**ABORT ( -- )**

Abort execution, return to outer interpreter. No message is displayed.

```
: EXAMPLE ( N -- )  1,000,000 >
   IF  CR ." Oh my God! We're going TOO HIGH!!!" CR
       ABORT  ( ABORT" would be better here )
       ( will QUIT with no "OK" message )
   THEN ;
```

ABORT is a deferred word that normally calls QUIT.

Related Words: ABORT" WARNING" .ERR QUIT

---

**ABORT" ( flag <message> -- ) "abort quote"**

Display message and abort if the flag is true.

```
: EXAMPLE ( N -- )  1,000,000 >
   ABORT" Oh my God, We're going TOO HIGH!!!" ;
```

Related Words: ABORT QUIT ?ABORT" WARNING"

---

**ABS ( n -- |n| ) "absolute"**

Calculate absolute value of N.

```
5 ABS .
-5 ABS .  ( both will print positive 5 )
```

Related Words: DABS

---

**ABS! ( n absolute-address -- ) "a b s store"**

Store longword, word, or byte at an absolute 68000 absolute address. Equivalent to >REL ! but faster. Useful for accessing hardware.

Related Words: >ABS ! W! C! ABS@ >REL

---

**ABSW! ( n absolute-address -- ) "a b s w store"**

See ABS!

---

**ABSC! ( n absolute-address -- ) "a b s c store"**

See ABS!

---

**ABS@ ( absolute-address -- n ) "a b s fetch"**

ABSW@ ( absolute-address -- n ) "a b s w fetch"

ABSC@ ( absolute-address -- n ) "a b s c fetch"

Fetches longword, word, or byte from absolute address. Equivalent to >REL @ but faster. Useful for accessing hardware.

Related Words: >ABS @ W@ C@ ABS!

---

**ADST ( absolute-addr <structure> -- )**

Dump a structure whose absolute address is on the stack. Similar to DST which takes a relative address. See DST .

File: JU:DUMP_STRUCT

**AGAIN ( -- )**

AGAIN at execution time: marks the end of an infinite loop and branches back to its corresponding BEGIN . Warning - an infinite loop won't stop until QUIT or RETURN is executed.

At compile time : compiles a BRANCH into the dictionary. Resolves the loop entry point address provided by BEGIN into a return branch offset and stores this offset in the dictionary. AGAIN and BEGIN are paired. Error is detected by ?PAIRS if no match.

```
\ Warning this word will never stop!!!
: ETERNITY BEGIN ." Forever is a long time!" AGAIN ;
```

Related Words: BEGIN BACK BRANCH DO UNTIL WHILE

**ALIAS ( <oldname> <newname> -- )**

Create an alternative name for an existing word.

```
ALIAS 2DUP DDUP
```

**ALIGN ( -- )**

Word-align the dictionary pointer DP .

ALIGN is used at the end of any dictionary-allocation process that may have left HERE at an odd address (the 68000 CPU and JForth both require the dictionary to be word aligned). ALIGN will allocate a byte in the dictionary if DP is odd; no effect if even.

Related Words: C, CARRAY ALLOT EVEN-UP

**ALLOCBLOCK ( memtype size -- addr | false )**

Allocates a memory area of the given size and type and returns the JForth-relative address of the block. If the call for memory is not successful, a FALSE flag is returned.

The programmer will call ALLOCBLOCK to get an area of memory from the free memory list maintained by AmigaDOS. JForth will remember the block and return it to AmigaDOS at BYE , if the programmer does not.

This call is a direct interface to the Amiga EXEC function, AllocMem. It returns, however, a JForth relative address, usable by all the JForth memory-reference words, such as @, C@, !, C!, etc.

The type parameter is referenced as in Amiga literature:

```
MEMF_CHIP   - memory in lowest 512K, accessible by graphics
              and sound hardware.
MEMF_FAST   - memory above 512K, cannot be used by hardware
              chips.
MEMF_PUBLIC - memory is to be used for different tasks or
              interrupt code, and may apply to task control
              blocks, messages, ports, etc.
MEMF_CLEAR  - this parameter may be specified to clear the
              memory upon allocation; otherwise no
              initialization is done.
```

These parameters may be combined (as long as they are not self-cancelling) by logically ORing them together. For example, to indicate a global area that may be used by the graphics chips, this argument would be formed:

```
MEMF_CHIP    MEMF_PUBLIC  OR
```

ALLOCBLOCK also provides a range of operators that may be used to expand the functionality of the memory block to act as stacks and buffers, and/or automatically be freed at QUIT .

The address returned from a successful call may subsequently be passed to FREEBLOCK to return the memory to AmigaDOS when no longer needed.

Later versions of the Amiga may support 1024K of CHIP RAM.

Related Words: FREEBLOCK MARKFREEBLOCK MEMF_CHIP MEMF_FAST MEMF_PUBLIC MEMF_CLEAR FREEBYTE FREEBYTEA PUSH POP -STACK +STACK SIZEMEM

---

**ALLOCBLOCK? ( memtype size -- addr )**

Identical to ALLOCBLOCK except this word reports an error and aborts if memory could not be allocated.

---

**ALLOT ( numbytes -- )**

ALLOT allocates space in the dictionary by advancing the dictionary pointer DP by N bytes.

NOTE: if you ALLOT an odd number of bytes, the word ALIGN must be called before any operations involving HERE are allowed to occur. It does not hurt to call ALIGN for even numbers so use ALIGN liberally.

VARIABLE CHEAP-ARRAY 96 ALLOT ( has room for 100 bytes )

Related Words: DP HERE ALIGN

---

**ALSO ( voc --voc-stack-- voc voc )**

Duplicate top of vocabulary stack. Used to make room for vocabulary in the current context. See the section on vocabularies.

```
VOCABULARY MUSIC  ( create a new vocabulary )
ALSO MUSIC  ORDER ( also search the MUSIC vocabulary )
PREVIOUS     ( put things back the way they were )
```

---

**AND ( a b -- a&b )**

Does logical AND on each pair of corresponding bits in A and B. Both bits an A AND B must be 1 for bit in result to be 1. Useful for masking data.

```
HEX 7E53 FF AND .  ( print 53 )
: EVEN?  ( N -- flag , true if odd ) 1 AND 0= ;
```

Related Words: OR XOR

---

**ANEW ( <name> -- ) "a new"**

Forgets a word if already defined, no effect otherwise. This is often used at the beginning of a file that is under development. Every time you recompile the file, it will automatically forget the code compiled the last time. The filename is typically appended to the prefix TASK- .

```
\ Roll dice.
INCLUDE? CHOOSE JU:RANDOM
ANEW TASK-DICE
: ROLL.DICE ( -- N )
    6 CHOOSE 1+
;
```

Load the file by entering:

```
INCLUDE DICE
ROLL.DICE .
```

You can now edit this file, adding and changing code. You can then include this file again and again

without getting multiple definitions of your code. Notice that the INCLUDE? is placed before the ANEW. This is so CHOOSE won't be forgotten each time.

One thing to remember in using ANEW is that if you load a bunch of files after loading DICE , then reload DICE, you will lose the definitions from the other files. (The actual files will be unaffected.) This is only a problem when you are loading a file that stops because it contains a word that is undefined. If you load the file that contains that undefined word, and then try to reload your file, the first thing ANEW will do is forget the code you just loaded. The same word will show up missing agin. You can while away many a rainy day stuck in this loop. The thing to do is to first FORGET your task word. Consider the following sequence:

```
INCLUDE MYFILE
( assume myfile crashes because of an undefined word )
( edit myfile to do an INCLUDE? for that word)
FORGET TASK-MYFILE
INCLUDE MYFILE
```

Unless you have more undefined words, you can continue just using INCLUDE MYFILE .

Related Words: FORGET IF.FORGOTTEN

---

**ANSI.BACKWARDS ( n -- )**

Move cursor N characters backwards. This is only one of a number of ANSI based "terminal editing" commands that can be found in the file JU:ANSI.

---

**APTR ( <name> -- ) "a pointer"**

Define a 32 bit pointer member in a structure. See the section on Amiga 'C' structures.

File: JU:MEMBER

---

**AREGS>ABS ( -- ) " a regs to a b s"**

Set the flag CONVERT-AREGS so that the very next Amiga CALL will convert parameters in address registers to absolute before passing them to the Amiga. This will save you having to call >ABS excessively. See chapter on Calling Amiga Libraries.

---

**ARGS ( <library_lib> <routine_name> -- )**

ARGS looks up arguments for an Amiga Library call and displays them. It provides a form of automatic documentation. It looks in the FD files so there must be one for the library you specify. The library need not be open.

```
ARGS GRAPHICS_LIB DRAW

Will display:  Draw(rastPort,x,y)(A1,D0/D1)
```

This tells you that the graphics DRAW routine takes 3 arguments. Place the absolute address of a RASTPORT on the stack followed by x and y values, then use CALL to invoke the routine. JForth will automatically build the necessary code to stuff the 68000 registers and make the call.

Related Words: CALL DCALL

---

**ARRAY ( numcells <name> -- )**

Creates a one dimensional array of length NUMCELLS of 32 bit values starting at the next available dictionary location . Cells are initialized at compile time to zero's. The index of the first array item is zero just like in 'C'. Let's create an array with 20 items.

```
20 ARRAY MY-ARRAY
```

The array that was created has the following stack diagram:

**MY-ARRAY ( index -- addr )**

Now lets store a value in that array and then fetch it back.

```
731 12 MY-ARRAY ! ( store 731 into cell 12 of MY-ARRAY )
0 MY-ARRAY @  ( get first cell value )
```

(ODE, the object oriented dialect, has a fancier type of array.)

Related Words: ALLOT ALLOCBLOCK OB.ARRAY  ARRAYOF

---

**ARRAYOF ( n <structure> <name> -- )**

Create an array of structures. For example:

```
10 ARRAYOF GADGET MY-GADGETS
```

---

**ASCII  ( <char> -- ASCII-value )**

Converts the next character in the input stream to its ASCII equivalent and puts the value on top of the stack .

```
ASCII A .  ( print 65 )
```

---

**ASHIFT  ( n shift-count -- n-shifted )**

Arithmetic shift N by SHIFT-COUNT. Shift left is shift-count is positive, right if negative. Preserve the sign of N when shifting left by "dragging" the sign bit.

```
-20 -2 ASHIFT . ( print -5 )
```

The word SHIFT is similar but does not preserve sign.

---

**ASM  ( -- , <wordname> )**

See the chapter on 68000 ASSEMBLY.

---

**AUTO.INIT  ( -- )**

Used for automatic initialization. You write this word. When JForth starts up, it searches the dictionary for a word called AUTO.INIT and executes the first one it finds. If you have an initialization word that you would like called at startup, define a word called AUTO.INIT that calls the previous AUTO.INIT then calls your word. This will add onto a chain of AUTO.INITS that initialize many parts of JForth.

Code that allocates memory, opens files or libraries, initializes hardware, opens windows or builds tables is often called with AUTO.INIT . The matching terminate code is then called from AUTO.TERM. In general, DO NOT allocate memory open files or windows, etc. at compile time. Place this code in a colon definition and use INIT words.

Here is an example that shows how to automatically initialize a system, as well as automatically clean up on BYE or if the code is forgotten.

```
( variable to avoid double init or term )
VARIABLE IF-MY-INIT
: MY.STARTUP  IF-MY-INIT @ 0=
    IF  ALLOC.STUFF SET.STUFF
        IF-MY-INIT ON
    THEN
;
: AUTO.INIT ( -- , add my.startup to init chain )
    AUTO.INIT ( important!!)
    MY.STARTUP
;
```

```
: MY.TERM  IF-MY-INIT @
    IF  CLEANUP.STUFF FREE.STUFF
        IF-MY-INIT OFF
    THEN
;
: AUTO.TERM MY.TERM AUTO.TERM ; ( called on BYE )
IF.FORGOTTEN MY.TERM  ( call MY.TERM if forgotten )
```

Related Words: AUTO.TERM IF.FORGOTTEN

---

**AUTO.REQUEST ( $body $posi $nega -- flag )**

Put an Amiga Auto Requester on the screen. The body will be the main message. There will be two possible responses, one positive and one negative. These might be "Yes" and "No", or "OK" and "Cancel". When the user selects a response, a TRUE will be returned for a positive response, otherwise FALSE.

File: JU:AUTO_REQUEST

---

**AUTO.TERM ( -- )**

This word is called automatically when JForth exits using BYE. You can define a word called AUTO.TERM to cleanup your code on BYE. See AUTO.INIT for an example of it's use.

---

**B->S ( byte -- sign-extended-byte-value ) "b to s"**

Sign extend a byte value to 32 bits.

```
HEX E7 B->S .HEX ( prints FFFFFFE7 )
```

Related Words: W->S S->D

---

**BACK ( addr -- )**

BACK resolves the supplied address ADDR into a backward branch offset relative to HERE and compiles the offset into the dictionary. It is used internally to compile UNTIL , AGAIN , etc.

---

**BASE ( -- addr )**

BASE is a user variable that contains the current number base used for input and output number conversion. HEX stores sixteen into BASE. DECIMAL stores ten into BASE.

The JForth ok prompt will tell you about the current base:

```
ok      - decimal
ok(hex) - hexadecimal
ok(bin) - binary
ok(7)   - in base 7
```

To display the base in decimal without destroying contents of BASE:

```
: BASE? BASE @ DUP DECIMAL . BASE ! ;

DECIMAL 7 BASE !
6 1 + .  ( print 10 )
```

---

**BEGIN ( -- )**

BEGIN marks the beginning of a loop. May be executed indefinitely as opposed to a DO loop which is limited to a certain number of passes. Used with UNTIL , WHILE , REPEAT and AGAIN.

At compile time, BEGIN places the address of the next available dictionary location onto the stack so UNTIL or REPEAT or AGAIN can compile a return branch into their definition. Begin_flag is

checked by UNTIL or REPEAT or AGAIN for a balanced loop.

compile time ( -- entry_point_address begin_flag )

```
entry_point_address = location of first word of loop.
begin_flag is for compiler security.

BEGIN code ( -- flag ) UNTIL ( quits when true )
BEGIN code  AGAIN
BEGIN code ( -- flag ) WHILE code-if-true REPEAT

: JABBER BEGIN ." Blah blah! " ?TERMINAL UNTIL ;
```

---

**BEGIN_FLAG ( -- n )**

A constant left by the compile time part of BEGIN and used by the compile time part of UNTIL, REPEAT, WHILE , or AGAIN .

---

**BENCH ( <forthword> -- )**

Benchmark a word, subtracting the overhead time found using BENCH.WITH. For example, many words are benchmarked by calling them many times from a DO LOOP. To get an accurate measurement of the word we need to subtract the time for an empty DO LOOP.

```
\ Measure speed of swap.
1,000,000 CONSTANT 1MEG
: TDO   1MEG 0 DO LOOP ; ( empty loop )
: TSWAP 1MEG 0 DO SWAP LOOP ;
BENCH.WITH TDO
23 45 BENCH TSWAP
```

Related Words: MEASURE BENCH.WITH '

---

**BENCH.WITH ( <forthword> -- )**

Determine overhead time for benchmarking Forth words. See BENCH for example.

---

**BINARY ( -- )**

Sets user variable BASE to 2 for binary input and output numeric conversion. The Forth ok prompt will be followed by a (bin) when BASE is set to binary.

Related Words: HEX DECIMAL BASE

---

**BIT-SET? ( n bit# -- flag )**

Test to see if a bit in N is equal to 1. The rightmost, least significant bit is bit number zero.

```
HEX 84 2 BIT-SET? ( true )
```

Related Words: SET-BIT AND OR XOR

---

**BL ( -- 32 ) "b l" or "blank"**

Constant equal to the ASCII value for a blank, or space.

```
: SAYAGAIN BL WORD COUNT TYPE ;
SAYAGAIN hello  ( prints HELLO )
```

Related Words: WORD -FIND

**BLK ( -- addr ) "b l k"**

A user-variable, used only in the BLOCK environment to hold which 1024 byte section of a SCREEN-FILE is being loaded and interpreted . BLK = 0 if not interpreting from a screen file.

WORD looks at BLK to determine the address of input data. QUIT sets BLK to 0.

Related Words: LOAD-FILE >IN BLOCK

**BLKERR ( -- ) "block error"**

This word is the default contents for the DEFERed word, BLOCK.

It serves to print the text at HERE, followed by the message " ... BLOCK not initialized!", finally executing QUIT.

This sequence warns that an attempt to execute BLOCK has been made without having loaded the "JU:BLOCK" file. This file defines the support environment for BLOCK, defining the correct handler to replace BLKERR in the vector of BLOCK.

Related Words: BLOCK QUIT HERE

**BLOCK ( block# -- buffer-addr )**

Returns the address at the start of the 1024 byte buffer numbered BLOCK# . If this block is not already loaded, it loads it from the file providing a system similar to virtual memory. Since JForth normally uses regular ASCII text files, you must load JU:BLOCK to use this facility.

Related Words: BUFFERADR BLK R# LOAD (FIRST) FLUSH INCLUDE

**BODY> ( data-addr -- CFA ) "body from"**

Convert the Body address (location of the data area) of a CREATE or CREATE/DOES> child to its CFA or tick address (start of the executable code). See >BODY.

BODY> will only return a meaningful result if the passed-in address points to the data field of a word created via CREATE, either directly or via a CREATE/DOES> defining word. This does NOT include the data location for other data-structures, such as VARIABLEs, CONSTANTs, or VALUEs.

[Note: In JForth Version 1.2 and earlier, **BODY>** was a noop, implying the **body** and the **cfa** were the same!]

Related Words: >BODY ' >NAME >LINK

**BOTH ( -- )**

BOTH is an immediate compiler directive used just preceding a ; at the end of a definition. Once stated, the compiler will verify that the word being compiled may safely be used as an inline definition.

If so, the word is initialized such that the compiler will either compile an indirect call to it, or move it inline, depending on the value of the user-variable MAX-INLINE when it is being compiled.

If not, a warning message is issued, informing the operator that the word cannot be compiled inline, and the word will be set as CALLED .

```
: EXAMPLE ( -- ) ROT OVER + BOTH ;
```

Related Words: ; INLINE MAX-INLINE CFA,

**BRANCH ( -- )**

Used internally. BRANCH is compiled by certain conditional words to re-direct program flow at run time. BRANCH adds a + or - offset to IP causing an unconditional jump.

Related Words: ELSE AGAIN REPEAT

---

**BSIN ( -- addr ) "b s in" or "back space in"**

Variable containing the value of the backspace character for input. Usually 08 hex is the default value.

Useful for working with different terminals.

Related Words: BSOUT

---

**BSORT ( #items -- ) "b sort"**

Sort items using Batcher sort. To control what gets sorted, you must set the deferred word BSORT-EXCH? . For more information, see the file JU:BSORT and the definition of BSORT-EXCH?

File: JU:BSORT, see also JA:SORTMERGE

Related Words: BSORT-EXCH? 2SORT

---

**BSORT-EXCH? ( index-a index-b -- ) "b sort dash exchange question"**

Called from BSORT. To use BSORT you must write a word that takes the index of two items, and exchanges them if they are in the wrong order. Then set BSORT-EXCH? to call your word. The items that are sorted can be anything including strings, integers, etc.

You could, for example, have a table that contained pointers to entries in a mailing address data base. To sort your table by zip code, write a word that accepts two indices into that table, compares the two mailing addresses they point to, then swap the table entries if they are out of order. Then set BSORT-EXCH? to call your word. When you then call BSORT, it will generate a pattern of indices that will eventually result in your entire table being sorted in a very short time. Please see the demo file JD:DEMO_GSORT for another example.

```
: MY-EXCH?   ( ia ib -- )
    2DUP OUT-OF-ORDER?  ( you must supply this word )
    IF EXCHANGE-THEM    ( you must supply this word )
    ELSE 2DROP
    THEN
;
' MY-EXCH? IS BSORT-EXCH?
25 BSORT
```

File: JU:BSORT

Related Words: BSORT 2SORT ADDR.EXCH?

---

**BSOUT ( -- addr ) "b s out"**

User variable containing the character used to output a backspace. Default is 08 hex.

Useful for working with different terminals.

Related Words: BSIN

---

**BSR-CODE ( -- 6100 , in HEX )**

Constant equal to the machine language 68000 opcode for BSR. BSR op code = 61XX hex where XX = displacement. If XX = 00 hex then the 16 bit word following BSR is used for a displacement value. BSR = Branch Subroutine.

Used internally for 68000 machine coding.

Related Words: DIDCODE JSR-CODE RTS-CODE

**BYE ( -- )**

BYE will exit JForth, returning the memory to the system.

As long as the programmer has utilized the JForth-supplied words for memory allocations, opening & closing files, and opening libraries, JForth will return any of these resources which are still pending.

NOTE: Only the standard predefined libraries are cleaned-up so the programmer should take care to close any custom libraries defined with :LIBRARY.

Related Words: AUTO.TERM

---

**BYTE ( <name> -- )**

Define a byte wide structure member. See section on Amiga 'C' Structures.

---

**BYTE-SWAP ( n -- ns )**

Swaps the lower byte-pair of n.

```
HEX 1234 BYTE-SWAP . ( print 3412 )
```

Related Words: SWAP WORD-SWAP