***pijFORTHos* FORTH Reference**

Each dialect of FORTH has its own definitions. Development in FORTH involves extending the vocabulary with words specific to your application. Most pre-defined words in [*pijFORTHos*](https://github.com/Avoncliff/pijFORTHos/blob/master/README.md) follow tradtional standards and conventions, but see the tables below for details.

**Built-in Definitions**

These definitions are present in [*pijFORTHos*](https://github.com/Avoncliff/pijFORTHos/blob/master/README.md) on initial [boot](https://github.com/Avoncliff/pijFORTHos/blob/master/doc/bootload.md).

**Built-in FORTH Variables**

Variables are words that place an address on the stack. Use @ (fetch) and ! (store) to read/write the current value of a variable. The following variables are pre-defined in *pijFORTHos*

| **Variable** | **Description** |
| --- | --- |
| STATE | Is the interpreter executing code (0), or compiling a word (non-zero)? |
| HERE | Address of the next free byte of memory. When compiling, compiled words go here. |
| LATEST | Address of the latest (most recently defined) word in the dictionary. |
| S0 | Address of the top of the parameter/data stack. |
| BASE | The current base for printing and reading numbers. (initially 10). |

Here is an example using the BASE variable:

BASE @ \ read the current value of BASE (to restore later)

16 BASE ! \ switch to hexadecimal

8000 100 DUMP \ hexdump 256 bytes starting at 0x8000

BASE ! \ restore the orignial value of BASE (from the stack)

**Built-in FORTH Constants**

Constants are words that place pre-defined value on the stack. They are useful mnemonics and help us avoid "magic" numbers in our code. The following constants are pre-defined in *pijFORTHos*

| **Variable** | **Description** |
| --- | --- |
| VERSION | The version number of this FORTH. |
| R0 | Address of the top of the return stack. |
| DOCOL | Address of DOCOL (the word interpreter). |
| PAD | Address of the 128-byte scratch-pad buffer (top of HERE memory). |
| F\_IMMED | The IMMEDIATE flag's actual value. |
| F\_HIDDEN | The HIDDEN flag's actual value. |
| F\_LENMASK | The length mask in the flags/len byte. |
| FALSE | Boolean predicate False (0) |
| TRUE | Boolean predicate True (-1), anything != 0 is TRUE |

Given the relationship between HERE and PAD, the following calculates the number of free memory cells available:

PAD \ the base of PAD is the end of available program memory

HERE @ - \ subtract the base address of free memory

4/ \ divide by 4 to convert bytes to (32-bit) cells

**Built-in FORTH Words**

There are many words pre-defined in *pijFORTHos*. They are presented here by category, so you can see related words together.

**Stack Manipulation**

| **Word** | **Stack** | **Description** |
| --- | --- | --- |
| DROP | ( a -- ) | drop the top element of the stack |
| DUP | ( a -- a a ) | duplicate the top element |
| SWAP | ( a b -- b a ) | swap the top two elements |
| OVER | ( a b -- a b a ) | push copy of second element on top |
| ROT | ( a b c -- b c a ) | stack rotation |
| -ROT | ( a b c -- c a b ) | backwards rotation |
| NIP | ( a b -- b ) | drop the second element of the stack |
| TUCK | ( a b -- b a b ) | push copy of top element below second |
| PICK | ( a\_n ... a\_0 n  -- a\_n ... a\_0 a\_n ) | copy n-th stack item |
| 2DROP | ( a b -- ) | drop the top two stack elements |
| 2DUP | ( a b -- a b a b ) | duplicate top two stack elements |
| 2SWAP | ( a b c d -- c d a b ) | swap top two pairs of stack elements |
| 2OVER | ( a b c d -- a b c d a b ) | copy second pair of stack elements |
| ?DUP | ( 0 -- 0 | a -- a a ) | duplicate if non-zero |
| DEPTH | ( -- n ) | the number of items on the stack |
| >R | (S: a -- ) (R: -- a ) | move the top element from the data stack to the return stack |
| R> | (S: -- a ) (R: a -- ) | move the top element from the return stack to the data stack |
| RDROP | (R: a -- ) | drop the top element from the return stack |
| RSP@ | ( -- addr ) | get return stack pointer |
| RSP! | ( addr -- ) | set return stack pointer |
| DSP@ | ( -- addr ) | get data stack pointer |
| DSP! | ( addr -- ) | set data stack pointer |

**Arithmetic Operations**

| **Word** | **Stack** | **Description** |
| --- | --- | --- |
| NEGATE | ( n -- -n ) | negation |
| + | ( n m -- n+m ) | addition |
| - | ( n m -- n-m ) | subtraction |
| \* | ( n m -- n\*m ) | multiplication |
| / | ( n m -- n/m ) | division |
| MOD | ( n m -- n%m ) | modulus |
| /MOD | ( n m -- r q ) | where n = q \* m + r |
| S/MOD | ( n m -- r q ) | alternative signed /MOD using Euclidean division |
| 1+ | ( n -- n+1 ) | increment |
| 1- | ( n -- n-1 ) | decrement |
| 2+ | ( n -- n+2 ) | increment by 2 |
| 2- | ( n -- n-2 ) | decrement by 2 |
| 4+ | ( n -- n+4 ) | increment by 4 |
| 4- | ( n -- n-4 ) | decrement by 4 |
| 2\* | ( n -- n\*2 ) | double |
| 2/ | ( n -- n/2 ) | halve (arithmetic shift right) |
| 4\* | ( n -- n\*4 ) | quadruple |
| 4/ | ( n -- n/4 ) | quarter |

**Logical and Bitwise Operations**

| **Word** | **Stack** | **Description** |
| --- | --- | --- |
| = | ( n m -- p ) | where p is TRUE when (n == m), FALSE otherwise |
| <> | ( n m -- p ) | where p = (n != m) |
| < | ( n m -- p ) | where p = (n < m) |
| > | ( n m -- p ) | where p = (n > m) |
| <= | ( n m -- p ) | where p = (n <= m) |
| >= | ( n m -- p ) | where p = (n >= m) |
| NOT | ( p -- !p ) | Boolean predicate not |
| 0= | ( n -- p ) | where p = (n == 0) |
| 0<> | ( n -- p ) | where p = (n != 0) |
| 0< | ( n -- p ) | where p = (n < 0) |
| 0> | ( n -- p ) | where p = (n > 0) |
| 0<= | ( n -- p ) | where p = (n <= 0) |
| 0>= | ( n -- p ) | where p = (n >= 0) |
| INVERT | ( a -- ~a ) | bitwise not |
| AND | ( a b -- a&b ) | bitwise and |
| OR | ( a b -- a|b ) | bitwise or |
| XOR | ( a b -- a^b ) | bitwise xor |
| LSHIFT | ( a n -- a<<n ) | logical shift left |
| RSHIFT | ( a n -- a>>n ) | logical shift right |

**Memory Access**

| **Word** | **Stack** | **Description** |
| --- | --- | --- |
| @ | ( addr -- value ) | read (fetch) value from addr |
| ! | ( value addr -- ) | write (store) value at addr |
| +! | ( amount addr -- ) | add amount to value at addr |
| -! | ( amount addr -- ) | subtract amount from value at addr |
| C! | ( c addr -- ) | write byte c at addr |
| C@ | ( addr -- c ) | read byte from addr |
| CMOVE | ( src dst len -- ) | copy len bytes from src to dst |
| COUNT | ( addr -- addr+1 c ) | extract first byte (len) of counted string |
| CHAR word | ( -- c ) | ASCII code of first character in word |

**Input and Output**

| **Word** | **Stack** | **Description** |
| --- | --- | --- |
| KEY | ( -- c ) | read a character from input |
| EMIT | ( c -- ) | write character c to output |
| CR | ( -- ) | print newline |
| SPACE | ( -- ) | print space |
| WORD | ( -- addr len ) | read next word from input |
| NUMBER | ( addr len -- n e ) | convert string to number n, with e unparsed characters |
| TELL | ( addr len -- ) | write a string to output |
| . | ( n -- ) | print signed number and a trailing space |
| U. | ( u -- ) | print unsigned number and a trailing space |
| .R | ( n width -- ) | print signed number, padded to width |
| U.R | ( u width -- ) | print unsigned number, padded to width |
| ? | ( addr -- ) | fetch and print signed number at addr |
| .S | ( -- ) | print the contents of the stack (non-destructive) |
| DECIMAL | ( -- ) | set number conversion BASE to 10 |
| HEX | ( -- ) | set number conversion BASE to 16 |
| 10# value | ( -- n ) | interpret decimal literal value w/o changing BASE |
| 16# value | ( -- n ) | interpret hexadecimal literal value w/o changing BASE |
| DUMP | ( addr len -- ) | pretty-printed memory dump |

**Definition and Compilation**

| **Word** | **Stack** | **Description** |
| --- | --- | --- |
| LIT word | ( -- ) | compile literal in FORTH word |
| LITS addr len | ( -- ) | compile literal string in FORTH word |
| FIND | ( addr len -- entry | 0 ) | search dictionary for entry matching string |
| >CFA | ( entry -- xt ) | get code field address from dictionary entry |
| >DFA | ( entry -- addr ) | get data field address from dictionary entry |
| CREATE | ( addr len -- ) | create a new dictionary entry |
| , | ( n -- ) | write the top element from the stack at HERE |
| [ | ( -- ) | change interpreter state to Immediate mode |
| ] | ( -- ) | change interpreter state to Compilation mode |
| : name | ( -- ) | define (compile) a new FORTH word |
| ; | ( -- ) | end FORTH word definition |
| IMMEDIATE | ( -- ) | set IMMEDIATE flag of last defined word |
| HIDDEN | ( entry -- ) | toggle HIDDEN flag of a word |
| HIDE word | ( -- ) | hide definition of following word |
| ' word | ( -- xt ) | find CFA of following word (compile only) |
| [COMPILE] word | ( -- ) | compile otherwise IMMEDIATE word |
| RECURSE | ( -- ) | compile recursive call to current word |
| LITERAL | (C: value --) (S: -- value) | compile LIT value |
| CONSTANT name | ( value -- ) | create named constant value |
| ALLOT | ( n -- addr ) | allocate n bytes of user memory |
| CELLS | ( n -- m ) | number of bytes for n cells |
| VARIABLE name | ( -- addr ) | create named variable location |

**Control Structures**

| **Word** | **Stack** | **Description** |
| --- | --- | --- |
| EXIT | ( -- ) | restore FIP and return to caller |
| BRANCH offset | ( -- ) | change FIP by following offset |
| 0BRANCH offset | ( p -- ) | branch if the top of the stack is zero |
| IF true-part THEN | ( p -- ) | conditional execution |
| IF true-part ELSE false-part THEN | ( p -- ) | conditional execution |
| UNLESS false-part ... | ( p -- ) | same as NOT IF |
| BEGIN loop-part p UNTIL | ( -- ) | post-test loop |
| BEGIN loop-part AGAIN | ( -- ) | infinite loop (until EXIT) |
| BEGIN p WHILE loop-part REPEAT | ( -- ) | pre-test loop |
| CASE cases... default ENDCASE | ( selector -- ) | select case based on selector value |
| value OF case-body ENDOF | ( -- ) | execute case-body if (selector == value) |

**System Operations**

| **Word** | **Stack** | **Description** |
| --- | --- | --- |
| QUIT | ( -- ) | clear return and data stacks, restart interpreter loop |
| UPLOAD | ( -- addr len ) | XMODEM file upload to memory image |
| BOOT | ( addr len -- ) | boot from memory image (see UPLOAD) |
| MONITOR | ( -- ) | enter bootstrap monitor |
| EXECUTE | ( xt -- ) | call procedure indicated by CFA |

**Additional Definitions in FORTH**

Many standard words can be defined using the built-in primitives shown above. The file jonesforth.f contains additional important and useful definitions. The entire contents of this file can simply be copy-and-pasted into the terminal session connected to the [*pijFORTHos*](https://github.com/Avoncliff/pijFORTHos/blob/master/README.md) console. Code at the end of the file displays a welcome message when processing is complete.

**Additional Constants Defined in FORTH**

The following constants are defined in jonesforth.f

| **Constant** | **Description** |
| --- | --- |
| '\n' | newline character (10) |
| BL | blank character (32) |
| ':' | colon character (58) |
| ';' | semicolon character (59) |
| '(' | left parenthesis character (40) |
| ')' | right parenthesis character (41) |
| '"' | double-quote character (34) |
| 'A' | capital A character (65) |
| '0' | digit zero character (48) |
| '-' | hyphen/minus character (45) |
| '.' | period character (46) |

**Additional Words Defined in FORTH**

The following words are defined in jonesforth.f

| **Word** | **Stack** | **Description** |
| --- | --- | --- |
| ( comment text ) | ( -- ) | comment inside definition |
| SPACES | ( n -- ) | print n spaces |
| WITHIN | ( a b c -- p ) | where p = ((a >= b) && (a < c)) |
| ALIGNED | ( addr -- addr' ) | round addr up to next 4-byte boundary |
| ALIGN | ( -- ) | align the HERE pointer |
| C, | ( c -- ) | write a byte from the stack at HERE |
| S" string" | ( -- addr len ) | create a string value |
| ." string" | ( -- ) | print string |
| VALUE name | ( n -- ) | create named value initialized to n |
| TO name | ( n -- ) | set named value to n |
| +TO name | ( d -- ) | add d to named value |
| DICT word | ( -- 0 | entry ) | dictionary entry for word, 0 if not found |
| ID. | ( entry -- ) | print word/name associated with dictionary entry |
| ?HIDDEN | ( entry -- p ) | get HIDDEN flag from dictionary entry |
| ?IMMEDIATE | ( entry -- p ) | get IMMEDIATE flag from dictionary entry |
| WORDS | ( -- ) | print all the words defined in the dictionary |
| FORGET name | ( -- ) | reset dictionary prior to definition of name |
| CFA> | ( xt -- 0 | entry ) | CFA> is the opposite of >CFA |
| SEE word | ( -- ) | print source code for word |
| :NONAME | ( -- xt ) | define (compile) an unnamed new FORTH word |
| ['] name | ( -- xt ) | compile LIT |
| CATCH | ( xt -- 0 | n ) | execute procedure reporting n THROW or 0 |
| THROW | ( n -- ) | send exception n to CATCH |
| ABORT | ( -- ) | THROW exception -1 |
| BINARY | ( -- ) | set number conversion BASE to 2 |
| OCTAL | ( -- ) | set number conversion BASE to 8 |
| 2# value | ( -- n ) | interpret binary literal value w/o changing BASE |
| 8# value | ( -- n ) | interpret hexadecimal literal value w/o changing BASE |
| # value | ( b -- n ) | interpret base-b literal value w/o changing BASE |
| PRINT-STACK-TRACE | ( -- ) | walk up return stack printing values |
| UNUSED | ( -- n ) | calculate number of cells remaining in user memory |