As some words span more than one function, they may be repeated for different groups.

CORE is the ANSI standard words and BV-CORE are the necessary extensions to this standard.

2.1.1. Words and Word creation

Word			Page
'	"tick"	CORE	7
;	"semicolon"	CORE	12
:	"colon"	CORE	12
;S	"semicolon S"	BV-CORE	12
?'	"query tick"	BV-CORE	16
CELL+	"cell-plus"	CORE	21
CELLS		CORE	21
CREATE		CORE	23
FIND		CORE	27
["left-bracket"	CORE	43
[']	"bracket-tick"	CORE	43
]	"right-bracket"	CORE	43
IMMEDIATE		CORE	29
LATEST	"latest"	BV-CORE	30
WORD		CORE	42

2.1.2. Maths

Word			Page
-	"minus"	CORE	7
*	"star"	CORE	8
*/	"star-slash"	CORE	8
*/MOD	"star-slash-mod"	CORE	8
/MOD	"slash-mod"	CORE	10
+	"plus"	CORE	9
1+	"one-plus"	CORE	11
1-	"one-minus"	CORE	11
2*	"two-star"	CORE	11
2/	"two-slash"	CORE	11
2-	"two minus"	BV-CORE	16
2+	"two plus"	BV-CORE	16
ABS	"abs"	CORE	18
D-	"double minus"	BV-CORE	23
D+	"double plus"	BV-CORE	23
FM/MOD	"f-m-slash-mod"	CORE	27

INVERT		CORE	29
M*	"m-star"	CORE	31
MOD		CORE	32
NEGATE		CORE	32
S>D	"s-to-d"	CORE	36
UM*	"u-m-star"	CORE	40
U/MOD	"u slash mod"	BV-CORE	40
UM/MOD	"u-m-slash-mod"	CORE	40
X**y	"x to the power of y"	BV-CORE	42

2.1.3. Memory

Word			Page
!	"store"	CORE	7
?@	"query fetch"	BV-CORE	17
?C@	"query C fetch"	BV-CORE	17
@	"fetch"	CORE	17
+!	"plus-store"	CORE	9
2!	"two-store"	CORE	11
2@	"two fetch"	BV-CORE	11
2R@	"two R fetch"	BV-CORE	16
C!	"c-store"	CORE	20
C@	"c-fetch"	CORE	21
CELL+	"cell-plus"	CORE	21
CELLS		CORE	21
CHAR	"char"	CORE	21
CHAR+	"char-plus"	CORE	21
CHARS	"chars"	CORE	21
CONSTANT		CORE	22
DUMP	"dump"	BV-CORE	25
DUMP1	"dump one"	BV-CORE	25
FILL		CORE	27
INTEGER	"integer"	BV-CORE	29
MOVE		CORE	32
VARIABLE		CORE	41
VP	"variable pointer"	BV-CORE	41
VSPACE\$	"vspace"	BV-CORE	41

2.1.4. Numbers

Word			Page
?NUMBER	"query-number"	BV-CORE	17
>DIGIT	"to digit"	BV-CORE	14
0	"zero"	BV-CORE	15
1	"one"	BV-CORE	15
-1	"minus one"	BV-CORE	15
2	"two"	BV-CORE	16
-2	"minus two"	BV-CORE	16
BASE	"Base"	CORE	19
SEED	"seed"	BV-CORE	37
	(-- addr)		
	Returns the address of the random number seed, at reset this will be 0 and can be changed similar to any other variable.		
	See RND		
SIGN		CORE	
SIGN		CORE	37
S>D	"s-to-d"	CORE	36

2.1.5. Number Formatting

Word			Page
#	"number-sign"	CORE	7
#>	"number-sign-greater"	CORE	7
#S	"number-sign-s"	CORE	7
(BASE.)	"brack base dot"	BV-CORE	8
<#	"less-number-sign"	CORE	13
HOLD		CORE	28

2.1.6. Input & Output

Word			Page
#TIB	"hash-tib"	BV-CORE	7
.	"dot"	CORE	9
.S	"dot S"	BV-CORE	10
BASE	"Base"	CORE	19
BINARY	"binary"	BV-CORE	20
DECIMAL		CORE	23
EMIT		CORE	25
EMIT1	"Emit 1"	BV-CORE	25
HEX	"hex"	BV-CORE	28
KEY	"key"	CORE	29
KEY?	"key query"	BV-CORE	29
KEY1	"key1"	BV-CORE	30
KEY1?	"key one query"	BV-CORE	30
PB.	"print binary"	BV-CORE	33
PD.	"print decimal"	BV-CORE	33
PH.	"print hex"	BV-CORE	33
SEED	"seed"	BV-CORE	37
	(-- addr)		
	Returns the address of the random number seed,		

	at reset this will be 0 and can be changed similar to any other variable.		
	See RND		
SIGN		CORE	
SIGN		CORE	37
TIB	"text input buffer"	BV-CORE	39
U.	"u-dot"	CORE	40
UART	"uart"	BV-CORE	40

2.1.7. Looping & Branching

Word			Page
+LOOP	"plus-loop"	CORE	9
OBRANCH	"zero branch"	BV-CORE	15
ABORT	"Abort"	CORE	18
ABORT"	"abort-quote"	CORE	18
AGAIN	"again"	BV-CORE	18
BEGIN	"Begin"	CORE	19
BRANCH	"branch"	BV-CORE	20
DO		CORE	24
ELSE		CORE	25
ESCAPE	"escape"	BV-CORE	26
EXIT		CORE	26
FOR	"for"	BV-CORE	27
I		CORE	28
IF		CORE	29
J		CORE	29
LEAVE		CORE	30
LOOP		CORE	31
NEXT	"next"	BV-CORE	33
REPEAT		CORE	36
THEN		CORE	39
UNLOOP		CORE	40
UNTIL		CORE	41
WHILE		CORE	41

2.1.8. Dictionary

Word			Page
,	"comma quote"	BV-CORE	9
,	"comma"	CORE	8
ALIGN	"Align"	CORE	18
ALLOT	"Allot"	CORE	19
C,	"c-comma"	CORE	21
CELL+	"cell-plus"	CORE	21
CELLS		CORE	21
CHAR	"char"	CORE	21
CHAR+	"char-plus"	CORE	21
CHARS	"chars"	CORE	21
CLEAR	"clear"	BV-CORE	22
DP	"dictionary pointer"	BV-CORE	25
HERE		CORE	28
S"	"s-quote"	CORE	36

2.1.9. Logic and Comparison

Word			Page
<	"less-than"	CORE	13
<>	"not equals"	BV-CORE	13
=	"equals"	CORE	13
>	"greater-than"	CORE	13
0<	"zero-less"	CORE	10

0=	"zero-equals"	CORE	10
AND	"And"	CORE	19
BETWEEN	"between"	BV-CORE	20
BIC	"bit clear"	BV-CORE	20
INVERT		CORE	29
LSHIFT	"l-shift"	CORE	31
LSHIFTC	"left shift with carry"	BV-CORE	31
NEGATE		CORE	32
OR		CORE	33
RSHIFT	"r-shift"	CORE	36
RSHIFTC	"right shift carry"	BV-CORE	35
U<	"u-less-than"	CORE	40
XOR	"x-or"	CORE	42

2.1.10. Buffers

Word			Page
#TIB	"hash-tib"	BV-CORE	7
PAD	"pad"	BV-CORE	33
TIB	"text input buffer"	BV-CORE	39

2.1.11. Strings

Word			Page
."	"dot-quote"	CORE	10
[CHAR]	"bracket-char"	CORE	43
=>	To Integer (n ---)	CORE	14
<p>This is used to set integer and local variable values, it will take a value from the stack and place it in the next integer encountered.</p> <p>Example: 123 => foo</p> <p>This will set foo to the value of 123, where foo has been defined as an integer.</p> <p>See INTEGER, +>, %@</p>			
2.1.12.			
>		Update	
(n ---)			
<p>Updates the following integer with the value on the stack, used for incrementing or decrementing an integer.</p> <p>Example: 123 => foo</p> <p>This will set foo to the value of 123, where foo has been defined as an</p>			

integer.

See INTEGER, +>, %@

2.1.13.

CK

"ackn

(-- addr)

Returns the address of where the ACK value is stored.

This is used in the BV loading protocol, the value stored at this address will be sent at the end of each line when the word FLOAD is used. By default this is 6 but can be changed to any other character value. To change to 3 for example use 3 ACK !. The value can also be viewed by using ACK @.

See FLOAD, ;S

>COUNT	"to count"	BV-CORE	
>COUNT	"to count"	BV-CORE	14
>DIGIT	"to digit"	BV-CORE	14
>IN	"to-in"	CORE	14
ACCEPT	"Accept"	CORE	18
BL	"b-l"	CORE	20
COUNT		CORE	23
CR	"c-r"	CORE	23
HEXOUT	"hex out"	BV-CORE	28
LENGTH	"length"	BV-CORE	30
S"	"s-quote"	CORE	36
S=	"string compare"	BV-CORE	36
SKIPBLANKS	"skipblanks"	BV-CORE	37
SOURCE		CORE	38
SPACE		CORE	38
SPACES		CORE	38
STYPE	"stype"	BV-CORE	38
TOUPPER	"to upper"	BV-CORE	39
TYPE		CORE	39

2.1.14. Stack

Word			Page
?DUP	"question-dupe"	CORE	17
+UNDER	"plus under"	BV-CORE	9
>L	"to L"	BV-CORE	15
>R	"to-r"	CORE	17
2DROP	"two-drop"	CORE	11
2DUP	"two-dupe"	CORE	11
2OVER	"two-over"	CORE	12
2SWAP	"two-swap"	CORE	12
DEPTH		CORE	23
DROP		CORE	24
DUP	"dupe"	CORE	25
L>	"from leave"	BV-CORE	30
LITERAL		CORE	30

OVER		CORE	33
PICK	"pick"	BV-CORE	34
R>	"r-from"	CORE	35
R@	"r-fetch"	CORE	35
ROT	"rote"	CORE	36
SP@	"stack pointer fetch"	BV-CORE	38
SWAP		CORE	38

2.1.15. System

Word			Page
?SECTOR	"query sector"	BV-CORE	17
BLANKCHECK	"blankcheck"	BV-CORE	20
CCLK	"clock"	BV-CORE	21
CHUNKS	"chunks"	BV-CORE	22
CMD!	"command store"	BV-CORE	22
CMD@	"command fetch"	BV-CORE	22
COLD	"cold"	BV-CORE	22
DEVICEID	"device ID"	BV-CORE	24
EXECUTE		CORE	26
EXITADDRESS	"exit address"	BV-CORE	26
FERASE	"flash erase"	BV-CORE	27
FOSC	"crystal frequency"	BV-CORE	27
FREERAM	"free ram"	BV-CORE	28
FWRITE	"flash write"	BV-CORE	28
LOADF	"load forth"	BV-CORE	31
MEM-RANGE	"memory range"	BV-CORE	32

NEW	"new"	BV-CORE	32
NEWSTART	"new flash start"	BV-CORE	33
PSIZE	"program size"	BV-CORE	34
QUIT		CORE	34
RAMSIZE	"ramsize"	BV-CORE	34
RAMTOP	"ramtop"	BV-CORE	35
RND	"rand"	BV-CORE	35
	(-- n)		
	Returns a 32bit random number.		
	See SEED		
RSHASH	"rs-hash"	BV-CORE	
RSHASH	"rs-hash"	BV-CORE	35
SAVE	"save"	BV-CORE	37
SECTOR	"sector"	BV-CORE	37
STATE		CORE	38
SYS!	"sys store"	BV-CORE	39
SYS@	"sys fetch"	BV-CORE	39
WHASH	"w hash"	BV-CORE	41
WRITE512	"write 512"	BV-CORE	42
WSIZE	"wsize"	BV-CORE	42

2.1.16. ' "tick" CORE

("<spaces>name" -- xt)

Skip leading space delimiters. Parse name delimited by a space. Find name and return xt, the execution token for name. An ambiguous condition exists if name is not found. When interpreting, ' xyz EXECUTE is equivalent to xyz.

See: 3.4 The Forth text interpreter, 3.4.1 Parsing, A.2.1.188 POSTPONE,

A.2.1.249 ['], D.6.7 Immediacy.

2.1.17. - "minus" CORE

(n1|u1 n2|u2 -- n3|u3)

Subtract n2|u2 from n1|u1, giving the difference n3|u3.

See: 3.3.3.1 Address alignment.

2.1.18. ! "store" CORE

(x a-addr --)

Store x at a-addr.

See: 3.3.3.1 Address alignment.

2.1.19. # "number-sign" CORE

(ud1 -- ud2)

Divide ud1 by the number in BASE giving the quotient ud2 and the remainder n. (n is the least-significant digit of ud1.) Convert n to external form and add the resulting character to the beginning of the pictured numeric output string. An ambiguous condition exists if # executes outside of a <# #> delimited number conversion.

See: 2.1.20 #>, 2.1.21 #S, 2.1.56 <#.

2.1.20. #> "number-sign-greater" CORE

(xd -- c-addr u)

Drop xd. Make the pictured numeric output string available as a character string. c-addr and u specify the resulting character string. A program may replace characters within the string.

See: 2.1.19 #, 2.1.21 #S, 2.1.56 <#.

2.1.21. #S "number-sign-s" CORE

(ud1 -- ud2)

Convert one digit of ud1 according to the rule for #. Continue conversion until the quotient is zero. ud2 is zero. An ambiguous condition exists if #S executes outside of a <# #> delimited number conversion.

See: 2.1.19 #, 2.1.20#>, 2.1.56 <#.

2.1.22. #TIB "hash-tib" BV-CORE

(-- u)

Returns the size of the Text Input Buffer in bytes

2.1.23. (BASE.) "brack base dot" BV-CORE

(value base --)

This word is used for printing a number on the stack to any base value. It will temporarily change the BASE in order to print the value give.

Example 12 16 (BASE.) will output &C

2.1.24. ("paren" ** Not Implemented ** CORE

Compilation: Perform the execution semantics given below.

Execution: ("ccc<paren>" --)

Parse ccc delimited by) (right parenthesis). (is an immediate word. The number of characters in ccc may be zero to the number of characters in the parse area.

[This is not part of the BV Forth core, // is used instead](#)

See: 3.4.1 Parsing, 11.2.1.22 (.

2.1.25. * "star" CORE

(n1|u1 n2|u2 -- n3|u3)

Multiply n1|u1 by n2|u2 giving the product n3|u3.

2.1.26. */ "star-slash" CORE

(n1 n2 n3 -- n4)

Multiply n1 by n2 producing the intermediate double-cell result d. Divide d by n3 giving the single-cell quotient n4. An ambiguous condition exists if n3 is zero or if the quotient n4 lies outside the range of a signed number. If d and n3 differ in sign, the implementation-defined result returned will be the same as that returned by either the phrase >R M* R> FM/MOD SWAP DROP or the phrase >R M* R> SM/REM SWAP DROP.

See: 3.2.2.1 Integer division.

2.1.27. */MOD "star-slash-mod" CORE

(n1 n2 n3 -- n4 n5)

Multiply n1 by n2 producing the intermediate double-cell result d. Divide d by n3 producing the single-cell remainder n4 and the single-cell quotient n5. An ambiguous condition exists if n3 is zero, or if the quotient n5 lies outside the range of a single-cell signed integer. If d and n3 differ in sign, the implementation-defined result returned will be the same as that returned by either the phrase >R M* R> FM/MOD or the phrase >R M* R> SM/REM.

See: 3.2.2.1 Integer division.

2.1.28. , "comma" CORE

(x --)

Reserve one cell of data space and store x in the cell. If the data- space pointer is aligned when , begins execution, it will remain aligned when , finishes execution. An ambiguous condition exists if the data- space pointer is not aligned prior to execution of ,.

See: 3.3.3 Data space, 3.3.3.1 Address alignment.

2.1.35. ." **"dot-quote" CORE**

Interpretation: Interpretation semantics for this word are undefined.

Compilation: ("ccc<quote>" --)

Parse ccc delimited by " (double-quote). Append the run-time semantics given below to the current definition.

Run-time: (--)

Display ccc.

See: 3.4.1 Parsing, 6.2.0200 .(.

2.1.36. .S **"dot S" BV-CORE**

(n1 --)

Prints out a the stack contents inside square brackets, i.e. [1 2 3] this is used as part of the prompt and is useful for debugging.

2.1.37. / **"slash" CORE**

(n1 n2 -- n3)

Divide n1 by n2, giving the single-cell quotient n3. An ambiguous condition exists if n2 is zero. If n1 and n2 differ in sign, the implementation-defined result returned will be the same as that returned by either the phrase >R S>D R> FM/MOD SWAP DROP or the phrase >R S>D R> SM/REM SWAP DROP.

See: 3.2.2.1 Integer division.

2.1.38. // **"comment" BV-CORE**

(--)

All text following this will be ignored until the next newline.

2.1.39. /MOD **"slash-mod" CORE**

(n1 n2 -- n3 n4)

Divide n1 by n2, giving the single-cell remainder n3 and the single-cell quotient n4. An ambiguous condition exists if n2 is zero. If n1 and n2 differ in sign, the implementation-defined result returned will be the same as that returned by either the phrase >R S>D R> FM/MOD or the phrase >R S>D R> SM/REM.

See: 3.2.2.1 Integer division.

2.1.40. 0< **"zero-less" CORE**

(n -- flag)

flag is true if and only if n is less than zero.

2.1.41. 0= **"zero-equals" CORE**

(x -- flag)

flag is true if and only if x is equal to zero.

2.1.42. **1+** **"one-plus" CORE**

(n1|u1 -- n2|u2)

Add one (1) to n1|u1 giving the sum n2|u2.

2.1.43. **1-** **"one-minus" CORE**

(n1|u1 -- n2|u2)

Subtract one (1) from n1|u1 giving the difference n2|u2.

2.1.44. **2!** **"two-store" CORE**

(x1 x2 a-addr --)

Store the cell pair x1 x2 at a-addr, with x2 at a-addr and x1 at the next consecutive cell. It is equivalent to the sequence SWAP OVER ! CELL+ !.

See: 3.3.3.1 Address alignment.

2.1.45. **2*** **"two-star" CORE**

(x1 -- x2)

x2 is the result of shifting x1 one bit toward the most-significant bit, filling the vacated least-significant bit with zero.

2.1.46. **2/** **"two-slash" CORE**

(x1 -- x2)

x2 is the result of shifting x1 one bit toward the least-significant bit, leaving the most-significant bit unchanged.

2.1.47. **2@** **"two-fetch" CORE**

(a-addr -- x1 x2)

Fetch the cell pair x1 x2 stored at a-addr. x2 is stored at a-addr and x1 at the next consecutive cell. It is equivalent to the sequence DUP CELL+ @ SWAP @.

See: 3.3.3.1 Address alignment, 2.1.44 2!.

2.1.48. **2DROP** **"two-drop" CORE**

(x1 x2 --)

Drop cell pair x1 x2 from the stack.

2.1.49. **2DUP** **"two-dupe" CORE**

(x1 x2 -- x1 x2 x1 x2)

Duplicate cell pair x1 x2.

2.1.55. < "less-than" CORE

(n1 n2 -- flag)

flag is true if and only if n1 is less than n2.

See: 2.1.231 U<.

2.1.56. <# "less-number-sign" CORE

(--)

Initialize the pictured numeric output conversion process.

See: 2.1.19 #, 2.1.20 #>, 2.1.21 #S.

2.1.57. <> "not equals" BV-CORE

(n1 n2 -- flag)

flag is true if n1 does not equal n2

2.1.58. = "equals" CORE

(x1 x2 -- flag)

flag is true if and only if x1 is bit-for-bit the same as x2.

2.1.59. > "greater-than" CORE

(n1 n2 -- flag)

flag is true if and only if n1 is greater than n2.

See: 6.2.2350 U>.

2.1.60. => To Integer CORE

(n ---)

This is used to set integer and local variable values, it will take a value from the stack and place it in the next integer encountered.

Example:

123 => foo

This will set foo to the value of 123, where foo has been defined as an integer.

See INTEGER, +>, %@

2.1.61. +> Update Integer CORE

(n ---)

Updates the following integer with the value on the stack, used for incrementing or decrementing an integer.

Example:

123 => foo

This will set foo to the value of 123, where foo has been defined as an integer.

See INTEGER, +>, %@

2.1.62. ACK "acknowledge" BV-CORE

(-- addr)

Returns the address of where the ACK value is stored.

This is used in the BV loading protocol, the value stored at this address will be sent at the end of each line when the word FLOAD is used. By default this is 6 but can be changed to any other character value. To change to 3 for example use 3 ACK !. The value can also be viewed by using ACK @.

See FLOAD, ;S

2.1.63. >COUNT "to count" BV-CORE

(s-addr --)

By default BV Forth uses zero terminated strings but some ANSI words require traditional counted strings. This converts a 0 terminated string into a counted string.

NOTE this changes the bytes at the address given so will not work in Flash.

2.1.64. >DIGIT "to digit" BV-CORE

(n1 - n2)

Converts the value n1 to an ACSII value n2 that represents the ASCII code for that digit. For example if n1=1 then n2 would be converted to 49. No error checking is implemented and so numbers outside the range 0 to 16 (0..F) will return unpredictable values.

2.1.65. >BODY "to-body" ** Not Implemented ** CORE

(xt -- a-addr)

a-addr is the data-field address corresponding to xt. An ambiguous condition exists if xt is not for a word defined via CREATE.

See: 3.3.3 Data space.

2.1.66. >IN "to-in" CORE

(-- a-addr)

a-addr is the address of a cell containing the offset in characters from the start of the input buffer to the start of the parse area.

2.1.67. >NUMBER "to-number" ** Not Implemented ** CORE

(ud1 c-addr1 u1 -- ud2 c-addr2 u2)

ud2 is the unsigned result of converting the characters within the string specified by c-addr1 u1 into digits, using the number in BASE, and adding each into ud1 after multiplying ud1 by the number in BASE.

Conversion continues left-to-right until a character that is not convertible, including any "+" or "-", is encountered or the string is entirely converted. c-addr2 is the location of the first unconverted character or the first character past the end of the string if the string was entirely converted. u2 is the number of unconverted

characters in the string. An ambiguous condition exists if ud2 overflows during the conversion.

Not implemented see 2.1.68 instead

2.1.68. **>L** **"to L" BV-CORE**

(n|u --)

Places a value on the leave stack. The leave stack is used by the looping words and so care must be taken to not only balance this stack but not use it across the boundaries of a loop construct.

2.1.69. **0** **"zero" BV-CORE**

(-- 0)

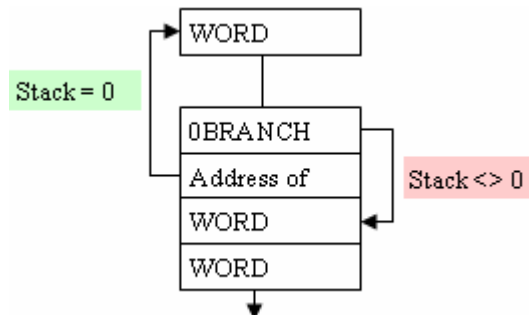
Places 0 on the stack, this is defined as a constant for efficiency purposes.

2.1.70. **OBRANCH** **"zero branch" BV-CORE**

(n1 --)

Fundamental part of branching used in IF, THEN and BEGIN type constructs. The word following OBRANCH will be an address into some other part of the word, back to BEGIN for example.

If n1 is 0 it will simply drop down to this following address and branch there, if not execution will continue after this branch address.



2.1.71. **1** **"one" BV-CORE**

(-- 1)

This is defened as a constant, it makes Forth words more efficient.

2.1.72. **-1** **"minus one" BV-CORE**

(-- 1)

This is defened as a constant, it makes Forth words more efficient. Do not confuse this with 1-

2.1.81. ?@ "query fetch" BV-CORE

(a-addr1 - a-addr1 | contents)

The ARM processor throws an exception if an address is accessed that is not available, to prevent this ?@ will first check the address, if it is allowed then the contents of the address will be fetched, otherwise the original address will be returned.

2.1.82. ?C@ "query C fetch" BV-CORE

(a-addr1 - a-addr1 | byte)

The ARM processor throws an exception if an address is accessed that is not available, to prevent this ?C@ will first check the address, if it is allowed then a single byte will be fetched at the address given, otherwise the original address will be returned.

2.1.83. ?NUMBER "query-number" BV-CORE

(addr1 - n1 0|-1)

Addr1 is the address of a 0 terminated string, if it is a valid number then that value is returned along with a -1. If it is not a valid number 0 is returned.

Valid number prefixes can be +, -, &, b or B representing decimal (without a prefix) Hex and Binary respectively.

2.1.84. ?SECTOR "query sector" BV-CORE

(-- n)

This gives the first free sector of Flash that can be written to .

2.1.85. >R "to-r" CORE

Interpretation: Interpretation semantics for this word are undefined.

Execution: (x --) (R: -- x)

Move x to the return stack.

See: 3.2.3.3 Return stack, 2.1.202 R>, 2.1.203 R@, 6.2.0340 2>R, 6.2.0410 2R>, 6.2.0415 2R@.

2.1.86. ?DUP "question-dupe" CORE

(x -- 0 | x x)

Duplicate x if it is non-zero.

2.1.87. @ "fetch" CORE

(a-addr -- x)

x is the value stored at a-addr.

See: 3.3.3.1 Address alignment.

2.1.88. ABORT "Abort" CORE

(i*x --) (R: j*x --)

Empty the data stack and perform the function of QUIT, which includes emptying the return stack, without displaying a message.

See: 9.6.2.0670 ABORT.

2.1.89. ABORT" "abort-quote" CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: ("ccc<quote>" --)

Parse ccc delimited by a " (double-quote). Append the run-time semantics given below to the current definition.

Run-time: (i*x x1 -- | i*x) (R: j*x -- | j*x)

Remove x1 from the stack. If any bit of x1 is not zero, display ccc and perform an implementation-defined abort sequence that includes the function of ABORT.

See: 3.4.1 Parsing, 9.6.2.0680 ABORT".

2.1.90. ABS "abs" CORE

(n -- u)

u is the absolute value of n.

2.1.91. ACCEPT "Accept" CORE

(c-addr +n1 -- +n2)

Receive a string of at most +n1 characters. An ambiguous condition exists if +n1 is zero or greater than 32,767. Display graphic characters as they are received. A program that depends on the presence or absence of non-graphic characters in the string has an environmental dependency. The editing functions, if any, that the system performs in order to construct the string are implementation-defined.

Input terminates when an implementation-defined line terminator is received. When input terminates, nothing is appended to the string, and the display is maintained in an implementation-defined way. +n2 is the length of the string stored at c-addr.

2.1.92. AGAIN "again" BV-CORE

(-- a-addr)

Compiles a jump to the address given on the stack, this address is usually left by BEGIN to form a continuous loop.

2.1.93. ALIGN "Align" CORE

(--)

If the data-space pointer is not aligned, reserve enough space to align it.

See: 3.3.3 Data space, 3.3.3.1 Address alignment.

2.1.99. BETWEEN "between" BV-CORE

(n1 n2 n3 -- flag)

If the value of n1 is between the values of n2 and n3 this will return true. NOTE that it will also return true if n1=n2 or n1=n3, also n2 and n3 can be any way round thus:

12 10 12 = true
 12 12 10 = true
 12 12 12 = true
 12 13 14 = flase

2.1.100. BIC "bit clear" BV-CORE

(value mask -- result)

This is a direct translation of the BIC ARM instruction, given a value it will clear the bits that are set in the mask thus:

&FF 2 BIC returns &FD

2.1.101. BINARY "binary" BV-CORE

(--)

Sets the BASE to 2

2.1.102. BL "b-l" CORE

(-- char)

char is the character value for a space.

2.1.103. BLANKCHECK "blankcheck" BV-CORE

(n1 - 8|0)

Checks to see if the given sector (n1) is blank, returns 8 if not blank and 0 if it is blank.

2.1.104. BRANCH "branch" BV-CORE

(--)

Fundamental part of branching used in IF, THEN and BEGIN type constructs. The word following BRANCH will be an address into some other part of the same word, back to BEGIN for example. Branch will jump to this address.

2.1.105. C! "c-store" CORE

(char c-addr --)

Store char at c-addr. When character size is smaller than cell size, only the number of low-order bits corresponding to character size are transferred.

See: 3.3.3.1 Address alignment

2.1.106. C, "c-comma" CORE

(char --)

Reserve space for one character in the data space and store char in the space. If the data-space pointer is character aligned when C, begins execution, it will remain character aligned when C, finishes execution. An ambiguous condition exists if the data-space pointer is not character-aligned prior to execution of C,.

See: 3.3.3 Data space, 3.3.3.1 Address alignment.

2.1.107. C@ "c-fetch" CORE

(c-addr -- char)

Fetch the character stored at c-addr. When the cell size is greater than character size, the unused high-order bits are all zeroes.

See: 3.3.3.1 Address alignment.

2.1.108. CCLK "clock" BV-CORE

(-- n1)

Returns the processor clock frequency in KHz, this can vary depending on the values set to the PLL. It is not the crystal frequency unless the PLL is switched off.

2.1.109. CELL+ "cell-plus" CORE

(a-addr1 -- a-addr2)

Add the size in address units of a cell to a-addr1, giving a-addr2.

See: 3.3.3.1 Address alignment.

2.1.110. CELLS CORE

(n1 -- n2)

n2 is the size in address units of n1 cells.

2.1.111. CHAR "char" CORE

("<spaces>name" -- char)

Skip leading space delimiters. Parse name delimited by a space. Put the value of its first character onto the stack.

See: 3.4.1 Parsing, 2.1.250 [CHAR].

2.1.112. CHAR+ "char-plus" CORE

(c-addr1 -- c-addr2)

Add the size in address units of a character to c-addr1, giving c-addr2.

See: 3.3.3.1 Address alignment.

2.1.113. CHARS "chars" CORE

(n1 -- n2)

n2 is the size in address units of n1 characters.

2.1.114. CHUNKS "chunks" BV-CORE

(n1 -- n2)

N2 is the number of 512 byte chunks required for n1:

n1	n2
5	1
100	1
512	1
513	2

2.1.115. CLEAR "clear" BV-CORE

(--)

Writes 0 to RAM from dictionary start to ramtop.

2.1.116. CMD! "command store" BV-CORE

(n1 --)

This word is used for building an IAP command, the command string is built into PAD. As PAD is used for other purposes unexpected results may occur when interpreting. N1 is stored as a command byte in preparation for In Application Programming.

2.1.117. CMD@ "command fetch" BV-CORE

(-- n1)

N1 is returned as a the result of an In Application Programming command.

2.1.118. COLD "cold" BV-CORE

(--)

Cold is the first word that is called after a hardware reset, it is responsible for setting up the system and sign on.

2.1.119. CONSTANT CORE

(x "<spaces>name" --)

Skip leading space delimiters. Parse name delimited by a space. Create a definition for name with the execution semantics defined below.name is referred to as a "constant".

name Execution: (-- x)

Place x on the stack.

See: 3.4.1 Parsing.

2.1.120. COUNT CORE

(c-addr1 -- c-addr2 u)

Return the character string specification for the counted string stored at c-addr1. c-addr2 is the address of the first character after c-addr1. u is the contents of the character at c-addr1, which is the length in characters of the string at c-addr2.

2.1.121. CR "c-r" CORE

(--)

Cause subsequent output to appear at the beginning of the next line.

2.1.122. CREATE CORE

("<spaces>name" --)

Skip leading space delimiters. Parse name delimited by a space. Create a definition for name with the execution semantics defined below. If the data-space pointer is not aligned, reserve enough data space to align it. The new data-space pointer defines name's data field. CREATE does not allocate data space in name's data field.

name Execution: (-- a-addr)

a-addr is the address of name's data field. The execution semantics of name may be extended by using DOES>.

See: 3.3.3 Data space, 2.1.129 DOES>.

2.1.123. D- "double minus" BV-CORE

(d1 d2 - d3)

Minus for double numbers, d3 is d1 - d2.

2.1.124. D+ "double plus" BV-CORE

(d1 d2 + d3)

Plus for double numbers, d3 is d1 + d2.

2.1.125. DECIMAL CORE

(--)

Set the numeric conversion radix to ten (decimal).

2.1.126. DEPTH CORE

(-- +n)

+n is the number of single-cell values contained in the data stack before +n was placed on the stack.

2.1.127. DEVICEID "device ID" BV-CORE

(-- n1)

Returns the device ID as a 32bit word value, see the Application Data sheet for the particular device, for an LPC2132 this will be 196369

2.1.128. DO CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: -- do-sys)

Place do-sys onto the control-flow stack. Append the run-time semantics given below to the current definition. The semantics are incomplete until resolved by a consumer of do-sys such as LOOP.

Run-time: (n1|u1 n2|u2 --) (R: -- loop-sys)

Set up loop control parameters with index n2|u2 and limit n1|u1. An ambiguous condition exists if n1|u1 and n2|u2 are not both the same type. Anything already on the return stack becomes unavailable until the loop-control parameters are discarded.

See: 3.2.3.2 Control-flow stack, 2.1.32 +LOOP, 2.1.171 LOOP.

2.1.129. DOES> CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: colon-sys1 -- colon-sys2)

Append the run-time semantics below to the current definition. Whether or not the current definition is rendered findable in the dictionary by the compilation of DOES> is implementation defined. Consume colon-sys1 and produce colon-sys2. Append the initiation semantics given below to the current definition.

Run-time: (--) (R: nest-sys1 --)

Replace the execution semantics of the most recent definition, referred to as name, with the name execution semantics given below. Return control to the calling definition specified by nest-sys1. An ambiguous condition exists if name was not defined with CREATE or a user-defined word that calls CREATE.

Initiation: (i*x -- i*x a-addr) (R: -- nest-sys2)

Save implementation-dependent information nest-sys2 about the calling definition. Place name's data field address on the stack. The stack effects i*x represent arguments to name.

name Execution: (i*x -- j*x)

Execute the portion of the definition that begins with the initiation semantics appended by the DOES> which modified name. The stack effects i*x and j*x represent arguments to and results from name, respectively.

See: 2.1.122 CREATE.

2.1.130. DROP CORE

(x --)

Remove x from the stack.

2.1.131. DP "dictionary pointer" BV-CORE

(-- a-addr)

Returns the address of the dictionary pointer. To get the value use @. HERE is defined as DP @

2.1.132. DUMP "dump" BV-CORE

(a-addr -- a-addr + 16)

Useful for experimenting with memory, dumps 16 bytes from the given address. The display is in two parts, the first part is the hex value and the second part is the character value.

2.1.133. DUMP1 "dump one" BV-CORE

(a-addr -- a-addr a-addr + 16)

Useful for experimenting with memory, dumps 16 bytes from the given address. This display shows 8 32bit words and is useful for debugging Forth words.

2.1.134. DUP "dupe" CORE

(x -- x x)

Duplicate x.

2.1.135. ELSE CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: orig1 -- orig2)

Put the location of a new unresolved forward reference orig2 onto the control flow stack. Append the run-time semantics given below to the current definition. The semantics will be incomplete until orig2 is resolved (e.g., by THEN). Resolve the forward reference orig1 using the location following the appended run-time semantics.

Run-time: (--)

Continue execution at the location given by the resolution of orig2.

See: 2.1.157 IF, 2.1.224 THEN.

2.1.136. EMIT CORE

(x --)

X is sent to UART0

See: 2.1.236 TYPE.

2.1.137. EMIT1 "Emit 1" BV-CORE

(x --)

X is sent to UART1

- 2.1.138. ENVIRONMENT? "environment-query" ** Not Implemented ** CORE**
 (c-addr u -- false | i*x true)
 c-addr is the address of a character string and u is the string's character count. u may have a value in the range from zero to an implementation-defined maximum which shall not be less than 31. The character string should contain a keyword from 3.2.6 Environmental queries or the optional word sets to be checked for correspondence with an attribute of the present environment. If the system treats the attribute as unknown, the returned flag is false; otherwise, the flag is true and the i*x returned is of the type specified in the table for the attribute queried.
[Not implemented in BV Forth](#)
- 2.1.139. ESCAPE "escape" BV-CORE**
 (--)
 Used to prematurely exit a word, when invoked the program will continue from the next word in the list. Use this in preference to EXIT id the program is going to be saved to Flash.
- 2.1.140. EVALUATE ** Not Implemented ** CORE**
 (i*x c-addr u -- j*x)
 Save the current input source specification. Store minus-one (-1) in SOURCE-ID if it is present. Make the string described by c-addr and u both the input source and input buffer, set >IN to zero, and interpret. When the parse area is empty, restore the prior input source specification. Other stack effects are due to the words EVALUATED.
[Not Implemented in BV Forth](#)
- 2.1.141. EXECUTE CORE**
 (i*x xt -- j*x)
 Remove xt from the stack and perform the semantics identified by it. Other stack effects are due to the word EXECUTEd.
- See: 2.1.16 ', 2.1.249 ['].**
- 2.1.142. EXIT CORE**
 Interpretation: Interpretation semantics for this word are undefined.
 Execution: (--) (R: nest-sys --)
 Return control to the calling definition specified by nest-sys. Before executing EXIT within a do-loop, a program shall discard the loop- control parameters by executing UNLOOP.
- See: 3.2.3.3 Return stack, 2.1.236 UNLOOP.**
- 2.1.143. EXITADDRESS "exit address" BV-CORE**
 (-- a-addr)
 Returns the code field address of the EXIT word as compiled in Flash. This is a constant that is used to discover the end of a Forth word when parsing through it.

During interpretation `EXIT` will give the same result but because of the way SAVE works, the above word combination cannot be saved to Flash.

2.1.144. FERASE "flash erase" BV-CORE

(start-sector end-sector – result)

Erases Flash from start sector to end sector, returns the result as specified in the LPC datasheet.

2.1.145. FILL CORE

(c-addr u char --)

If u is greater than zero, store char in each of u consecutive characters of memory beginning at c-addr.

2.1.146. FIND CORE

(c-addr -- c-addr 0 | xt 1 | xt -1)

Find the definition named in the counted string at c-addr. If the definition is not found, return c-addr and zero. If the definition is found, return its execution token xt. If the definition is immediate, also return one (1), otherwise also return minus-one (-1). For a given string, the values returned by FIND while compiling may differ from those returned while not compiling.

See: 3.4.2 Finding definition names, A.2.1.16 ', A.2.1.249 ['], .2.1.188 POSTPONE, D.6.7 Immediacy.

2.1.147. FM/MOD "f-m-slash-mod" CORE

(d1 n1 -- n2 n3)

Divide d1 by n1, giving the floored quotient n3 and the remainder n2. Input and output stack arguments are signed. An ambiguous condition exists if n1 is zero or if the quotient lies outside the range of a single-cell signed integer.

See: 3.2.2.1 Integer division, 2.1.215 SM/REM, 2.1.234 UM/MOD.

2.1.148. FOR "for" BV-CORE

(u --)

Start of a for next loop. This loop can iterate through positive integers and always starts from 0, it is equivalent to 0 DO.

2.1.149. FOSC "crystal frequency" BV-CORE

(-- u)

Returns the crystal frequency in KHz, this is not necessarily the same frequency as the CPU clock frequency.

2.1.150. FREERAM "free ram" BV-CORE

(-- u)

Returns the available free RAM in bytes. The free RAM is space that can be used for Forth words, this will be somewhat less than the total system RAM.

2.1.151. FWRITE "flash write" BV-CORE

(start-address n1 – flag)

Writes to Flash data beginning at the start address for n1 number of chunks, a chunk is a 512 byte block. Return -1 if success.

2.1.152. HERE CORE

(-- addr)

addr is the data-space pointer.

See: 3.3.3.2 Contiguous regions.

2.1.153. HEX "hex" BV-CORE

(--)

Stores 16 to BASE

2.1.154. HEXOUT "hex out" BV-CORE

(value width --)

Writes value to console in hex format to the given width. This word does not append the leading &.

2.1.155. HOLD CORE

(char --)

Add char to the beginning of the pictured numeric output string. An ambiguous condition exists if HOLD executes outside of a <# #> delimited number conversion.

2.1.156. I CORE

Interpretation: Interpretation semantics for this word are undefined.

Execution: (-- n|u) (R: loop-sys -- loop-sys) n|u is a copy of the current (innermost) loop index. An ambiguous condition exists if the loop control parameters are unavailable.

2.1.157. IF CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: -- orig)

Put the location of a new unresolved forward reference orig onto the control flow stack. Append the run-time semantics given below to the current definition. The semantics are incomplete until orig is resolved, e.g., by THEN or ELSE.

Run-time: (x --)

If all bits of x are zero, continue execution at the location specified by the resolution of orig.

See: 3.2.3.2 Control flow stack, 2.1.135 ELSE, 2.1.224 THEN.

2.1.158. IMMEDIATE CORE

(--)

Make the most recent definition an immediate word. An ambiguous condition exists if the most recent definition does not have a name.

See: D.6.7 Immediacy.

2.1.159. INTEGER "integer" BV-CORE

(-- ccc)

Creates a variable to hold a 32bit integer. When executed it returns the value of the integer rather than the address as VARIABLE would do.

To set and modify the value of an integer use =>, +> and %@

See 2.1.238

2.1.160. INVERT CORE

(x1 -- x2)

Invert all bits of x1, giving its logical inverse x2.

See: 2.1.181 NEGATE, 2.1.41 0=.

2.1.161. J CORE

Interpretation: Interpretation semantics for this word are undefined.

Execution: (-- n|u) (R: loop-sys1 loop-sys2 -- loop-sys1 loop-sys2)

n|u is a copy of the next-outer loop index. An ambiguous condition exists if the loop control parameters of the next-outer loop, loop-sys1, are unavailable.

2.1.162. KEY "key" CORE

(-- char)

Receive one character char from UART0. Execution will remain in KEY until a character is received.

See: 10.6.2.1307 EKEY, 10.6.1.1755 KEY?.

2.1.163. KEY? "key query" BV-CORE

(-- flag)

Flag will be -1 if there is a character available from UART0, KEY? will not get the character.

2.1.164. KEY1 “key1” BV-CORE
 (-- char)
 Receive one character char from UART1. Execution will remain in KEY1 until a character is received.

2.1.165. KEY1? "key one query" BV-CORE
 (-- flag)
 Flag will be -1 if there is a character available from UART1, KEY1? will not get the character.

2.1.166. L> “from leave” BV-CORE
 (-- n1)
 Retrieves a value from the leave sack, this must be balanced with a >L and must not be used across loop constructs.

2.1.167. LATEST “latest” BV-CORE
 (-- a-addr)
 This is the address of the variable that stores the code field address of the most recently created word. This is mostly used with fetch like so:
 LATEST @
 This will return the code field address of the most recently created word.

2.1.168. LEAVE CORE
 Interpretation: Interpretation semantics for this word are undefined.
 Execution: (--) (R: loop-sys --)
 Discard the current loop control parameters. An ambiguous condition exists if they are unavailable. Continue execution immediately following the innermost syntactically enclosing DO ... LOOP or DO ... +LOOP.

See: 3.2.3.3 Return stack, 2.1.32 +LOOP, 2.1.171 LOOP.

2.1.169. LENGTH “length” BV-CORE
 (s-addr -- n)
 Returns the length of a 0 terminated string at the address on the stack.

2.1.170. LITERAL CORE
 Interpretation: Interpretation semantics for this word are undefined.
 Compilation: (x --)
 Append the run-time semantics given below to the current definition.
 Run-time: (-- x)
 Place x on the stack.

2.1.171. LOADF "load forth" BV-CORE

(--)

Start of the mechanism for loading Forth files using a simple acknowledge protocol. When this word is used it will accept and compile words a line at a time. When it has finished compiling a line it will send an ACK that the terminal can wait for and then send another line. The loading is turned off by ;S.

2.1.172. LOOP CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: do-sys --)

Append the run-time semantics given below to the current definition. Resolve the destination of all unresolved occurrences of LEAVE between the location given by do-sys and the next location for a transfer of control, to execute the words following the LOOP.

Run-time: (--) (R: loop-sys1 -- | loop-sys2)

An ambiguous condition exists if the loop control parameters are unavailable. Add one to the loop index. If the loop index is then equal to the loop limit, discard the loop parameters and continue execution immediately following the loop. Otherwise continue execution at the beginning of the loop.

See: 2.1.127 DO, 2.1.156 I, 2.1.163 LEAVE.

2.1.173. LSHIFT "l-shift" CORE

(x1 u -- x2)

Perform a logical left shift of u bit-places on x1, giving x2. Put zeroes into the least significant bits vacated by the shift. An ambiguous condition exists if u is greater than or equal to the number of bits in a cell.

2.1.174. LSHIFTC "left shift with carry" BV-CORE

(value n – result flag)

The value is left shifted by the amount given by n and gives the result. After the shift if carry is set then flag is set to -1.

2.1.175. M* "m-star" CORE

(n1 n2 -- d)

d is the signed product of n1 times n2.

2.1.176. MAX CORE

(n1 n2 -- n3)

n3 is the greater of n1 and n2.

2.1.177. MIN CORE

(n1 n2 -- n3)

n3 is the lesser of n1 and n2.

2.1.178. MEM-RANGE “memory range” BV-CORE

(addr -- flag)

Tests the address given for a valid memory range for this chip, this is used to prevent access violations. If the address is within range the flag will be -1

2.1.179. MOD CORE

(n1 n2 -- n3)

Divide n1 by n2, giving the single-cell remainder n3. An ambiguous condition exists if n2 is zero. If n1 and n2 differ in sign, the implementation-defined result returned will be the same as that returned by either the phrase >R S>D R> FM/MOD DROP or the phrase >R S>D R> SM/REM DROP.

See: 3.2.2.1 Integer division.

2.1.180. MOVE CORE

(addr1 addr2 u --)

If u is greater than zero, copy the contents of u consecutive address units at addr1 to the u consecutive address units at addr2. After MOVE completes, the u consecutive address units at addr2 contain exactly what the u consecutive address units at addr1 contained before the move.

See: 17.6.1.0910 CMOVE, 17.6.1.0920 CMOVE>.

2.1.181. MS “millisecond” BV-CORE

(u --)

Delay for the number of milliseconds given by u. This uses timer 1 and is locked in a continuous loop.

2.1.182. NEGATE CORE

(n1 -- n2)

Negate n1, giving its arithmetic inverse n2.

See: 2.1.159 INVERT, 2.1.41 0=.

2.1.183. NEW “new” BV-CORE

(--)

This will erase the Flash memory from the end of the core system to the end of Flash. It is recommended that the Flash be filled as much as possible before using this as the Flash has a limited number of erase cycles, although this is very large.

- 2.1.184. NEWSTART** "new flash start" **BV-CORE**
 (-- address)
 Gives the address of the first clear part of Flash.
- 2.1.185. NEXT** "next" **BV-CORE**
 (u --)
 End of a for next loop. NEXT terminates the FOR loop.
- 2.1.186. OR** **CORE**
 (x1 x2 -- x3)
 x3 is the bit-by-bit inclusive-or of x1 with x2.
- 2.1.187. OVER** **CORE**
 (x1 x2 -- x1 x2 x1)
 Place a copy of x1 on top of the stack.
- 2.1.188. PAD** "pad" **BV-CORE**
 (-- addr)
 PAD returns the address of the start of a general purpose buffer (80 characters big). This is used for various core Forth words so it is advisable to only use it within a single word.
- 2.1.189. PB.** "print binary" **BV-CORE**
 (-- u)
 Prints the value stored on the stack as binary regardless of the current BASE setting
- 2.1.190. PD.** "print decimal" **BV-CORE**
 (-- u)
 Prints the value stored on the stack as decimal regardless of the current BASE setting
- 2.1.191. PH.** "print hex" **BV-CORE**
 (-- u)
 Prints the value stored on the stack as hex regardless of the current BASE setting

2.1.192. PICK "pick" BV-CORE

(x..xn x2 - x3)

X2 is the pick number and x..xn are values on the stack, this will pick a stack item and place it on top of the stack thus:

(1 2 3 4 **0**) PICK >> (1 2 3 **4 4**) Copies first stack item
 (1 2 3 4 **2**) PICK >> (1 **2** 3 4 **2**) Copies third stack item

2.1.193. POSTPONE CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: ("<spaces>name" --)

Skip leading space delimiters. Parse name delimited by a space. Find name. Append the compilation semantics of name to the current definition. An ambiguous condition exists if name is not found.

See: 3.4.1 Parsing.

2.1.194. PSIZE "program size" BV-CORE

(-- u)

Returns the current size of the words in RAM in bytes.

2.1.195. QUIT CORE

(--) (R: i*x --)

Empty the return stack, store zero in SOURCE-ID if it is present, make the user input device the input source, and enter interpretation state. Do not display a message. Repeat the following:

Accept a line from the input source into the input buffer, set >IN to zero, and interpret.

Display the implementation-defined system prompt if in interpretation state, all processing has been completed, and no ambiguous condition exists.

See: 3.4 The Forth text interpreter.

2.1.196. R0 "r zero" BV-CORE

(-- addr)

Returns the start of the return stack.

2.1.197. RAMSIZE "ramsize" BV-CORE

(-- u)

Returns the RAM available for Forth words, in bytes.

2.1.198. RAMTOP "ramtop" BV-CORE

(-- addr)

Returns the address of the top of the RAM that is available to Forth words, this is the physical ram top minus any system stacks and buffers. This will reduce as variables are added to the system.

2.1.199. RND "rand" BV-CORE

(-- n)

Returns a 32bit random number.

See SEED

2.1.200. RSHASH "rs-hash" BV-CORE

(string-address -- u)

Given the address of a valid zero terminated string this will return the hash of that string. The hash is used for uniquely identifying all BV Forth words.

2.1.201. RSHIFTC "right shift carry" BV-CORE

(value n – result flag)

The value is right shifted by the amount given by n and gives the result. After the shift if carry is set then flag is set to -1.

2.1.202. R> "r-from" CORE

Interpretation: Interpretation semantics for this word are undefined.

Execution: (-- x) (R: x --)

Move x from the return stack to the data stack.

See: 3.2.3.3 Return stack, 2.1.84 >R, 2.1.203 R@, 6.2.0340 2>R, 6.2.0410 2R>, 6.2.0415 2R@.

2.1.203. R@ "r-fetch" CORE

Interpretation: Interpretation semantics for this word are undefined.

Execution: (-- x) (R: x -- x)

Copy x from the return stack to the data stack.

See: 3.2.3.3 Return stack, 2.1.84 >R, 2.1.202 R>, 6.2.0340 2>R, 6.2.0410 2R>, 6.2.0415 2R@.

2.1.204. RECURSE ** Not

Implemented ** CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (--) Append the execution semantics of the current definition to the current definition. An ambiguous condition exists if RECURSE appears in a definition after DOES>.

[Not implemented in current version](#)

See: 2.1.129 DOES>, 2.1.204 RECURSE.

2.1.205. REPEAT CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: orig dest --)

Append the run-time semantics given below to the current definition, resolving the backward reference dest. Resolve the forward reference orig using the location following the appended run-time semantics.

Run-time: (--)

Continue execution at the location given by dest.

See: 2.1.98 BEGIN, 2.1.239 WHILE.

2.1.206. ROT "rote" CORE

(x1 x2 x3 -- x2 x3 x1)

Rotate the top three stack entries.

2.1.207. RSHIFT "r-shift" CORE

(x1 u -- x2)

Perform a logical right shift of u bit-places on x1, giving x2. Put zeroes into the most significant bits vacated by the shift. An ambiguous condition exists if u is greater than or equal to the number of bits in a cell.

2.1.208. S" "s-quote" CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: ("ccc<quote>" --)

Parse ccc delimited by " (double-quote). Append the run-time semantics given below to the current definition.

Run-time: (-- c-addr u)

Return c-addr and u describing a string consisting of the characters ccc. A program shall not alter the returned string.

See: 3.4.1 Parsing, 6.2.0855 C", 11.2.1.208 S".

2.1.209. S>D "s-to-d" CORE

(n -- d)

Convert the number n to the double-cell number d with the same numerical value.

2.1.210. S= "string compare" BV-CORE

(str1 str2 -- flag)

Compares two 0 terminated strings str1 and str2, if the characters of string 2 match those of string 1 the flag will be true. Str1 is the search string thus:

Str1	str2	flag
'abc'	'abcdef'	-1
'abcdef'	'abc'	0

2.1.217. SOURCE CORE

(-- c-addr u)

c-addr is the address of, and u is the number of characters in, the input buffer.

2.1.218. SP@ "stack pointer fetch" BV-CORE

(-- addr)

Returns the address of the data stack pointer.

2.1.219. SPACE CORE

(--)

Display one space.

2.1.220. SPACES CORE

(n --)

If n is greater than zero, display n spaces.

2.1.221. STATE CORE

(-- a-addr)

a-addr is the address of a cell containing the compilation-state flag. STATE is true when in compilation state, false otherwise. The true value in STATE is non-zero, but is otherwise implementation-defined. Only the following standard words alter the value in STATE: : (colon), ; (semicolon), ABORT, QUIT, :NONAME, [(left-bracket), and] (right- bracket).

Note:

A program shall not directly alter the contents of STATE.

See: 3.4 The Forth text interpreter, 2.1.52 :, 2.1.53 ;, 2.1.88 ABORT, 2.1.194 QUIT, 2.1.248 [, 2.1.251], 6.2.0455 :NONAME, 15.6.2.2250 STATE.

2.1.222. STYPE "stype" BV-CORE

(s-addr1 --)

Displays the 0 terminated string given at s-addr1 to the console, this is the Forth equivalent to TYPE using a counted string.

2.1.223. SWAP CORE

(x1 x2 -- x2 x1)

Exchange the top two stack items.

2.1.230. U. "u-dot" CORE

(u --)

Display u in free field format.

2.1.231. U< "u-less-than" CORE

(u1 u2 -- flag)

flag is true if and only if u1 is less than u2.

See: 2.1.54 <.

2.1.232. UART "uart" BV-CORE

(0|1 -)

Redirects all output and input to the specified UART, default is UART 0.

2.1.233. UM* "u-m-star" CORE

(u1 u2 -- ud)

Multiply u1 by u2, giving the unsigned double-cell product ud. All values and arithmetic are unsigned.

2.1.234. U/MOD "u slash mod" BV-CORE

(u1 u2 - uR uQ)

A 32bit unsigned divide giving remainder and quotient.

2.1.235. UM/MOD "u-m-slash-mod" CORE

(ud u1 -- u2 u3)

Divide ud by u1, giving the quotient u3 and the remainder u2. All values and arithmetic are unsigned. An ambiguous condition exists if u1 is zero or if the quotient lies outside the range of a single-cell unsigned integer.

See: 3.2.2.1 Integer division, 2.1.147 FM/MOD, 2.1.215 SM/REM.

2.1.236. UNLOOP CORE

Interpretation: Interpretation semantics for this word are undefined.

Execution: (--) (R: loop-sys --)

Discard the loop-control parameters for the current nesting level. An UNLOOP is required for each nesting level before the definition may be EXITed. An ambiguous condition exists if the loop-control parameters are unavailable.

See: 3.2.3.3 Return stack.

2.1.237. UNTIL CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: dest --)

Append the run-time semantics given below to the current definition, resolving the backward reference dest.

Run-time: (x --)

If all bits of x are zero, continue execution at the location specified by dest.

See: 2.1.98 BEGIN.

2.1.238. VARIABLE CORE

("<spaces>name" --)

Skip leading space delimiters. Parse name delimited by a space. Create a definition for name with the execution semantics defined below. Reserve one cell of data space at an aligned address. name is referred to as a "variable" name

Execution: (-- a-addr)

a-addr is the address of the reserved cell. A program is responsible for initializing the contents of the reserved cell.

See: 3.4.1 Parsing.

2.1.239. VP "variable pointer" BV-CORE

(- addr)

Returns the address of the variable pointer, use @ to get the contents.

2.1.240. VSPACE\$ "vspace" BV-CORE

(u1 - ccc)

Reserves u1 bytes of space (rounded to the nearest whole 4 bytes) in the variable space and assigns the name given in ccc to point to the start of that space.

Example: 4 VSPACE\$ A10

The above will allocate 4 bytes of variable space to the new variable A10.

2.1.241. WHASH "w hash" BV-CORE

(lfa - t-hash)

Given the link field address (CFA - 16), this will return the hash of that word.

Example: ` M* 16 - WHASH

The above example obtains the CFA of word M*, converts it to the LFA by subtracting 16 and WHASH will reveal the words hash value in the upper half of a 32bit word.

2.1.242. WHILE CORE

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: dest -- orig dest)

Put the location of a new unresolved forward reference orig onto the control flow stack, under the existing dest. Append the run-time semantics given below to the

current definition. The semantics are incomplete until orig and dest are resolved (e.g., by REPEAT).

Run-time: (x --)

If all bits of x are zero, continue execution at the location specified by the resolution of orig.

2.1.243. WORD CORE

(char "<chars>ccc<char>" -- c-addr)

Skip leading delimiters. Parse characters ccc delimited by char. An ambiguous condition exists if the length of the parsed string is greater than the implementation-defined length of a counted string.

c-addr is the address of a transient region containing the parsed word as a counted string. If the parse area was empty or contained no characters other than the delimiter, the resulting string has a zero length. A space, not included in the length, follows the string. A program may replace characters within the string.

Note: The requirement to follow the string with a space is obsolescent and is included as a concession to existing programs that use CONVERT. A program shall not depend on the existence of the space.

See: 3.3.3.6 Other transient regions, 3.4.1 Parsing.

2.1.244. WRITE512 "write 512" BV-CORE

(addr1 - flag)

Writes 512 bytes from RAM, start of RAM specified by addr1, to the next free Flash location. The next free Flash location is given by NEWFSTART this will be updated after the write.

2.1.245. WSIZE "wsize" BV-CORE

(u1 - u2)

Give a number u1, u2 is rounded up to the nearest 512.

2.1.246. Xy "x to the power of y" BV-CORE**

(u1 u2 - u3)

Raises u1 to the power of u2, this will give unpredictable results if signed numbers are used.

2.1.247. XOR "x-or" CORE

(x1 x2 -- x3)

x3 is the bit-by-bit exclusive-or of x1 with x2.

4. Appendix B Resources

Here are some useful resources:

www.byvac.co.uk Sells the BV511 ARM board and IASI devices

www.pin1.org Support for BV Forth and projects

www.forth.org The source for Forth code and documentation

<http://pygmy.utoh.org/forth.html> Frank Sergeant's Forth page

<http://www.taygeta.com/forthlit.html> Comprehensive Forth resources including tutorials and books.