! (n addr --) "store"

```
addr = even address
```

Store 32 bit value N at even address ADDR. This is the primary word for storing data in memory. Variables and other JForth data structures will always be word-aligned, ie. have an even address. Code which may conceivably store 32-bit values to odd addresses should use ODD! . The address is assumed to be a JForth relative address, not a 68000 absolute address.

```
\ create and set a variable to '234'
VARIABLE VAR1
234 VAR1 !
```

Related Words: >REL >ABS ODDD! ODD! ODDW! CMOVE FILL ODDW@ D! W! C! +! @ C@ W@ ..@ ..! TRAPS

!CSP (--) "store C S P"

Stores the current parameter stack address in the user variable CSP. Used with ?CSP to see if parameter stack is balanced and there were no compilation errors. Used to check stack depth between : and ;

Definition: : !CSP (--) SP@ CSP ! ;

Related Words: CSP ?CSP

" (<string> -- $addr) "quote"

Returns the address of quote delimited string. The string will be stored as a count byte followed by the text. Note that a space is required after the first quote. If encountered while COMPILING, quote compiles the string into the dictionary, returning that address when executed. If encountered while INTERPRETING, quote places the count at PAD, and the string at PAD+1, returning the PAD address. Be aware that the PAD is used by many Forth words so strings generated when INTERPRETING are considered temporary.

```
" Hello" COUNT TYPE
```

Related Words: EMIT COUNT TYPE 0"

 # (d1 -- d2)

```
"sharp" ( fig '83 )  "number" ( '79 )
```

Used to convert numbers to a character string. Divide D1 by current BASE, leaving remainder as D2. Convert quotient to ASCII character, place character at address in HLD (below PAD), decrement HLD.

```
: PHONE#  ( N -- , Print N as phone number)
    S->D <# # # # #  ASCII - HOLD  #S #>
    ( -- addr count ) TYPE SPACE ;
```

Related Words: #S <# #> HOLD PAD HLD COMMAS NO-COMMAS SIGN

#> (d1 -- text_addr count)

Drop D1, terminating numeric output conversion, leaving address and count of the string.

Standards: '79, '83, FIG.

```
: UD.  ( d1 -- , print an unsigned double number)
    <#  #S  #>  TYPE ;
```

#DIGITS (N -- #characters) "number digits"

Calculates how many characters are needed to display a number N, including commas if enabled.

Related Words: N>TEXT # NO-COMMAS COMMAS

#K (-- addr) "number k"

Variable used by SAVE-FORTH to determine the size of a JForth image file. To increase the size of your dictionary, set this variable, do a SAVE-FORTH, then run the resultant image.

```
\ Increase dictionary size by 20K
20 #K +!
SAVE-FORTH MyImage
```

Related Words: MAP SAVE-FORTH #U

#RELOCS (-- addr) "number relocs"

A user-variable, containing the number of long absolute relocations that have been compiled. Internal.

Related Words: PUSHRELOC .IMAGE MAP

#S (ud -- 0 0) "number s"

#S converts the unsigned double value on the stack into an ASCII character string in memory. String location starts one byte below PAD and works down. Produces only enough digits to represent the number. No leading zeros. Always produces at least one digit which may be zero. See # and #> .

Related Words: <# # #> SIGN HOLD HLD PAD #S

#TIB (-- addr) "number t i b"

A user-variable containing the TOTAL number of characters in the TIB (terminal input buffer).

```
: DUMP.TIB    TIB #TIB @   TYPE ;
```

Related Words: TIB 'TIB >IN QUERY

#U (-- addr) "number u"

#U is a variable that contains the number of allowable user variables. If you run out of user variables, increase #U and do a SAVE-FORTH. Works like #K .

Warning: once you make #U larger, you can't make it smaller. You must go to an earlier Forth to get smaller user area.

#VOCS (-- addr) "number vocs"

A user-variable, containing the current number of vocabularies defined in the dictionary.

JForth allows a maximum of 32 vocabularies, two of which are defined in the minimum development system. #VOCS helps you to keep track of what vocabularies are available and what words can be used.

Related Words: #CHARS

$ (<hex-number> -- N) "dollar"

$ is used to force the interpretation of the next word in the input stream as a hexadecimal number, regardless of the current BASE. $ is an IMMEDIATE word and, if found within a colon definition, will also compile the value into the dictionary as a LITERAL.

```
decimal 256   $ 80 -   .  ( prints 128 )
```

Related Words: HEX DECIMAL

$" string quote See "

$, string comma

(char --) (<text> --in--) (--inline-- <text>)

Takes text from the input stream, delimited by char, compiles this text into the next available dictionary location.

$, is a primitive used by the string literal words. It performs the actual job of installing the text into the dictionary.

$, is usually only called from higher-level words, such as " .

$- ($1 $2 -- flag) "string minus"

Compare two strings alphabetically, return:

```
flag = 0  if strings equal,
flag = +1 if $1 greater than $2,
flag = -1 if $2 greater than $1.
```

Related Words: $= COMPARE

$= ($1 $2 -- flag) "string equals"

$= performs a case-sensitive compare between 2 strings and returns true if equal, else false.

For case-INsensitive string matches, see MATCH? and TEXT=? .

Related Words: $- COMPARE MATCH? TEXT=?

$>0 ($string --) "string to zero"

Convert a normal Forth string, with a count byte, to a 'C' type NUL terminated string. The conversion is done in place by shifting the characters down by one and placing a NUL, zero byte, at the end. NUL terminated strings are used when passing strings to the Amiga libraries.

Related Words: 0COUNT >DOS

$APPEND (addr count dest-string --) "string append"

This word is used to append text to the string at destination-string address.

The count at address destination-string will be updated to reflect the increased string size.

It is the responsibility of the calling program to insure that sufficient space exists above address destination-string to accommodate the increased string size.

$DOLINES ($filename --)

This is equivalent to DOLINES except it takes a filename on the stack. See DOLINES

$DOS ($doscommand --) "string dos"

Execute a string as a DOS command.

```
: PWD ( -- , Print working directory. )
    " cd" $DOS ;
```

$FOPEN ($filename -- file-pointer) "string f open"

Opens the file whose name is on the stack. $filename is the address of a count byte. A zero is returned if the file could not be opened.

See FOPEN, 0FOPEN and chapter on File I/O.

```
: OPENTEMP  ( -- file-pointer )
    " ram:temp" $FOPEN DUP 0=
    WARNING" RAM:TEMP could not be opened!" ;
```

$LOGTO ($filename --)

Log all output to the file whose name is on the stack. See LOGTO

$MOVE ($addr-source addr-dest --) "string move"

$MOVE moves a complete string, including the count byte.

```
VARIABLE MYSTR 256 ALLOT
" A text string" MYSTR $MOVE   ( move string to MYSTR )
```

$SIZE ($addr -- true-size) "string size"

$SIZE takes a string count-byte address and returns the number of bytes in the string plus the count byte plus any padding required to word-align the end. Use COUNT if you want the actual number of characters in the string, without count-byte and padding.

$TYPE ($addr --) "string type"

$TYPE is a shorthand for COUNT TYPE. It takes a string count-byte address and types out that string.

' (<word> -- cfa) "tick"

All words in the dictionary have a body containing executable 68000 code. This word returns that address for any entry in the CONTEXT vocabulary.

If executed within a colon definition. this IMMEDIATE word will locate the cfa of the next word as usual, then compile it as an address literal (ALITERAL) to be pushed to the stack at run-time.

```
\ Use ' for indirect execution.
: FOO ." Hello world!" CR ;
' FOO   EXECUTE   ( does FOO )
```

'>BODY (cfa -- pfa) "tick to body"

See >BODY

'>NAME (cfa -- nfa) "tick to name"

See >NAME

'TIB (-- addr) "tick T I B"

This is a user-variable containing the JForth relative address of that users TIB area. The value of this user-variable is placed on the stack by TIB .

'WORD (-- $addr) "tick word"

'WORD is a DEFERred word that returns the address that WORD will use to PLACE its output.

'WORD usually is set to execute HERE.

((<text> --) "left parenthesis"

Text between parentheses is considered to be a comment in Forth. The word (will eat text until the closing) .

The opening (must be followed by a space. The comment is terminated with the next) character, or

end-of-line. Note that a comment may NOT span more than one line.

```
( for humans only , put anything you want inside )
```

Related Words: \ .IF

($") "parenthesis string quote"

Compile time: (--) (<string> --in--) (--inline-- <string>)

Run time: (-- $addr)

($") is used in inline string words like " $" etc... ($") is compiled before using $, to place a string inline. At run time, ($") will place the inline address of the string on the stack, and skip over the inline string to continue executing the code after it.

```
: $\ COMPILE ($") ASCII \ $,  ; IMMEDIATE
: FOO $\ A string with " in it!\ $TYPE ;
```

Related Words: $" 0" (($")) "

(($")) "double parenthesis string quote"

(-- $addr) ($addr ip --R-- ip)

(($")) is used in inline string words like (?ABORT") . (($")) is used within words that must parse inline strings. It is almost the same as ($") but it is used at 1 level of calling lower. See the STRINGS and STRING+ files for examples.

```
: (?ABORT")  (($"))  SWAP
    IF   CR $TYPE QUIT
    THEN  DROP ;
: ABORT"  COMPILE (?ABORT") ASCII "   $,  ; IMMEDIATE
```

((CREATE)) (--) "paren paren create"

This word creates a dictionary header out of a valid JForth string residing at HERE. The dictionary pointer DP is left pointing to the code-field-address cfa , with the name-field-address being left SMUDGEd.

((CREATE)) is a primitive used by all defining words .

(+LOOP) "paren plus loop"

(inc --) (n1 n2 --LOOP-STACK-- n1 n2+inc)

```
n2 = index approaching overflow
n1 = correction value to recreate index for I
inc = increment amount
```

(+LOOP) is the run time action compiled by +LOOP . Used with DO .

(+LOOP) adds inc to n2. If n2 overflows, the LOOP is satisfied, and execution continues once the loop indices have been dropped from the LOOP-STACK. If n2 does NOT overflow, return to the address immediately following the corresponding DO code.

Related Words: LOOP DO -LOOP +LOOP -DO LOOP-BACK LOOP-FORWARD REPEAT DO_FLAG

(-LOOP) "paren minus loop"

(inc --) (n1 n2 --LOOP STACK-- n1 n2-inc)

```
n2  = index approaching overflow
n1  = correction value to recreate index for I
```

```
inc = increment amount
```

(-LOOP) is compiled by -LOOP . Used with DO .

(-LOOP) subtracts inc from n2. If n2 overflows, the LOOP is satisfied, and execution continues once the loop indices have been dropped from the LOOP-STACK. If n2 does NOT overflow, return to the address immediately following the corresponding DO code.

```
Related Words: LOOP DO -LOOP +LOOP -DO LOOP-BACK  LOOP-FORWARD
REPEAT DO_FLAG
```

(.") "paren dot quote"

(--) ($addr --R-- addr)

```
$addr = string address
addr  =  address of next word to be interpreted
```

This word is a run time string-handler, used to print the string immediately following it's own compiled location. It also modifies the program counter to jump around the string. It is compiled by ." to output the string when executed.

(.) (n --) "paren dot"

The number primitive executed for signed integer output in the current BASE, usually executed by . (dot).

Definition: : (.) (n --) S->D D. ;

(;) (--) "paren semi-colon"

This word is a primitive used in closing any colon definition. It does the following.

1. Verify stack integrity (via ?CSP).
2. Compiles the RTS opcode (necessary at the end of all JForth definitions).
3 Reveals the name-field to FIND, so that it may accessed.
4. Terminates COMPILE mode, returning to INTERPRET mode.

(?DO) (limit index --) "paren question do"

(?DO) is compiled by DO .

```
At run time, (?DO) check to see if the difference between the
index and limit is a positive, non-zero value.
- If so, it converts the index and limit to LOOP parameters,
  and pushes them on the LOOP-STACK.
- IF not, it drops both parameters, and jumps past the
  corresponding LOOP word.
```

(?LEAVE) (flag --) "paren question leave"

(?LEAVE) is compiled by ?LEAVE . Used for conditionally leaving a DO LOOP.

```
(?LEAVE), at run time, will check a flag on the stack...
- If non-zero, it will drop the loop indices and jump past
  the corresponding LOOP word.
- If zero, operation continues.
```

(?TERMINAL) (-- true-if-key-flag) "paren question terminal"

(?TERMINAL) is the run time code for ?TERMINAL . ?TERMINAL is a deferred word normally set to (?TERMINAL) . In the '83 standard this is referred to as (?KEY) .

(ABORT) (--) "paren abort"

This is the default word which the DEFERred word ABORT executes. It calls QUIT.

(CFA,) (cfa --) "paren c f a comma"

Compile the appropriate code necessary to execute the word whose CFA is on the stack.

Once the compiler has decided to compile a reference to an existing word, this is the primitive which figures out the particular manner of compilation which will be used, ie. INLINE, BSR, short or long JSR.

This is the default contents of the DEFERred word CFA, and is the real workhorse of the compile process.

(CFA,) determines whether the referenced CFA is of the INLINE type. If so, a copy of the body of that definition is moved to the current DP, and its length is ALLOTed from the dictionary.

If (CFA,) determines the word is of the BOTH variety, it checks the value of MAX-INLINE, and if the length of the word is less, it is compiled inline; otherwise referenced via a call, as shown.

If the word must be referenced via a call, the compiler will select the most efficient of 3 types of subroutine calls to use.

If the called word is in the lowest 96K of the dictionary, the compiler will use the JUMP-SUBROUTINE (JSR), INDEXED WITH DISPLACEMENT mode, either through register A4 (ORG) or A3 (+64K).

If the called word is not in the lower 96K of JForth, but is within 32K of the calling instruction, the BRANCH-SUBROUTINE (BSR) mode is used.

Otherwise, the compiler will use the JUMP-SUBROUTINE (JSR), LONG ABSOLUTE mode. This type of reference requires the JForth System Manager to maintain a table of all long absolute references, that they may be relocated by the Amiga Loader at startup. This table is invisible to the programmer.

(COMMAS) (-- addr) "paren commas"

This is the user variable that the number-formatting and printing routines check to see if they should include commas in the output stream. If true, include commas.

Related Words: COMMAS NO-COMMAS

(CR) (--) "paren carriage return"

(CR) is the default contents of the DEFERed word CR . (CR) outputs a carriage return and a line feed to the standard EMIT device.

(CREATE) (<name> --) "paren create"

(CREATE) is the default contents for the DEFERed word :CREATE .

(CREATE) eats the next word in the input stream (parsed by the character value stored in CREATECHAR), and from it, creates a new dictionary header, in the CURRENT vocabulary. The name is left smudged.

Related Words: :CREATE REDEF?

(DO) (limit index --) "paren do"

(DO) performs similarly to (?DO), except that it performs no index/limit value checking. This means that: 1. If the index and limit are equal, the LOOP will be executed once. 2. If the index is greater than the limit, it will LOOP until the index has been incremented and is subsequently equal to the limit.

NOTE: this word, when used with LOOP, may be used for positive-growing LOOPS:

(EMIT) (char --) "paren emit"

This is an EMIT function primitive that performs single-character I/O to the CONSOLE window. It is executed by the deferred word EMIT when in the SLOW I/O mode.

This word transfers characters to the screen in single-character fashion. The normal I/O technique utilized in JForth is buffered, and this word is only used, oddly enough, for KEY, to echo each character typed to the user. In the SLOW I/O mode, it is the only EMIT primitive used.

(EMIT) will update OUT to reflect the current cursor column number.

NOTE: (EMIT) is NOT the default contents for the DEFERed word EMIT. See <FASTEMIT>.

(EXPECT) (addr nchars --) "paren expect"

Default word for EXPECT . See EXPECT . EXPECT is deferred so that it can be changed by the user if needed.

(FIND) "paren find"

($name lfa -- cfa true | $name false)

(FIND) is a primitive used to locate the most recently defined occurrence of the given name. It will start searching the dictionary at the lfa passed on the stack.

If (FIND) locates the name, it returns its CFA and a TRUE flag.

If the word is not found, (FIND) will return the original name and a FALSE flag.

(FIRST) (-- addr) "paren first"

This is a user-variable that is only used in the BLOCK environment, and is initialized when the source file JU:BLOCK is compiled.

After initialization, it contains the address of the first virtual buffer available under BLOCK.

(ID.) (nfa --) "paren i d dot"

This is the default contents of the DEFERred word ID. . It accepts the count byte address of a name-field, and TYPEs the name on the standard EMIT device.

(INTERPRET) (--) "paren interpret"

Run time code for INTERPRET . This is the default contents of the DEFERed word INTERPRET. It is the core of what is known in Forth as the "Outer Interpreter". JForth does not have an "Inner Interpreter since it compiles directly to 68000 machine code. (INTERPRET) is what interprets and executes the commands that you enter at the keyboard. It is also used when compiling a file.

(INTERPRET) gets its input from the TIB. (INTERPRET) uses >IN to index into the TIB. When >IN equals #TIB, or an error occurs, (INTERPRET) returns.

After (INTERPRET) has parsed a word from the input stream, it searches first the CONTEXT, then CURRENT vocabularies. If found, the resultant address will either be executed or compiled, based on the STATE of the system, and (INTERPRET) will continue with the next word.

If not found, it will see if the text can be successfully converted to a number (using the current BASE) and if so, the resultant number is passed to LITERAL.

If a number cannot be derived, the system QUITS, displaying the offending text on the console.

`Related Words: QUERY $INTERPRET`

(KEY) (-- char) "paren key"

Wait for a character from the console. When received, return the character without echo. Flushes any pending output. Default contents of DEFERred word KEY .

(LEAVE) "paren leave"

(--) (n1 n2 --LOOP-STACK--) n1 and n2 are loop indices

(LEAVE) is compiled by LEAVE.

(LEAVE) forces immediate termination of a DO LOOP by dropping the loop indices, and jumping past the corresponding LOOP word.

(LIMIT) (-- addr) "bracket limit"

This is a user-variable that is only used in the BLOCK environment, and is initialized when the source file JU:BLOCK is compiled.

After initialization, it contains the address of the END of the virtual buffer area available under BLOCK.

(LONGCFA,) (cfa --) "paren long c f a comma"

This is the default contents for the DEFERred word LONGCFA, .

This word is used to compile a LONG ABSOLUTE JUMP-SUBROUTINE to a cfa in the dictionary. (LONGCFA,) will automatically create an entry in the Long Relocations table for that reference.

(LOOP) paren loop

(--) (n1 n2 --LOOP-STACK-- n1 n2+1)

```
n2 = index approaching overflow
n1 = correction value to recreate index for I
```

(LOOP) is compiled by LOOP.

(LOOP) increments n2 by 1. If n2 overflows, the LOOP is satisfied, and execution continues once the loop indices have been dropped from the LOOP-STACK. If n2 does not overflow, return to the address immediately following the corresponding DO code.

(NUMBER) ($string -- d true | false) "paren number"

Default contents of deferred word NUMBER . This converts the character string at $string to a double number using the current BASE. If the conversion fails, false is returned.

(OF) "parenthesis of"

(case-value of-value -- case-value)

(OF) is the internal run time function for OF used in CASE statements. It compares the top item on the stack with a duplicate of the next thing on the stack. If they are equal it does not branch, else it branches to the ELSE part of the OF ... ELSE structure.

(QUIT) (--) "paren quit"

(QUIT) is used to terminate the currently running program and initialize the INTERPRETer to return to the CONSOLE. When called, it has the following specific effects:

```
1. Closes all files that had MARKFCLOSE executed on them.
2. Frees all memory blocks that had MARKFREEBLOCK executed on
them.
3. Initializes the TIB from TIB0, #TIMES variable to 1.
4. ALIGNS the dictionary.
```

```
5. Terminates COMPILE MODE, forcing INTERPRET MODE.
6. Clears the USP stack.
7. Enters the QUERY INTERPRET loop.
```

This is normally executed by QUIT and performs many JForth-System initializations. If you desire to change the operation of QUIT, you should only supplement (QUIT), executing your code then calling (QUIT) . When the JForth Kernal is TURNKEYed, the action of (QUIT) MUST BE REPLACED with the startup word for your turnkey operation.

(QUIT) is the default contents of the DEFERed execution word, QUIT, and may remain so in any version of your application that has not been distributed or released in any way. You are required to replace this cfa with one of your own definition in a TURNKEYed application. See TURNKEY and CLONE.

(SOURCE) (-- TIB #TIB) "paren source"

```
TIB  = Terminal Input Buffer address
#TIB = Number of characters in TIB
```

This primitive returns the address and length of the TIB.

*** (n1 n2 -- n1*n2) "times"**

Multiply two 32 bit numbers.

```
\ Multiply 23 time 7
23 7 * .  ( prints 161 )
```

***/ (n multiplier divisor -- n*m/d) "times slash"**

*/ calculates n times multiplier as a double length value. This is useful if the product would be greater than 32 bits.

```
1,000,000,000 234 567 */  ( correct )
1,000,000,000 234 * 567 / ( overflows! )
```

***/MOD (n1 n2 n3 -- n4 n5) "times slash mod"**

```
n5 = n1 times n2 and then divided by n3
n4 = remainder
```

*/MOD carries n1 times n2 as a double length value. This allows greater accuracy than would a single length intermediate product. Then n3 is divided into n1 * n2 yielding n5 quotient and n4 remainder.

```
2 5 7 */MOD  ( yields 3 1 )
```

+ (n2 n1 -- n1+n2) "plus"

+ adds the top two values on the stack and places the result on the stack.

```
3 4 + .   ( prints 7
```

+! (n addr --) "plus store"

+! adds the value n to the value stored at address addr and leaves the result at addr .

```
VARIABLE VAR1
10 VAR1 !
5 VAR1 +!
VAR1  ?   ( prints 15 )
```

+- "plus minus"

```
( n1  n2  --  n1 )    if n2 > 0 or n2 = 0
( n1  n2  -- -n1 )    if n2 < 0
```

+- negates the sign of second stack value n1 if top of stack value n2 is negative. Top value n2 is then dropped.

```
5 -7 +- .   ( prints -5 )
```

+DOS (addr count --) "plus dos"

Adds a string to the existing null-terminated string at DOS0 . This is used when interfacing with Amiga DOS.

```
" df0:c/" COUNT >DOS
" dir" COUNT +DOS
DOS0 0COUNT TYPE  ( print "df0:c/dir" )
```

+LOOP "plus loop"

Compile time: (loop-addr do-flag --)

Run time: (inc-val --) (index limit --R-- index limit |)

Used in DO LOOPs where you want to increase by some number other than 1.

+LOOP checks for a do-flag at compile time and gives error if not found. +LOOP compiles (+LOOP) At run time (+LOOP) increments index by inc-val and jumps back to loop body until index crosses boundary between limit and limit-1 (overflows).

```
: TDO+ 50 0 DO I . 10 +LOOP ;
TDO+   ( print 0 10 20 30 40 )
```

+SHIFT (a b -- a<<b) "plus shift"

+SHIFT is a faster smaller routine than SHIFT that does only left shifts toward the MSB. 0's are shifted into the LSB positions.

```
3 2 +SHIFT .  ( prints 12 )
```

Related Words: SHIFT ASHIFT -SHIFT U2*

+STACK (n var-addr --) "plus stack"

+STACK is used to PUSH items to a memory block (allocated via ALLOCBLOCK) being used as a stack. It accepts as arguments, the value n1 to push and the address var-addr of a VARIABLE or USER-variable being used to hold the address of the allocated memory area.

If the VARIABLE or USER-variable contains 0, +STACK will allocate a 1024-byte block to be used, sufficient for 256 cells, and place the address of the block in the storage location. It will then push the value to the area.

If the VARIABLE or USER-variable contains other than zero, it is assumed by +STACK to be the address of an existing memory block.

If the allocated stack area is full, and cannot accept the new element, +STACK will attempt to increase the allocated size in 1K increments.

Please see the chapter on Memory Management.

Related Words: PUSH POP -STACK