,   (n --)   "comma"

'Comma' compiles N into the dictionary at HERE and advances the dictionary pointer by 4. It can be used to supplement CREATEd data structures in the dictionary.

```
\ Create an array called sizes
CREATE SIZES 330 , 250 ,  230 , 470 ,
2 CELLS SIZES + @ .  ( prints 230 )
```

`Related Words: ALLOT  HERE  !  @  W,`

-   (a b -- a-b)   "minus"

Subtract top of stack B from next to top A. Result is put on top of stack.

```
10 7 - .  ( prints 3 )
```

`Related Words: + CELL- 2-  1-  D-`

-2SORT   (n1 n2 -- biggest smallest)   "dash 2 sort"

Sort top two items on stack into reverse order.

```
: SAFEDO  -2SORT DO ." hello" cr LOOP ;
10 0 SAFEDO
0 10 SAFEDO  ( both work )
```

`Related Words: 2SORT MIN MAX`

->   (n <name> --)   "to"

Store N into a value data structure or a local variable. VALUE data structures and local variables are self fetching so you can't use Reverse Polish words like ! . See VALUE for example.

File: JU:VALUE or JU:LOCALS

-CLIST -CONSOLE -DISKFONT -GRAPHICS -ICON -INTUITION -LAYERS

Close the appropriate Amiga Library IF open. See LIBRARY . This should be done before terminating an application on each library that had been opened.

-DO "minus do"

(to from --)   (--R-- previous-loop-parameters)

Begin a DO LOOP without checking the to and from values.

It is often used when you want to count down instead of up, in a loop, and is normally used with -LOOP as a result. You may also use -DO if you want to save space and time in setting up ordinary loops, where you do not need the limit checking of DO.

The TO an FROM values are processed at run time into limit-overflow values and placed in 2 loop parameter data registers.

```
\ Countdown from 10 to 0. Note 11 loops.
: COUNTDOWN   0 10 -DO I . 1 -LOOP ;
```

Note that the following loop with -DO will execute once. The equivalent loop with DO will not execute.

```
: ONCE  0 0 -DO ." Hello" cr LOOP ;
```

`Related Words: DO LOOP -LOOP +LOOP`

-DOS (--) "minus dos"

This word has no effect in JForth. Two libraries, EXEC and DOS are opened by the JForth startup routines, and should only be closed by BYE. It is provided to be compatible with the word sets provided for the other library definitions.

-DUP (n1 -- n1 n1 | 0) "dash dup"

Duplicates the top of the stack only if it is non zero. This can save you from having to do ELSE DROP.

Standards: '79 and '83 use ?DUP for this definition. FIG uses -DUP

```
: MAYBECLOSE  ( filepointer -- , close if safe )
    -DUP IF FCLOSE THEN ;
```

Related Words: ?DUP DUP XDUP IF 0=

-EXEC (--) "minus exec"

This word has no effect in JForth. Two libraries, EXEC and DOS are opened by the JForth startup routines, and should only be closed by BYE. It is provided to be compatible with the word sets provided for the other library definitions.

-FIND (<word> -- pfa cntadr true | 0) "dash find"

FIG standard version of FIND. Not recommended.

File: JU:MULTISTANDARD

-LIB (var-holding-lib-pointer --) "minus lib"

This word is the primitive used by all the library closing routines.

It accepts the address of a VARIABLE or USER-variable that contains the pointer to an open library. If the var contains 0, -LIB does nothing. The word set provided for handling Amiga libraries usually precludes the necessity for the programmer to use -LIB.

Related Words: LIBRARY

-LOOP (n --) "minus loop"

-LOOP works like +LOOP but subtracts the value on the stack instead of adding it. -LOOP is normally used with -DO for loops where the count is counting down instead of up. See -DO

Related Words: -DO +LOOP DO LOOP (-LOOP) (+LOOP)

-MATHFFP -MATHIEEEDOUBBAS -MATHIEEESINGBAS -MATHTRANS -POTGO

(--)

These words will close the appropriate Amiga library, if it is open. If not, there is no effect.

Related Words: LIBRARY

-ROT (a b c -- c a b) "minus rot"

Rotate the top three items on the stack in the reverse direction that ROT would. Equivalent to but faster than using ROT twice.

-SHIFT (n s -- n>>s) "minus shift"

Shift n by s positions to the right.

-SHIFT is a shorter, faster version of SHIFT that only does negative shifts toward the LSB. See

SHIFT for more info. Zeros are shifted into the MSB so be careful with negative numbers.

```
     100 2 -shift . ( prints 25 )
```

Related Words: SHIFT +SHIFT ASHIFT U2/

-STACK   (value var-address --)   "minus stack"

```
     value           = item that is searched for and removed
     var-address     = address of a variable
```

-STACK is used to remove items from a memory block that is being used as a stack. It expects the address of a VARIABLE or USER-variable that points to the allocated memory area, and the value to find and remove. If the VARIABLE or USER-variable contains 0, nothing will happen. Otherwise, the stack area is searched for the value, removed if found, and the FREEBYTE counter for the area updated. If -STACK causes the stack area to become empty, it will free the memory area, clearing the pointer address that was passed as an argument. See section on memory management.

Related Words: ALLOCBLOCK +STACK PUSH POP

-TRAILING (addr count -- addr count') "dash trailing"

-TRAILING adjusts the count of a string to remove trailing blanks. It does this by scanning backward from the last character toward the first character and the first non-blank character stops the scan.

```
     " Hello     " COUNT .S -TRAILING .S TYPE
```

Related Words: COUNT TYPE SKIP

.   (n --)   "dot"

Print the integer number on the top of the stack. Use the value of BASE as the current numeric base. Print a space after the number.

```
     2 3 + .  ( will print "5 " )
```

Related Words: U. .R D. D.R .HEX BASE ? N>TEXT F.

."   (<text> --)   "dot quote"

Print a text string terminated with a ".

If used inside a colon definition, the text will be compiled into the word for later printing. This word will also work outside of colon definitions although the Forth '83 standard says it should fail.

With buffered I/O, the output may not appear until you call CR KEY or FLUSHEMIT. See FAST .

```
     ." Hello world!"
```

Related Words: (.") TYPE CR

.. (struct-addr <member> -- member-addr) "dot dot"

Return the address of a member in a 'C' like structure. See chapter on the Amiga 'C' structure interface.

```
     SCREEN MY-SCREEN ( declare screen structure )
     MY-SCREEN .. SC_RASTPORT ( -- address-rastport-member)
```

File: JU:C_STRUCT

..!   (n struct-addr <member> --)   "dot dot store"

Set the member of a structure to n. C! , W! or ! will be used depending on the width of the member. See chapter on the Amiga 'C' structure interface.

```
     NEWSCREEN MY-NEWSC ( declare screen structure )
     3 MY-NEWSC ..! NS_DEPTH  ( set number of bitplanes )
```

File: JU:C_STRUCT

..@ (struct-addr <member> -- n) "dot dot fetch"

Fetch the value of a structure member. C@ , W@ or @ will be used depending on the width of the member. See chapter on the Amiga 'C' structure interface.

```
     MY-WINDOW ..@ WD_WIDTH ( fetch width of window )
```

File: JU:C_STRUCT

.ELSE   (--)   "dot else"

Conditional compilation word used with .IF . See .IF

.ERR   (--)   "dot err"

.ERR is useful when reporting errors. It clears the stack, executes CR , then TYPEs the text at HERE, in preparation for your error message and subsequent QUIT.

```
     : ?HaveError   ( flag -- , quits with error message if error )
         IF .ERR  ." flag was not Zero!"  QUIT
         THEN  ;

     \ Here is an example with the resulting printout.
     0 ?HaveError  ok
     1 ?HaveError
     ?HAVEERROR ? ... flag was not Zero!
```

`Related Words: HERE ABORT" QUIT TYPE`

.FILE   (<filename> --)

Uses DOLINES to print the contents of a file with line numbers. Similar to TYPEILE .

File: JU:DOLINES

.HEX   (n --)   "dot hex"

Print the top of stack as a hexadecimal, HEX , number regardless of the current base.

```
     DECIMAL 17 .HEX  ( prints 11 )
```

.HX   (n --)   "dot h x"

Prints one digit. This will print a digit in any base up to 36.

Equivalent definition:

```
     : .HX  ( n -- )
         DUP 9 > IF 37 + ELSE 30 + THEN EMIT ;
```

`Related Words: .HEX  .S`

.IF   (flag --)   "dot if"

.IF , along with .ELSE and .THEN , allows for conditional compilation. These words work like IF ELSE THEN but they are immediate words and work in the interpret state as well. JForth source files include numerous examples of the use of these conditional words.

.IF and related operators are NOT nestable.

```
     false .IF  : FOO ( one way ) ;
```

```
.ELSE  : FOO ( a different way ) ;
.THEN
```

Related Words: .IF .ELSE .NEED EXISTS? .THEN

.IMAGE   (--)   "dot image"

.IMAGE will report whether the current image is:

Relocatable (contains LONG-ABSOLUTE address references, and if so, how many)

OR

Position-Independent (the compiler has been able to use relative addressing modes for all threading).

A LONG-ABSOLUTE reference (JSR) will be compiled only if: 1) the calling program is over 32K away from the called program... AND 2) the called program is not resident in the bottom 96K of JForth.

Related Words: JSR-CODE MAP CFA,

.MODULES   (--)

Display state of detachable modules. See chapter on modules.

.NEED   (--)   "dot need"

.NEED is a conditional compilation word that works with .THEN and .ELSE . If the word following .NEED is in the dictionary, .NEED will skip to the .ELSE or .THEN part of the conditional compilation structure. If the word does not exist, it will continue processing the text that follows.

```
.NEED WIERD ( WIERD ain't in the dictionary  ... )
    : WIERD   CR ." Like wow, man. " ;
.ELSE  cr ." this place is already weird"  ( WIERD is around)
.THEN
```

Related Words: .IF .ELSE .THEN EXISTS?

.R   (n width --)   "dot r"

Print N right justified in a field WIDTH characters wide. This is useful for preparing formatted output like tables, etc. If the number is wider than WIDTH it will still print all the digits.

```
742  12  .R    ( prints  ___742 )
   ( 7 spaces to the left )
```

Related Words: D.R . # COMMAS

.RSTACK   (--)   "dot r stack"

Print the contents of the return stack.

File: JU:.RSTACK

.S   (--)   "dot s"

Print the contents of the data stack. The number on the right is the top of stack. The stack will be unchanged by the operation. Often placed inside troublesome words when debugging.

Related Words: DEPTH DEBUG SP@ .RSTACK

.THEN   (--)   "dot then"

Terminate a .IF or .NEED conditional construct. See .IF for more details.

/   (a b -- a/b)   "slash" or "divide"

Divide the top two items on the stack and leave the result. This is an integer 32 bit divide. The answer will be truncated. Negative results will be rounded towards zero. Use /MOD if you want the remainder.

```
     33 5 / . ( prints 6 )
     -17 8 / . ( prints -2 )
```

`Related Words: /MOD M/  CELL/  D2/  2/  W/  U/  DU2/  U2/`

/MOD   (a b -- remainder a/b)   "slash mod"

Performs 32 bit integer divide of A by B. Leave remainder and quotient.

```
     73 10 /MOD .S  ( prints 3 7 )
```

`Related Words: /  MOD`

/STRING (addr cnt index -- addr' cnt')"slash string"

/STRING indexes into the string at addr, returning the address and count of the remaining string.

```
     " Hello World" COUNT 6 /STRING TYPE ( prints World )
```

0   (-- 0)   "zero"

Defined as a constant to speed up compilation.

0"   (<text> -- 0string)   "zero quote"

Convert the following text to a NUL terminated string. The text should be terminated by a ". NUL, or zero, terminated strings are used by 'C' and are therefore needed when passing strings to the Amiga Libraries. The parameter 0string is the address of the first character in the string.

```
     0" Hello" 0COUNT TYPE
```

`Related Words: 0COUNT >ABS $>0 DOS0 " $APPEND 0FOPEN`

0<   (n -- flag)   "zero less than"

Compares the top of stack to zero. Leave TRUE if less than zero, otherwise leaves FALSE.

```
     : TEST 0< IF ." Negative!" THEN ;
     -5 TEST ( print Negative!)
```

`Related Words: 0> 0= +-`

0=   (n -- flag)   "zero equals"

Compares the top of stack to zero. Leave TRUE if equal to zero, otherwise leaves FALSE. Can be used like NOT to complement a flag.

`Related Words: 0> NOT IF`

0>   (n -- flag)   "zero greater than"

Compares the top of stack to zero. Leave TRUE if greater than zero, otherwise leaves FALSE.

`Related Words: 0< 0=`

0BRANCH   (flag --)   "zero branch"

0BRANCH is compiled into a definition by IF UNTIL WHILE to do a conditional branch. If flag = FALSE, the program will branch to another location. If flag = TRUE, execution continues with the next instruction.

Standards: FIG, called ?BRANCH in '83 and '79 .

`Related Words: IF  WHILE  UNTIL  BRANCH  NOT0BRANCH`

`0COUNT   ( 0string -- addr count )   "zero count"`

Count the characters in a NUL terminated string.

```
0" Hello" 0COUNT TYPE
```

`0FOPEN   ( 0name -- filepointer | 0 )   "zero f open"`

This word takes the address of a NULL-TERMINATED string and submits it to AmigaDOS to request it be opened as a filename.   If the file can be successfully opened, its pointer is returned, else a FALSE is returned. This pointer may be passed in other file utilities to initiate reads, writes, seeks, etc.

Please see the chapter on File I/O for more information.

Related Words: FILE? FOPEN $FOPEN FREAD

`0RP   ( -- )   "zero r p"`

0RP initializes RP to the value in the user variable RP0 . RP = return stack pointer. This is usually used in error recovery.

Warning: This clears the subroutine calling stack of the 680x0 so be very careful when using this.

`Related Words: RP0 RP!`

`0SP   ( a? b? whatever -- , empty )   "zero s p"`

Clear the data stack by initializing SP to the value in the user variable SP0 . SP = stack pointer. Remember the top of stack is cached in D7. Only recommended for interactive use and in error recovery.

```
11 22 33 0SP  ( stack now empty )
```

`Related Words: SP0 SP!`

`1   ( -- 1 )`

Defined as a constant to speed compilation.

`1+   ( n -- n+1 )   "one plus"`

```
Add one to the value n on top of the stack.
7 1+ . ( print 8 )
```

`Related Words: 2+  1- 2- 2* 2/ + -`

`1-   ( n -- n-1 )   "one minus"`

Subtract one from the value n on top of the stack.

`Related Words: + - 1+ 2+ 2- 1 2`

`2   ( -- 2 )`

Defined as a constant to speed compilation.

`2!  ( d addr -- )   "two store"`

Store a 64 bit double number at the given address. Only even addresses are allowed on a 68000 system. Use ODDD! for odd address double stores. 2! is a synonym for D! .

Related Words: D! ODD-D!

2*   (n -- n*2)   "two times"

Multiply top of stack by 2.

```
    -5 2* . ( print -10 ) 5 2* . ( print 10 )
```

2N   (n -- 2**n)   "two to the n"**

Raise two to the Nth power.

```
    3 2**N . ( print '8'  = 2*2*2 )
```

File: JU:BSORT

2+   (n -- n+2)   "two plus"

Add two to the value n on the top of the stack.

```
    85  2+  .  ( prints 87 )
```

2-   (n -- n-2)   "two minus"

Subtract two from the value n on the top of the stack.

2/   (n -- n/2)   "two slash"

Divide the top of stack by two. This uses an arithmetic left shift which is much faster than the normal divide.

Related Words: 2* U2* U2/ DU2/ DU2* W/ /MOD U/ M* U* D2/ / M/ DIGS/ */MOD

2@   (addr -- d)   "two fetch"

Fetch a double number from the given address. Only even addresses are allowed. See D@ .

Related Words: D@ ODD-D@ X@

2DROP   (a b --)

Drop 2 cells from top of stack.

Standards: '83 and FIG . Called DDROP in '79 .

Related Words: DROP

2DUP   (a b -- a b a b)

Duplicates the top pair of stack values.

Standards: '83 and FIG . Called DDUP in '79 .

Related Words: DUP

2OVER   (a b c d -- a b c d a b)

Copies pair of numbers over top pair.

Related Words: OVER DUP 2DUP DOVER

2SORT   (n1 n2 -- smallest biggest)   "2 sort"

Sort two numbers.

Related Words: -2SORT MIN MAX

`2SWAP   ( a b c d -- c d a b )`

Swaps the top two pairs of stack values.

`Related Words: SWAP 2DUP`

`3   ( -- 3 )`

Defined as a constant to speed compilation.

`32K   ( -- 32768 )   "thirty-two k"`

Constant equal to 32 * 1024.

`64K   ( -- 65,536 )   "sixty four k"`

Constant equal to 64 * 1024