CoreWords

Documentation for the core libraries

Updated Dec 20, 2007 by cbiffle

References to the ANS Forth standard below (e.g. "ANS 6.1.0010") are to the *draft* of the standard. The final standard is expensive and not easy to obtain. The behavior of all ANS-defined words is believed to be similar to the final ANS standard, but may vary; please report any such bugs.

Note: this page is not yet complete.

Conventions

This page uses Forth conventions for documenting words and their behavior, with some extensions.

Stack Comments

Stack comments show a "picture" of the contents of the stack before and after a word executes -- before on the left, after on the right, with a dash in between. For example, a comment that reads

```
( a b c -- d e )
```

indicates that a particular word will consume three values from the stack (`a`, `b`, and `c`) and produce two values, `d` and `e`.

The types of the values are traditionally indicated by their names:

- `n` is a signed integer.
- `u` is an unsigned integer.
- `x` is any cell-sized value (integer, pointer, flag), and a sign that the word doesn't care about the specific contents.
- `d` is a *double-cell* integer -- in PropellerForth, this means a 64-bit integer. This may appear as `du` or `dn` to indicate its signed-ness.
- `addr` is a cell-aligned (32-bit-aligned) shared-memory address.
- `haddr` is a halfcell-aligned shared-memory address. (PropellerForth-specific.)
- `caddr` is a byte- (character-) aligned shared-memory address.
- `laddr`is an address in Cog-local RAM. (PropellerForth-specific.)
- `f` is a boolean flag, either `true` or `false`. (Forth uses -1 and 0 to represent `true` and `false`. The `f` type is PropellerForth-specific; ANS Forth just uses `x`.)
- `xt` is an *execution token* -- a pointer to another Forth word that may be executed or manipulated.

Words that consume or produce multiple values assign each one a number to tell them apart: `x2`, `n5`, etc.

For words defined specially by PropellerForth, you may see more descriptive names for values in stack comments. If the type of the values is not immediately obvious, the text below the stack comment should explain it.

Standards and History

Each word indicates what standard defines its behavior (if any) and when it became available in PropellerForth:

- The **Defined by:** field references the formal definition of the word's behavior -- currently either the ANS Forth draft standard, or PropellerForth itself.
- The **Since:** field names the earliest version of PropellerForth that included this word by default.

Core Words

!

(x addr --)

Stores `x` into shared RAM at `addr`. `addr` must be cell-aligned. Consumes both.

Since: 20061114

Defined by: ANS 6.1.0010

@

(addr -- x)

Loads 32 bits of shared RAM at `addr`. `addr` must be cell-aligned.

Since: 20061114

Defined by: ANS 6.1.0650

+

(x1 x2 -- x3)

Adds `x1` to `x2`, giving `x3`.

Since: 20061114

Defined by: ANS 6.1.0120

-

(x1 x2 -- x3)

Subtracts `x2` from `x1`, giving `x3`.

Since: 20061114

Defined by: ANS 6.1.0160

and

(x1 x2 -- x3)

Bitwise-ANDs `x1` and `x2`, giving `x3`.

Since: 20061114

Defined by: ANS 6.1.0720

C!

(c caddr --)

Stores the low eight bits of `c` in shared memory at `caddr`. Consumes both.

Since: 20061114

Defined by: 6.1.0850

C@

(caddr -- c)

Loads the byte at `caddr` in shared memory, *without sign-extending.*

Since: 20061114

Defined by: 6.1.0870

dup

(x -- x x)

Pushes a copy of the top cell on the stack.

Since: 20061114

Defined by: ANS 6.1.1290

drop

(x --)

Pops and discards the top cell on the stack.

Since: 20061114

Defined by: ANS 6.1.1260

(EMIT)

(c --)

Writes the low 8 bits of `c` to the default serial port.

`(EMIT)` is the word used to generate output on startup, and is not terribly well-optimized or general-purpose. Users should probably not bother with it.

Since: 20061114

Defined by: PropellerForth

"equals"

(x1 x2 -- f)

Note: the name of this word is =. This wiki does not allow = to be used as a header, which is why the header above reads *"equals"*.

Checks whether `x1` and `x2` are equal; pushes `true` if so, `false` otherwise.

Since: 20061114

Defined by: ANS 6.1.0530

execute

(...args... xt -- ...result...)

Pops the execution token `xt` from the stack and executes it, allowing it to consume arguments and produce a result.

An execution token is simply the address of a word's entry in the dictionary. The ' ("tick") word loads the execution token of other words.

Since: 20061114

Defined by: ANS 6.1.1370

exit

(--)

Returns from a colon definition. `exit` is automatically compiled by `;` at the end of a definition, but can be called by name to return early.

Since: 20061114

Defined by: ANS 6.1.1380

>

(n1 n2 -- f)

Checks whether `n1` is greater than `n2`; pushes `true` if so, `false` otherwise.

Since: 20061114

Defined by: ANS 6.1.0540

H!

(h haddr --)

Stores the low 16 bits (halfcell) of `h` in shared memory at `haddr`. `haddr` must be halfcell-aligned.

Since: 20061114

Defined by: PropellerForth

H@

(haddr -- h)

Loads 16 bits from shared memory at `haddr` and pushes it on the stack, *without sign extending.* `haddr` must be halfcell-aligned.

Since: 20061114

Defined by: PropellerForth

hubop

(parameter operation -- result)

Performs an operation through the Propeller's shared Hub. `operation` selects the operation to perform, and `parameter` is an operation-specific parameter. All hub operations return a result, though some aren't interesting.

For specifics on the hub operations, see the Propeller specification for the `HUBOP` instruction.

Since: 20061114

Defined by: PropellerForth

(KEY)

(-- c)

Reads an 8-bit character from the default serial port, blocking if needed.

`(KEY)` is the word used to read input on startup, and is not terribly well-optimized or general-purpose. Users should probably not bother with it.

Since: 20061114

Defined by: PropellerForth

literal

In compiled definition: *(-- x)*

Pushes the 32-bit constant following it in the instruction stream onto the stack. The constant must be 32-bit aligned.

In both interpreted mode and compiled definitions, it's usually best to just enter numbers without directly calling `literal`. In the interpreter in particular, `literal` won't work.

Since: 20061114

Defined by: ANS 6.1.1780

lit16

In compiled definition: *(-- h)*

Reads a 16-bit constant from the instruction stream, sign extends it, and pushes it.

The compiler uses `lit16` instead of `literal` wherever possible, since it saves two to four bytes (depending on alignment). Besides short numeric constants, it can also be used for all execution tokens, since user memory addresses are always less than 32768. (Any greater and `lit16` would make them negative.)

Since: 20061114

Defined by: PropellerForth

L!

(x laddr --)

Stores `x` into Cog-local memory at `laddr`. `L!` can be used to modify Cog registers (or break the kernel).

Cog-local memory is *word-addressed*, unlike shared memory.

Since: 20061114

Defined by: PropellerForth

L@

(laddr -- x)

Loads a cell out of Cog-local memory at `laddr`. `L@` can be used to read Cog registers or peek at the kernel.

Cog-local memory is *word-addressed*, unlike shared memory.

Since: 20061114

Defined by: PropellerForth

<

(n1 n2 -- f)

Checks whether `n1` is less than `n2`; pushes `true` if so, `false` otherwise.

Since: 20061114

Defined by: ANS 6.1.0480

lshift

(x1 x2 -- x3)

Shifts `x1` left by `x2` bits, giving `x3`.

Since: 20061114

Defined by: ANS 6.1.1805

lrot

(x1 x2 -- x3)

Rotates `x1` left by `x2` bits, giving `x3`. Rotation is similar to shifting, but the bits normally lost from the left side reappear at the right side (instead of the right filling with zeroes).

Since: 20061114

Defined by: PropellerForth

max

(n1 n2 -- n3)

`n3` is the larger of `n1` and `n2`.

Since: 20061114

Defined by: ANS 6.1.1870

min

(n1 n2 -- n3)

`n3` is the smaller of `n1` and `n2`.

Since: 20061114

Defined by: ANS 6.1.1880

M

(u1 u2 -- d)

Multiplies `u1` by `u2`, giving the double-cell unsigned product `d`.

Since: 20061114

Defined by: ANS 6.1.1810

negate

(n1 -- n2)

Negates `n1` (as though subtracting it from 0), giving `n2`.

Since: 20061114

Defined by: ANS 6.1.1910

nip

(x1 x2 -- x2)

Discards the next-to-topmost cell on the stack.

Since: 20061114

Defined by: ANS 6.2.1930

-and

(x1 x2 -- x3)

Complements (NOTs) `x2` and bitwise-ANDs with `x1`, giving `x3`. `-and` is equivalent to the sequence `not and`, but uses the Propeller's built-in `andn` instruction and is roughly twice as fast.

Since: 20061114

Defined by: PropellerForth

or

(x1 x2 -- x3)

Bitwise-ORs `x1` and `x2`, giving `x3`.

Since: 20061114

Defined by: ANS 6.1.1980

over

(x1 x2 -- x1 x2 x1)

Pushes a copy of the next-to-topmost cell on the stack.

Since: 20061114

Defined by: ANS 6.1.1990

R>

(R: x1 -- x1)

Moves the top cell of the return stack onto the data stack.

Since: 20061114

Defined by: ANS 6.1.2060

R@

(R: x1 -- x1 R: x1)

Copies the top cell of the return stack onto the data stack.

Since: 20061114

Defined by: ANS 6.1.2070

rshift

(x1 x2 -- x3)

Shifts `x1` right by `x2` bits, giving `x3`.

Since: 20061114

Defined by: ANS 6.1.2162

RSP!

(addr --)

Replaces the current task's return stack pointer with `addr`.

Since: 20061114

Defined by: PropellerForth

RSP@

(-- addr)

Pushes the current task's return stack pointer.

Since: 20061114

Defined by: PropellerForth

rrot

(x1 x2 -- x3)

Rotates `x1` right by `x2` bits, giving `x3`. Rotation is similar to shifting, but the bits normally lost from the right side reappear at the left side (instead of the left filling with zeroes).

Since: 20061114

Defined by: PropellerForth

SP!

(addr --)

Replaces the current task's data stack pointer with `addr`.

Since: 20061114

Defined by: PropellerForth

SP@

(-- addr)

Pushes the current task's data stack pointer (before this instruction was executed).

Since: 20061114

Defined by: PropellerForth

swap

(x1 x2 -- x2 x1)

Exchanges the top two cells on the stack.

Since: 20061114

Defined by: ANS 6.1.2260

>R

(x1 -- R: x1)

Moves the top cell of the stack onto the return stack.

Since: 20061114

Defined by: ANS 6.1.0580

2>R

(x1 x2 -- R: x1 x2)

Moves the top two cells of the data stack onto the return stack, retaining their order.

Since: 20061114

Defined by: ANS 6.2.0340

2/

(n1 -- n2)

Divides `n1` by 2, giving `n2`. `2/` works properly for signed numbers.

Since: 20061114

Defined by: ANS 6.1.0330

U<

(u1 u2 -- f)

Checks whether `u1` is less than `u2`, where both values are treated as unsigned. Returns `true` if so, `false` otherwise.

Since: 20061114

Defined by: ANS 6.1.2340

xor

(x1 x2 -- x3)

Bitwise-Exclusive-ORs `x1` and `x2`, giving `x3`.

Since: 20061114

Defined by: ANS 6.1.2490

wait

(u1 --)

Delays for `u1` machine cycles. Does not allow other tasks to execute on this Cog.

The delay is not entirely accurate, as it disregards the overhead of calling and returning from `wait` itself. Use `waitcnt` to have more control over the delays.

Implementation note: small delays (<100 cycles) may delay for an additional 2^{32} cycles.

Since: 20061114

Defined by: PropellerForth

waitcnt

(count offset -- next-count)

Waits for the Cog's cycle counter to reach `count`, then returns `count` + `offset`.

Implementation note: small delays (<100 cycles) may delay for an additional 2^{32} cycles.

Since: 20061114

Defined by: PropellerForth

waitpeq

(state mask --)

Waits for the input pins specified by `mask` (as 1-bits) to equal the corresponding bits in `state`.

Since: 20061114

Defined by: PropellerForth

"zero-equals"

(x -- f)

Note: the name of this word is `0=`. This wiki does not permit the `=` sign to appear in section headers, so its name is shown as "zero-equals" above.

Checks whether `x` is 0. Returns `true` if so, `false` otherwise.

Since PropellerForth uses 0 to represent `false`, this word can also check if a flag is `false`.

This word is equivalent to the sequence `0 =`, but is much faster.

Since: 20061114

Defined by: ANS 6.1.0270