 Table of Contents

6. Glossary

See: A.6 Glossary

6.1 Core words

6.1.0010 **!**
store CORE

```
( x a-addr -- )
```

Store x at a-addr.

See: 3.3.3.1 Address alignment

6.1.0030 **#**
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: 6.1.0050 #S

6.1.0040 **#>**
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: 6.1.0030 # , 6.1.0050 #S , 6.1.0490 <#

6.1.0050 **#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.

6.1.0070 **'**
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.6.1.2033 POSTPONE , 6.1.2510 ['] , A.6.1.0070 ' , D.6.7 Immediacy.

6.1.0080 **(**
paren 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.

See: 3.4.1 Parsing, 11.6.1.0080 (, A.6.1.0080 (

6.1.0090 ***
star CORE

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

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

6.1.0100 ***/**
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

6.1.0110 ***/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

6.1.0120 **+**
plus CORE

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

Add n2|u2 to n1|u1, giving the sum n3|u3.

See: 3.3.3.1 Address alignment

6.1.0130 **+!**
plus-store CORE

(n|u a-addr --)

Add n|u to the single-cell number at a-addr.

See: 3.3.3.1 Address alignment

6.1.0140 **+LOOP**
plus-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 +LOOP.

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

An ambiguous condition exists if the loop control parameters are unavailable. Add n to the loop index. If the loop index did not cross the boundary between the loop limit minus one and the loop limit, continue execution at the beginning of the loop. Otherwise, discard the current loop control parameters and continue execution immediately following the loop.

See: 6.1.1240 DO , 6.1.1680 I , 6.1.1760 LEAVE , A.6.1.0140 +LOOP

6.1.0150 ,
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, A.6.1.0150 ,

---

6.1.0160 -
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.

6.1.0180 .
dot CORE

(n --)

Display n in free field format.

See: 3.2.1.2 Digit conversion, 3.2.1.3 Free-field number display.

6.1.0190 ."
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 .(, A.6.1.0190 ."

6.1.0230 /
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

6.1.0240 **/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

6.1.0250 **0<**
zero-less CORE

```
( n -- flag )
```

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

6.1.0270 **0=**
zero-equals CORE

```
( x -- flag )
```

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

6.1.0290 **1+**
one-plus CORE

```
( n1|u1 -- n2|u2 )
```

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

6.1.0300 **1-**
one-minus CORE

```
( n1|u1 -- n2|u2 )
```

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

6.1.0310 **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

6.1.0320 **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.

See: A.6.1.0320 2*

6.1.0330 **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.

See: A.6.1.0330 2/

6.1.0350 **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, 6.1.0310 2! , A.6.1.0350 2@

6.1.0370 **2DROP**
two-drop CORE

```
( x1 x2 -- )
```

Drop cell pair x1 x2 from the stack.

6.1.0380 **2DUP**
two-dupe CORE

```
( x1 x2 -- x1 x2 x1 x2 )
```

Duplicate cell pair x1 x2.

6.1.0400 **2OVER**
two-over CORE

```
( x1 x2 x3 x4 -- x1 x2 x3 x4 x1 x2 )
```

Copy cell pair x1 x2 to the top of the stack.

6.1.0430 **2SWAP**
two-swap CORE

```
( x1 x2 x3 x4 -- x3 x4 x1 x2 )
```

Exchange the top two cell pairs.

6.1.0450 **:**
colon CORE

```
( C: "<spaces>name" -- colon-sys )
```

Skip leading space delimiters. Parse name delimited by a space. Create a definition for name, called a **colon definition**. Enter compilation state and start the current definition, producing colon-sys. Append the initiation semantics given below to the current definition.

The execution semantics of name will be determined by the words compiled into the body of the definition. The current definition shall not be findable in the dictionary until it is ended (or until the execution of DOES> in some systems).

```
Initiation: ( i*x -- i*x )  ( R:  -- nest-sys )
```

Save implementation-dependent information nest-sys about the calling definition. The stack effects i*x represent arguments to name.

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

Execute the definition name. The stack effects i*x and j*x represent arguments to and results from name, respectively.

See: 3.4 The Forth text interpreter, 3.4.1 Parsing, 3.4.5 Compilation, 6.1.2500 [, 6.1.2540] , 15.6.2.0470 ;CODE , A.6.1.0450 : , RFI 0005 Initiation semantics.

6.1.0460 **;**
semicolon CORE

```
Interpretation: Interpretation semantics for this word are undefined.

Compilation: ( C: colon-sys -- )
```

Append the run-time semantics below to the current definition. End the current definition, allow it to be found in the dictionary and enter interpretation state, consuming colon-sys. If the data-space pointer is not aligned, reserve enough data space to align it.

```
Run-time: ( -- ) ( R: nest-sys -- )
```

Return to the calling definition specified by nest-sys.

See: 3.4 The Forth text interpreter, 3.4.5 Compilation, A.6.1.0460 ;

6.1.0480 **<**
less-than CORE

```
( n1 n2 -- flag )
```

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

See: 6.1.2340 U<

6.1.0490 **<#**
less-number-sign CORE

```
( -- )
```

Initialize the pictured numeric output conversion process.

See: 6.1.0030 # , 6.1.0040 #> , 6.1.0050 #S

6.1.0530 **=**
equals CORE

```
( x1 x2 -- flag )
```

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

6.1.0540 **>**
greater-than CORE

```
( n1 n2 -- flag )
```

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

See: 6.2.2350 U>

6.1.0550 **>BODY**
to-body 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, A.6.1.0550 >BODY

6.1.0560 **>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.

6.1.0570 **>NUMBER**
to-number 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.

See: 3.2.1.2 Digit conversion

6.1.0580 **>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, 6.1.2060 R> , 6.1.2070 R@ , 6.2.0340 2>R , 6.2.0410 2R> , 6.2.0415 2R@

6.1.0630 **?DUP**
question-dupe CORE

```
( x -- 0 | x x )
```

Duplicate x if it is non-zero.

6.1.0650 **@**
fetch CORE

```
( a-addr -- x )
```

x is the value stored at a-addr.

See: 3.3.3.1 Address alignment

6.1.0670 **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

6.1.0680 **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" , A.6.1.0680 ABORT"

6.1.0690 **ABS**
abs CORE

```
( n -- u )
```

u is the absolute value of n.

6.1.0695 **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.

See: A.6.1.0695 ACCEPT

6.1.0705 **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, A.6.1.0705 ALIGN

6.1.0706 **ALIGNED**
CORE

```
( addr -- a-addr )
```

a-addr is the first aligned address greater than or equal to addr.

See: 3.3.3.1 Address alignment, 6.1.0705 ALIGN

6.1.0710 **ALLOT**
CORE

```
( n -- )
```

If n is greater than zero, reserve n address units of data space. If n is less than zero, release |n| address units of data space. If n is zero, leave the data-space pointer unchanged.

If the data-space pointer is aligned and n is a multiple of the size of a cell when ALLOT begins execution, it will remain aligned when ALLOT finishes execution.

If the data-space pointer is character aligned and n is a multiple of the size of a character when ALLOT begins execution, it will remain character aligned when ALLOT finishes execution.

See: 3.3.3 Data space

6.1.0720 **AND**
CORE

```
( x1 x2 -- x3 )
```

x3 is the bit-by-bit logical **and** of x1 with x2.

6.1.0750 **BASE**
CORE

```
( -- a-addr )
```

a-addr is the address of a cell containing the current number-conversion radix {{2...36}}.

6.1.0760 **BEGIN**
CORE

```
Interpretation: Interpretation semantics for this word are undefined.

Compilation: ( C: -- dest )
```

Put the next location for a transfer of control, dest, onto the control flow stack. Append the run-time semantics given below to the current definition.

```
Run-time: ( -- )
```

Continue execution.

See: 3.2.3.2 Control-flow stack, 6.1.2140 REPEAT , 6.1.2390 UNTIL , 6.1.2430 WHILE , A.6.1.0760 BEGIN

6.1.0770 **BL**
b-l CORE

```
( -- char )
```

char is the character value for a space.

See: A.6.1.0770 BL

6.1.0850 **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

6.1.0860 **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.

6.1.0870 **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

6.1.0880 **CELL+**
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, A.6.1.0880 CELL+

6.1.0890 **CELLS**
CORE

```
( n1 -- n2 )
```

n2 is the size in address units of n1 cells.

See: A.6.1.0890 CELLS

6.1.0895 **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, 6.1.2520 [CHAR] , A.6.1.0895 CHAR

6.1.0897 **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

6.1.0898 **CHARS**
chars CORE

```
( n1 -- n2 )
```

n2 is the size in address units of n1 characters.

6.1.0950 **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, A.6.1.0950 CONSTANT

6.1.0980 **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.

6.1.0990 **CR**

c-r CORE

(--)

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

6.1.1000 **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, A.6.1.1000 CREATE

6.1.1170 **DECIMAL**
CORE

(--)

Set the numeric conversion radix to ten (decimal).

6.1.1200 **DEPTH**
CORE

(-- +n)

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

6.1.1240 **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, 6.1.0140 +LOOP , A.6.1.1240 DO

6.1.1250 **DOES>**
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: A.6.1.1250 DOES> , RFI 0003 Defining words etc., RFI 0005 Initiation semantics.

6.1.1260 **DROP**
CORE

```
( x -- )
```

Remove x from the stack.

6.1.1290 **DUP**
dupe CORE

```
( x -- x x )
```

Duplicate x.

6.1.1310 **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: 6.1.1700 IF , A.6.1.1310 ELSE

6.1.1320 **EMIT**
CORE

```
( x -- )
```

If x is a graphic character in the implementation-defined character set, display x. The effect of EMIT for all other values of x is implementation-defined.

When passed a character whose character-defining bits have a value between hex 20 and 7E inclusive, the corresponding standard character, specified by 3.1.2.1 Graphic characters, is displayed. Because different output devices can respond differently to control characters, programs that use control characters to perform specific functions have an environmental dependency. Each EMIT deals with only one character.

See: 6.1.2310 TYPE

6.1.1345 **ENVIRONMENT?**
environment-query 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.

See: A.6.1.1345 ENVIRONMENT?

6.1.1360 **EVALUATE**
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.

See: 7.6.1.1360 EVALUATE , A.6.1.1360 EVALUATE , RFI 0006 Writing to Input Buffers.

6.1.1370 **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: 6.1.0070 ' , 6.1.2510 [']

6.1.1380 **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, A.6.1.1380 EXIT

6.1.1540 **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.

6.1.1550 **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, 6.1.0070 ' , 6.1.2510 ['] , A.6.1.1550 FIND , A.6.1.2033 POSTPONE , D.6.7 Immediacy.

6.1.1561 **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, 6.1.2214 SM/REM , 6.1.2370 UM/MOD , A.6.1.1561 FM/MOD

6.1.1650 **HERE**
CORE

```
( -- addr )
```

addr is the data-space pointer.

See: 3.3.3.2 Contiguous regions

6.1.1670 **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.

6.1.1680 **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.

6.1.1700 **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, A.6.1.1700 IF

6.1.1710 **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: A.6.1.1710 IMMEDIATE , D.6.7 Immediacy, RFI 0007 Distinction between *immediacy* and *special compilation semantics*.

6.1.1720 **INVERT**
CORE

```
( x1 -- x2 )
```

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

See: 6.1.1910 NEGATE , 6.1.0270 0= , A.6.1.1720 INVERT

6.1.1730 **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.

See: A.6.1.1730 J

6.1.1750 **KEY**
CORE

(-- char)

Receive one character char, a member of the implementation-defined character set. Keyboard events that do not correspond to such characters are discarded until a valid character is received, and those events are subsequently unavailable.

All standard characters can be received. Characters received by KEY are not displayed.

Any standard character returned by KEY has the numeric value specified in 3.1.2.1 Graphic characters. Programs that require the ability to receive control characters have an environmental dependency.

See: 10.6.2.1305 EKEY , 10.6.1.1755 KEY?

6.1.1760 **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, A.6.1.1760 LEAVE

6.1.1780 **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.

See: A.6.1.1780 LITERAL

6.1.1800 **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: 6.1.1240 DO , 6.1.1680 I , A.6.1.1800 LOOP

6.1.1805 **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.

6.1.1810 **M***
m-star CORE

```
( n1 n2 -- d )
```

d is the signed product of n1 times n2.

See: A.6.1.1810 M*

6.1.1870 **MAX**
CORE

```
( n1 n2 -- n3 )
```

n3 is the greater of n1 and n2.

6.1.1880 **MIN**
CORE

```
( n1 n2 -- n3 )
```

n3 is the lesser of n1 and n2.

6.1.1890 **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

6.1.1900 **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> , A.6.1.1900 MOVE

6.1.1910 **NEGATE**
CORE

```
( n1 -- n2 )
```

Negate n1, giving its arithmetic inverse n2.

See: 6.1.1720 INVERT , 6.1.0270 0=

6.1.1980 **OR**
CORE

(x1 x2 -- x3)

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

6.1.1990 **OVER**
CORE

(x1 x2 -- x1 x2 x1)

Place a copy of x1 on top of the stack.

6.1.2033 **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: D.6.7 Immediacy, 3.4.1 Parsing, A.6.1.2033 POSTPONE , 6.2.2530 [COMPILE]

6.1.2050 **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

6.1.2060 **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, 6.1.0580 >R , 6.1.2070 R@ , 6.2.0340 2>R , 6.2.0410 2R> , 6.2.0415 2R@

6.1.2070 **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, 6.1.0580 >R , 6.1.2060 R> , 6.2.0340 2>R , 6.2.0410 2R> , 6.2.0415 2R@

6.1.2120 **RECURSE**
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>.

See: 6.1.2120 RECURSE , A.6.1.2120 RECURSE

6.1.2140 **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: 6.1.0760 BEGIN , 6.1.2430 WHILE , A.6.1.2140 REPEAT

6.1.2160 **ROT**
rote CORE

(x1 x2 x3 -- x2 x3 x1)

Rotate the top three stack entries.

6.1.2162 **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.

6.1.2165 **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" , 11.6.1.2165 S" , A.6.1.2165 S"

6.1.2170 **S>D**
s-to-d CORE

(n -- d)

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

6.1.2210 **SIGN**
CORE

(n --)

If n is negative, add a minus sign to the beginning of the pictured numeric output string. An ambiguous condition exists if SIGN executes outside of a <# #> delimited number conversion.

6.1.2214 **SM/REM**
s-m-slash-rem CORE

```
( d1 n1 -- n2 n3 )
```

Divide d1 by n1, giving the symmetric 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, 6.1.1561 FM/MOD , 6.1.2370 UM/MOD , A.6.1.2214 SM/REM

6.1.2216 **SOURCE**
CORE

```
( -- c-addr u )
```

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

See: A.6.1.2216 SOURCE , RFI 0006 Writing to Input Buffers.

6.1.2220 **SPACE**
CORE

```
( -- )
```

Display one space.

6.1.2230 **SPACES**
CORE

```
( n -- )
```

If n is greater than zero, display n spaces.

6.1.2250 **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, 15.6.2.2250 STATE , A.6.1.2250 STATE , RFI 0007 Distinction between *immediacy* and *special compilation semantics*.

6.1.2260 **SWAP**
CORE

```
( x1 x2 -- x2 x1 )
```

Exchange the top two stack items.

6.1.2270 **THEN**
CORE

```
Interpretation: Interpretation semantics for this word are undefined.

Compilation: ( C: orig -- )
```

Append the run-time semantics given below to the current definition. Resolve the forward reference orig using the location of the appended run-time semantics.

```
Run-time: ( -- )
```

Continue execution.

See: 6.1.1310 ELSE , 6.1.1700 IF , A.6.1.2270 THEN

6.1.2310 **TYPE**
CORE

```
( c-addr u -- )
```

If u is greater than zero, display the character string specified by c-addr and u.

When passed a character in a character string whose character-defining bits have a value between hex 20 and 7E inclusive, the corresponding standard character, specified by 3.1.2.1 graphic characters, is displayed. Because different output devices can respond differently to control characters, programs that use control characters to perform specific functions have an environmental dependency.

See: 6.1.1320 EMIT

6.1.2320 **U.**
u-dot CORE

```
( u -- )
```

Display u in free field format.

6.1.2340 **U<**
u-less-than CORE

```
( u1 u2 -- flag )
```

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

See: 6.1.0480 <

6.1.2360 **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.

6.1.2370 **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, 6.1.1561 FM/MOD , 6.1.2214 SM/REM

6.1.2380 **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, A.6.1.2380 UNLOOP

6.1.2390 **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: 6.1.0760 BEGIN , A.6.1.2390 UNTIL

6.1.2410 **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, A.6.1.2410 VARIABLE

6.1.2430 **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.

See: A.6.1.2430 WHILE

6.1.2450 **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, 6.2.2008 PARSE , A.6.1.2450 WORD

6.1.2490 **XOR**
x-or CORE

(x1 x2 -- x3)

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

6.1.2500 **[**
left-bracket CORE

```
Interpretation: Interpretation semantics for this word are undefined.

Compilation: Perform the execution semantics given below.

Execution: ( -- )
```

Enter interpretation state. [is an immediate word.

See: 3.4 The Forth text interpreter, 3.4.5 Compilation, 6.1.2540] , A.6.1.2500 [

6.1.2510 **[']**
bracket-tick 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 run-time semantics given below to the current definition.

An ambiguous condition exists if name is not found.

```
Run-time: ( -- xt )
```

Place name's execution token xt on the stack. The execution token returned by the compiled phrase **[']** **X** is the same value returned by **'** **X** outside of compilation state.

See: 3.4.1 Parsing, 6.1.0070 ' , A.6.1.2033 POSTPONE , A.6.1.2510 ['] , D.6.7 Immediacy.

6.1.2520 **[CHAR]**
bracket-char CORE

```
Interpretation: Interpretation semantics for this word are undefined.

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

Skip leading space delimiters. Parse name delimited by a space. Append the run-time semantics given below to the current definition.

```
Run-time: ( -- char )
```

Place char, the value of the first character of name, on the stack.

See: 3.4.1 Parsing, 6.1.0895 CHAR , A.6.1.2520 [CHAR]

6.1.2540 **]**
right-bracket CORE

```
( -- )
```

Enter compilation state.

See: 3.4 The Forth text interpreter, 3.4.5 Compilation, 6.1.2500 [, A.6.1.2540] ,

---

6.2 Core extension words

See: A.6.2 Core extension words

6.2.0060 **#TIB**
number-t-i-b CORE EXT

```
( -- a-addr )
```

a-addr is the address of a cell containing the number of characters in the terminal input buffer.

Note: This word is obsolescent and is included as a concession to existing implementations.

See: A.6.2.0060 #TIB

6.2.0200 **.(**
dot-paren CORE EXT

```
Compilation: Perform the execution semantics given below.

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

Parse and display ccc delimited by) (right parenthesis). .(is an immediate word.

See: 3.4.1 Parsing, 6.1.0190 ." , A.6.2.0200 .(

6.2.0210 **.R**
dot-r CORE EXT

```
( n1 n2 -- )
```

Display n1 right aligned in a field n2 characters wide. If the number of characters required to display n1 is greater than n2, all digits are displayed with no leading spaces in a field as wide as necessary.

See: A.6.2.0210 .R

6.2.0260 **0<>**
zero-not-equals CORE EXT

```
( x -- flag )
```

flag is true if and only if x is not equal to zero.

6.2.0280 **0>**
zero-greater CORE EXT

```
( n -- flag )
```

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

6.2.0340 **2>R**
two-to-r CORE EXT

```
Interpretation: Interpretation semantics for this word are undefined.

Execution: ( x1 x2 -- ) ( R:  -- x1 x2 )
```

Transfer cell pair x1 x2 to the return stack. Semantically equivalent to **SWAP >R >R** .

See: 3.2.3.3 Return stack, 6.1.0580 >R , 6.1.2060 R> , 6.1.2070 R@ , 6.2.0410 2R> , 6.2.0415 2R@ , A.6.2.0340 2>R

6.2.0410 **2R>**
two-r-from CORE EXT

```
Interpretation: Interpretation semantics for this word are undefined.

Execution: ( -- x1 x2 ) ( R:  x1 x2 -- )
```

Transfer cell pair x1 x2 from the return stack. Semantically equivalent to **R> R> SWAP** .

See: 3.2.3.3 Return stack, 6.1.0580 >R , 6.1.2060 R> , 6.1.2070 R@ , 6.2.0340 2>R , 6.2.0415 2R@ , A.6.2.0410 2R>

6.2.0415 **2R@**
two-r-fetch CORE EXT

Interpretation: Interpretation semantics for this word are undefined.

Execution: (-- x1 x2) (R: x1 x2 -- x1 x2)

Copy cell pair x1 x2 from the return stack. Semantically equivalent to **R> R> 2DUP >R >R SWAP** .

See: 3.2.3.3 Return stack, 6.1.0580 >R , 6.1.2060 R> , 6.1.2070 R@ , 6.2.0340 2>R , 6.2.0410 2R>

6.2.0455 **:NONAME**
colon-no-name CORE EXT

(C: -- colon-sys) (S: -- xt)

Create an execution token xt, enter compilation state and start the current definition, producing colon-sys. Append the initiation semantics given below to the current definition.

The execution semantics of xt will be determined by the words compiled into the body of the definition. This definition can be executed later by using xt EXECUTE.

If the control-flow stack is implemented using the data stack, colon-sys shall be the topmost item on the data stack.

Initiation: (i*x -- i*x) (R: -- nest-sys)

Save implementation-dependent information nest-sys about the calling definition. The stack effects i*x represent arguments to xt.

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

Execute the definition specified by xt. The stack effects i*x and j*x represent arguments to and results from xt, respectively.

See: A.6.2.0455 :NONAME , 3.2.3.2 Control-flow stack.

6.2.0500 **<>**
not-equals CORE EXT

(x1 x2 -- flag)

flag is true if and only if x1 is not bit-for-bit the same as x2.

6.2.0620 **?DO**
question-do CORE EXT

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: -- do-sys)

Put 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)

If n1|u1 is equal to n2|u2, continue execution at the location given by the consumer of do-sys. Otherwise set up loop control parameters with index n2|u2 and limit n1|u1 and continue executing immediately following ?DO. Anything already on the return stack becomes unavailable until the loop control parameters are discarded. An ambiguous condition exists if n1|u1 and n2|u2 are not both of the same type.

See: 3.2.3.2 Control-flow stack, 6.1.0140 +LOOP , 6.1.1240 DO , 6.1.1680 I , 6.1.1760 LEAVE , 6.1.2380 UNLOOP , A.6.2.0620 ?DO

6.2.0700 **AGAIN**
CORE EXT

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: (--)

Continue execution at the location specified by dest. If no other control flow words are used, any program code after AGAIN will not be executed.

See: 6.1.0760 BEGIN , A.6.2.0700 AGAIN

6.2.0855 **C"**
c-quote CORE EXT

Interpretation: Interpretation semantics for this word are undefined.

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

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

Run-time: (-- c-addr)

Return c-addr, a counted string consisting of the characters ccc. A program shall not alter the returned string.

See: 3.4.1 Parsing, 6.1.2165 S" , 11.6.1.2165 S" , A.6.2.0855 C"

6.2.0873 **CASE**
CORE EXT

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: -- case-sys)

Mark the start of the CASE ... OF ... ENDOF ... ENDCASE structure. Append the run-time semantics given below to the current definition.

Run-time: (--)

Continue execution.

See: A.6.2.0873 CASE

6.2.0945 **COMPILE,**
compile-comma CORE EXT

Interpretation: Interpretation semantics for this word are undefined.

Execution: (xt --)

Append the execution semantics of the definition represented by xt to the execution semantics of the current definition.

See: A.6.2.0945 COMPILE,

6.2.0970 **CONVERT**
CORE EXT

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

ud2 is the result of converting the characters within the text beginning at the first character after c-addr1 into digits, using the number in BASE, and adding each digit to ud1 after multiplying ud1 by the number in BASE. Conversion continues until a character that is not convertible is encountered. c-addr2 is the location of the first unconverted character. An ambiguous condition exists if ud2 overflows.

Note: This word is obsolescent and is included as a concession to existing implementations. Its function is superseded by 6.1.0570 >NUMBER.

See: 3.2.1.2 Digit conversion, A.6.2.0970 CONVERT

6.2.1342 **ENDCASE**
end-case CORE EXT

```
Interpretation: Interpretation semantics for this word are undefined.

Compilation: ( C: case-sys -- )
```

Mark the end of the CASE ... OF ... ENDOF ... ENDCASE structure. Use case-sys to resolve the entire structure. Append the run-time semantics given below to the current definition.

```
Run-time: ( x -- )
```

Discard the case selector x and continue execution.

See: A.6.2.1342 ENDCASE

6.2.1343 **ENDOF**
end-of CORE EXT

```
Interpretation: Interpretation semantics for this word are undefined.

Compilation: ( C: case-sys1 of-sys -- case-sys2 )
```

Mark the end of the OF ... ENDOF part of the CASE structure. The next location for a transfer of control resolves the reference given by of-sys. Append the run-time semantics given below to the current definition. Replace case-sys1 with case-sys2 on the control-flow stack, to be resolved by ENDCASE.

```
Run-time: ( -- )
```

Continue execution at the location specified by the consumer of case-sys2.

See: A.6.2.1343 ENDOF

6.2.1350 **ERASE**
CORE EXT

```
( addr u -- )
```

If u is greater than zero, clear all bits in each of u consecutive address units of memory beginning at addr .

6.2.1390 **EXPECT**
CORE EXT

```
( c-addr +n -- )
```

Receive a string of at most +n characters. 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 of characters are implementation-defined.

Input terminates when an implementation-defined line terminator is received or when the string is +n characters long. When input terminates, nothing is appended to the string and the display is maintained in an implementation-defined way.

Store the string at c-addr and its length in SPAN.

Note: This word is obsolescent and is included as a concession to existing implementations. Its function is superseded by 6.1.0695 ACCEPT.

See: A.6.2.1390 EXPECT

6.2.1485 **FALSE**
CORE EXT

```
( -- false )
```

Return a false flag.

See: 3.1.3.1 Flags

6.2.1660 **HEX**
CORE EXT

(--)

Set contents of BASE to sixteen.

6.2.1850 **MARKER**
CORE EXT

(**"<spaces>name"** --)

Skip leading space delimiters. Parse name delimited by a space. Create a definition for name with the execution semantics defined below.

name Execution: (--)

Restore all dictionary allocation and search order pointers to the state they had just prior to the definition of name. Remove the definition of name and all subsequent definitions. Restoration of any structures still existing that could refer to deleted definitions or deallocated data space is not necessarily provided. No other contextual information such as numeric base is affected.

See: 3.4.1 Parsing, 15.6.2.1580 FORGET , A.6.2.1850 MARKER

6.2.1930 **NIP**
CORE EXT

(x1 x2 -- x2)

Drop the first item below the top of stack.

6.2.1950 **OF**
CORE EXT

Interpretation: Interpretation semantics for this word are undefined.

Compilation: (C: -- of-sys)

Put of-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 of-sys such as ENDOF.

Run-time: (x1 x2 -- | x1)

If the two values on the stack are not equal, discard the top value and continue execution at the location specified by the consumer of of-sys, e.g., following the next ENDOF. Otherwise, discard both values and continue execution in line.

See: 6.2.0873 CASE , 6.2.1342 ENDCASE , A.6.2.1950 OF

6.2.2000 **PAD**
CORE EXT

(-- c-addr)

c-addr is the address of a transient region that can be used to hold data for intermediate processing.

See: 3.3.3.6 Other transient regions, A.6.2.2000 PAD

6.2.2008 **PARSE**
CORE EXT

(char **"ccc<char>"** -- c-addr u)

Parse ccc delimited by the delimiter char.

c-addr is the address (within the input buffer) and u is the length of the parsed string. If the parse area was empty, the resulting string has a zero length.

See: 3.4.1 Parsing, A.6.2.2008 PARSE

6.2.2030 **PICK**
CORE EXT

```
( xu ... x1 x0 u -- xu ... x1 x0 xu )
```

Remove u. Copy the xu to the top of the stack. An ambiguous condition exists if there are less than u+2 items on the stack before PICK is executed.

See: A.6.2.2030 PICK

6.2.2040 **QUERY**
CORE EXT

```
( -- )
```

Make the user input device the input source. Receive input into the terminal input buffer, replacing any previous contents. Make the result, whose address is returned by TIB, the input buffer. Set >IN to zero.

Note: This word is obsolescent and is included as a concession to existing implementations.

See: A.6.2.2040 QUERY , RFI 0006.

6.2.2125 **REFILL**
CORE EXT

```
( -- flag )
```

Attempt to fill the input buffer from the input source, returning a true flag if successful.

When the input source is the user input device, attempt to receive input into the terminal input buffer. If successful, make the result the input buffer, set >IN to zero, and return true. Receipt of a line containing no characters is considered successful. If there is no input available from the current input source, return false.

When the input source is a string from EVALUATE, return false and perform no other action.

See: 7.6.2.2125 REFILL , 11.6.2.2125 REFILL , A.6.2.2125 REFILL

6.2.2148 **RESTORE-INPUT**
CORE EXT

```
( xn ... x1 n -- flag )
```

Attempt to restore the input source specification to the state described by x1 through xn. flag is true if the input source specification cannot be so restored.

An ambiguous condition exists if the input source represented by the arguments is not the same as the current input source.

See: A.6.2.2182 SAVE-INPUT

6.2.2150 **ROLL**
CORE EXT

```
( xu xu-1 ... x0 u -- xu-1 ... x0 xu )
```

Remove u. Rotate u+1 items on the top of the stack. An ambiguous condition exists if there are less than u+2 items on the stack before ROLL is executed.

See: A.6.2.2150 ROLL

6.2.2182 **SAVE-INPUT**
CORE EXT

```
( -- xn ... x1 n )
```

x1 through xn describe the current state of the input source specification for later use by RESTORE-INPUT.

See: A.6.2.2182 SAVE-INPUT

6.2.2218 **SOURCE-ID**
source-i-d CORE EXT

```
( -- 0 | -1 )
```

Identifies the input source as follows:

```
SOURCE-ID       Input source
-1              String (via EVALUATE)
 0              User input device
```

See: 11.6.1.2218 SOURCE-ID

6.2.2240 **SPAN**
CORE EXT

```
( -- a-addr )
```

a-addr is the address of a cell containing the count of characters stored by the last execution of EXPECT.

Note: This word is obsolescent and is included as a concession to existing implementations.

6.2.2290 **TIB**
t-i-b CORE EXT

```
( -- c-addr )
```

c-addr is the address of the terminal input buffer.

Note: This word is obsolescent and is included as a concession to existing implementations.

See: A.6.2.2290 TIB , RFI 0006.

6.2.2295 **TO**
CORE EXT

```
Interpretation: ( x "<spaces>name" -- )
```

Skip leading spaces and parse name delimited by a space. Store x in name. An ambiguous condition exists if name was not defined by VALUE.

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

Skip leading spaces and parse name delimited by a space. Append the run-time semantics given below to the current definition. An ambiguous condition exists if name was not defined by VALUE.

```
Run-time: ( x -- )
```

Store x in name.

Note: An ambiguous condition exists if either POSTPONE or [COMPILE] is applied to TO.

See: 13.6.1.2295 TO , A.6.2.2295 TO

6.2.2298 **TRUE**
CORE EXT

```
( -- true )
```

Return a true flag, a single-cell value with all bits set.

See: 3.1.3.1 Flags, A.6.2.2298 TRUE

6.2.2300 **TUCK**
CORE EXT

```
( x1 x2 -- x2 x1 x2 )
```

Copy the first (top) stack item below the second stack item.

6.2.2330 **U.R**
u-dot-r CORE EXT

```
( u n -- )
```

Display u right aligned in a field n characters wide. If the number of characters required to display u is greater than n, all digits are displayed with no leading spaces in a field as wide as necessary.

6.2.2350 **U>**
u-greater-than CORE EXT

```
( u1 u2 -- flag )
```

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

See: 6.1.0540 >

6.2.2395 **UNUSED**
CORE EXT

```
( -- u )
```

u is the amount of space remaining in the region addressed by HERE , in address units.

6.2.2405 **VALUE**
CORE EXT

```
( x "<spaces>name" -- )
```

Skip leading space delimiters. Parse name delimited by a space. Create a definition for name with the execution semantics defined below, with an initial value equal to x.

name is referred to as a **value**.

name Execution: (-- x)

Place x on the stack. The value of x is that given when name was created, until the phrase **x TO name** is executed, causing a new value of x to be associated with name.

See: 3.4.1 Parsing, A.6.2.2405 VALUE , 6.2.2295 TO

6.2.2440 **WITHIN**
CORE EXT

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

Perform a comparison of a test value n1|u1 with a lower limit n2|u2 and an upper limit n3|u3, returning true if either (n2|u2 < n3|u3 and (n2|u2 <= n1|u1 and n1|u1 < n3|u3)) or (n2|u2 > n3|u3 and (n2|u2 <= n1|u1 or n1|u1 < n3|u3)) is true, returning false otherwise. An ambiguous condition exists if n1|u1, n2|u2, and n3|u3 are not all the same type.

See: A.6.2.2440 WITHIN

6.2.2530 **[COMPILE]**
bracket-compile CORE EXT

```
Intrepretation: Interpretation semantics for this word are undefined.

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

Skip leading space delimiters. Parse name delimited by a space. Find name. If name has other than default compilation

semantics, append them to the current definition; otherwise append the execution semantics of name. An ambiguous condition exists if name is not found.

See: 3.4.1 Parsing, 6.1.2033 POSTPONE , A.6.2.2530 [COMPILE]

6.2.2535 \
backslash CORE EXT

Compilation: Perform the execution semantics given below.

Execution: (**"ccc<eol>"**--)

Parse and discard the remainder of the parse area. \ is an immediate word.

See: 7.6.2.2535 \ , A.6.2.2535 \

 Table of Contents
 Next Section