Website: http://home.hccnet.nl/anij/nof/noforth.html

january 2026

Documentation

noForth m , noForth r and noForth t

```
noForth m is for MSP430, runs in ROM, 1 cell is 16 bits.
noForth r is for RISC-V, runs in ROM, 1 cell is 32 bits.
noForth t is for RP2040, runs in RAM, 1 cell is 32 bits.
```

When we use the word ROM it may be Flash ROM or FRAM.

Until now we have the following noForth variants:

```
m  mv  mc  mcv
r  rv  rc  rcv
t  tv  t#  tv# (solo or duo)

    m = for MSP430
    r = for RISC-V
    t = for RP2040

    c = compact
    v = with vocabularies
only for RP2040:
    # = with multitasker
    solo = noForth on core 0
    duo  = a noForth on both cores
```

"Low Power" noForths for MSP430 are marked with a dash:

```
    m-  mv-  mc-  mcv-
```

- The priorities for noForth are: 1. robust and comfortable, 2. fast, 3. compact.
  For the compact variants: 1. robust and comfortable, 2. compact, 3. fast.
- The noforth images contain only the noForth kernel. For the words `.S WORDS MANY DMP` and `SEE` include the file <noforth m tools.f>, <noforth r tools.f> or <noforth t tools.f>.
- Fetches and stores (when not bytewise) need aligned addresses. No warning appears.
- noForth is case insensitive.

Standard words are not documented here.

(*	D.STR	MSEE
><	DU.	NEXT
@+	DU.STR	?NEGATE
?ABORT	DU*S	NOFORTH\
ADR	DU/S	OK
APP	DU2/	?PAIR
ARSHIFT	'EMIT	PLACE
B+B	?EXIT	R0
B-B	EXTRA	RDROP
BASE?	FLYER	ROM!
BEYOND	FOR	ROMC!
BIA	FRESH	ROMH!
*BIC	FREEZE	ROMMOVE
**BIC	FREEZE2	ROUTINE
*BIS	FROZEN	RTYPE
**BIS	H!	S<>
BIT*	H@	S0
BIT**	H@+	SCAN
BIX	H+H	SEE
*BIX	H-H	SHIELD
**BIX	HOR	SKIP
BL-WORD	HOT	STATE?
BN	HX	TIB
BOUNDS	IB	TIB/
CELL	#IB	+TO
CELL-	>IN?	UMAX
CH	INCR	UMIN
CHERE	INSIDE	UPPER
COLD	IVECS	V:
COLD2	IWORDS	VALUE
DAS	'KEY	VEC!
>DIG	'KEY?	VER
DIG?	LFA>	.VOC
DIVE	LFA>N	X!
DM	M,	X@
DMP	MANY	XC!
DN	MDAS	XC@
?DNEGATE		

Chapters

noForth t runs in RAM. All documentation on this page about compiling in ROM is not valid for noForth t, so there is no `CHERE ROM! ROMC!` etc.

1. noForth m | r | t

Differences between noForth m, noForth r and noFort t

- noForth m is a 16 bits forth.
- The following words are not in noForth m:

```
H!  H@  H@+  ROMH!
[IF] [ELSE] [THEN]
2ROT -ROT -2ROT  2TUCK ARSHIFT
RECURSE
SM/REM
PLACE ( a1 n a2 -- ) \ Write string a1,n to a2 as counted string.
```

You find the noForth m code for these words in the file <noforth m more words.f>.

⇧

2. C and v variants

Conditional compiling:

V: (immediate) is a NOOP in a noForth v variant, in a noForth without vocabularies it is a backslash.

Only in noForth with vocabularies:

> EXTRA is a vocabulary with non-standard useful words.
> INSIDE is a vocabulary with internal words.
>
> ```
> : FRESH ( -- ) only extra also forth also definitions ;
> fresh order ↵ ( FORTH FORTH EXTRA ONLY : FORTH )
> ```
>
> When noForth starts, FRESH is executed.
> .VOC (wid --) \ Show the vocabulary name. 'wid' is a number in 0..127

Only in noForth without vocabularies:

> IWORDS shows the hidden auxiliary words.
> WORDS shows all words except the hidden words. All words can be found normally.

⇧

3. Parsing

BL-WORD (-- adr) \ Execute BL WORD with automatic refill.
BEYOND (char --) \ Ignore input stream (using refill) until 'char' is found. Used in '('.

```
: (  ( -- )  ch )  beyond ; immediate
```

(* \ Multi line comment until *)
*) must be the first word on a line!

⇧

4. Memory layout

FROZEN → HOT

FROZEN is the address in Flash where noForth system data is stored.
When noForth starts, this data is copied to RAM at address HOT where noForth can use it and change it.

HOT → FROZEN

FREEZE copies the actual RAM data to Flash. FREEZE defines how noForth comes back after a reset or COLD .

COLD (--) \ Restart noForth.

Forgetting

SHIELD ('name' --) \ Similar to MARKER . The difference: a shield does not forget itself, a marker does.
The word NOFORTH\ is such a shield; when you execute it, all definitions after NOFORTH\ are gone and only the kernel plus the word NOFORTH\ is left.

MSP430 memory layout of noForth m

The addresses are not the same in all noForth m variants. The labels are forth words. Type the name to get the address on the stack.

RAM

```
HOT         \ warm system data + spaces allotted by programs
HERE        \ actual start of ALLOTtable space and start of
            \ the circulair internal noForth buffer
            \ for BL-WORD S" FLYER and numberprinting
FHERE       \ actual pointer in circulair buffer
TIB         \ input buffer
TIB/        \ end of input buffer
S0          \ data stack (down)
R0          \ return stack and end of RAM
```

Flash ROM

```
ORIGIN      \ start of dictionary
CHERE       \ actual start of free dictionary space
IVECS       \ one cell before the interrupt table
10000       \ Extended memory if present
```

Info block

```
FROZEN      \ cold system data
```

RISC-V memory layout of noForth r

RAM

```
20000000 HOT        \ start of warm system data (max 200 bytes)
     ... UHERE      \ actual start of free Uspace
20000200 FLYBUF     \ circulair FLYER buffer (400 bytes)
     ... FHERE      \ actual pointer in FLYER buffer
20000600 FLYBUF/
20000680 S0         \ data stack (80 bytes down)
20000880 R0         \ return stack (200 bytes down)
20000880 TIB        \ input buffer (80 bytes)
20000900 TIB/
20000900 SYSBUF     \ TIDY buffer (400 bytes)
20000D00 SYSBUF/    \ start of ALLOTted RAM
     ... HERE       \ actual start of free RAM space
20008000 RAMBORDER
20008000 end of RAM
```

Flash ROM

```
 0000    interrupt vectors
 0200    FROZEN      \ cold system data (max 200 bytes)
 0400    ORIGIN      \ start of dicionary
  ...    CHERE       \ start of free dictionary space
1F000    BORDER
20000    end of Flash ROM
```

RP2040 memory layout of noForth t

RAM

```
core 0   core 1
21000000 21020000   IVECS   \ 48 vectors
210000D0 210200D0   IVECS/
210000D0 210200D0   HOT     \ start of warm system data (limit=origin)
... UHERE                   \ actual start of free Uspace
21000200 21020200   ORIGIN  \ the noForth dictionary starts here
... HERE                    \ start of free dictionary space
2101F800 2103F800   FLYBUF  \ circulair FLYER buffer (1024 bytes)
... FP                      \ actual pointer in FLYER buffer
2101FC00 2103FC00   FLYBUF/
2101FE80 2103FE80   R0      \ return stack (640 bytes down)
2101FF80 2103FF80   S0      \ data stack   (256 bytes down)
2101FF80 2103FF80   TIB     \ input buffer (128 bytes)
21020000 21040000   TIB/
21020000 21040000   BORDER  \ systems end
21040000 21040000   MEMTOP  \ end of RAM
```

XRAM External RAM block (Used for multitasker background tasks)

```
20040000 20041000           \ Start XRAM
20041000 20042000           \ End XRAM
```

Flash

```
000000              \ secundairy boot code (max. FC bytes)
(1)0000000 FROZEN   \ saved noForth image nr # 1
(1)0041000 FROZEN2  \ saved noForth image nr # 2
085000              \ Free Flash ROM memory
200000              \ End of Flash ROM memory (max 1000000)
```

5. Utilities

These 5 commands print only one line. Press space bar for next line, press [enter] to leave. Also: press a number key (n=0,1,..9) to display n*4 lines.
SEE ('name' --) \ Decompile, starting at the CF of 'name'.
MSEE (addr --) \ Decompile, starting at addr.
DAS ('name' --) \ Disassemble, starting at the address in the CFA of 'name'.
MDAS (addr --) \ Disassemble, starting at addr.
DMP (addr --) \ A 'dump' that needs only a start address, no count.

MANY (--) \ Restart interpretation of the actual input buffer until a key is pressed.
Example:

```
bl hex ↵  OK
dup emit dup . 1+ many ↵  20 !21 "22 #23 $24  etc.
```

These utility words are not in the noForth kernel. They are in the files <noforth m tools.f>, <noforth r tools.f> or <noforth t tools.f>. together with .S WORDS and DMP.
The disassembler is in the files <noforth m das.f>, <noforth r das.f> or <noforth t das.f>.

⇧

6. Prefixes, Number input

Prefixes

Prefixes are incomplete words. They become a complete word in combination with the immediately following word or text in the input stream. Prefixes are input tools. They read the input stream, both compiling and interpreting. They are not compiled.

Base prefixes

HX DM and BN cause a temporary base-change only while the next word in the input stream is being executed or compiled.

```
hx 10 . ↵  16  OK
: HUNDRED hx 64 ;
hundred . ↵  100  OK
```

These prefixes are made to be used before numbers, but you can also use them interactively before other words. If those words do number output, it will be in the prefixed base.

```
10 hx . ↵  A  OK
' noforth hx dmp ↵  ...
```

The following HX has no effect, because base is 16 only while '.' is compiled...

```
: HAHA hx . ;
10 haha ↵  10  OK
```

Double number prefix

DN makes double number input possible, both compiling and interpreting

```
dn 13579753 d. ↵  13579753  OK
```

A dot at the end is also possible:

```
13579753. d. ↵  13579753  OK
```

Commas in numbers

Number input in noForth may contain commas for readability, noForth ignores them.

```
2,345 . ↵  2345  OK
dn 13,579,753 d. ↵  13579753  OK
```

Combining prefixes

Base prefixes can be used before DN

```
bn dn 1,1111,1111,1111,1111 hx d. ↵  1FFFF  OK
```

⇧

7. Values, more prefixes

A VALUE ('name' --) in a noForth that runs in ROM does not take an initial value from stack when it is defined! It makes no sense to initialize RAM locations at compile time because after a power off/on the data will be lost. Initialisation must be done by the program. This is not the case in noforth t.

```
value KM
```

Value prefixes TO +TO INCR ADR

```
3 to km     km . ↩  3  OK
4 +TO km    km . ↩  7  OK
INCR km     km . ↩  8  OK
ADR km      @ .  ↩  8  OK
```

ADR makes it easy to access a value in assembler:

```
 #1  ADR km &  sub
```

Character prefix

CH (<name> -- ...) is a character prefix and can be used always when the character immediately follows. It puts the value of the first character of 'name' on stack; in definitions that value is compiled as a number. When the character does not follow immediately: use CHAR .

```
ch A . ↩  65  OK
: ....  key  dup  ch ?  = if ... ;
```

⇧

8. System values

IB (-- a) \ Address of actual input buffer. See also memory layout.
#IB (-- n) \ Length of actual input (contents)
APP (-- xt) \ Value, may be set by the user. Contains the token that will be executed at cold start before QUIT is reached. The default token is ' NOOP
OK (-- x) \ Value, may be set by the user.
The lowest 3 bits determine how the prompt looks.
When bit 15 is set, noForth will communicate with ACK/NAK:

```
 ok  hx 8000 or  to ok ( freeze )
```

ACK (06) → noForth is ready to receive a new line.
NAK (15) → noForth is ready to receive a new line (but there was an error).

The value HOR counts the number of characters sent by EMIT. After a CR it is set to zero.
Not in noForth m:

> The value VER counts the number of CRs sent by EMIT .
> ~~The standard variables STATE BASE and >IN also exist as values with the names STATE? BASE? and >IN? . BASE? and TO BASE? do the same as BASE @ and BASE ! .~~
> Versions from november 2025 or later no longer have STATE? BASE? and >IN? .

⇧

9. Program flow

`?EXIT` (flag --) \ short for `IF EXIT THEN`

`?ABORT` (flag --) \ If flag is not zero, the name of the word that has `?ABORT` in it is printed.
Example:

```
: TEST ( x -- )  0= ?abort ;
0 test ↵  Msg from TEST \ Error # F25F
```

The error number = throw number = NFA of the word containing `?ABORT`.
See Error messages.

`DIVE` (--) \ Swap Instruction Pointer with top of return stack; for coroutines.
Example:

```
: (.)  ch (  emit dive  ch )  emit ;
: .ML ( x -- )  (.)  . ." million" ;
67 .ml <enter> (67 million)
```

`DIVE` is used in `FLYER`.

`FLYER` is used in state smart words. `FLYER` handles the state-smartness of words in a uniform way. You need to define the compile time action only.

```
: CCC  FLYER ... ; immediate
```

When CCC executes:
0. In compile time `FLYER` is a no-op.
1. Executing: `FLYER` sets compilation state,
2. the rest of the definition is handled,
3. then state is set back to zero.
4. The just compiled code (in RAM) is executed.
5. The just compiled code (in RAM) is forgotten.

With FLYER and the word -FLY (not in noForth) you can loop interactively:

```
: -FLY    2r> r> 2>r >r ; immediate
CR FLYER TIB 9 FOR COUNT EMIT NEXT DROP -FLY
```

⇧

10. For-Next

For-Next needs only 1 cell on the return stack and is faster than Do-Loop.
`( u )` `FOR` `..` `NEXT` \ loop u times with `I` counting down from u-1 to zero.
Code between `FOR` and `NEXT` is skipped when u = 0.
`I` (-- index) can be used with For-Next as well as with Do-Loop (`I` equals `R@`).

```
: 4x ( -- ) 4 for i . next ;
4x [enter]  3 2 1 0  ok
```

`LEAVE` and `UNLOOP` function only with Do-Loop. Use `RDROP` or `R>` to leave a For-Next conditionally:

```
: ccc1 .. for .. key? if r> exit then .. next -1 ;
```

`WHILE` can be used with For-Next and Do-Loop:

```
: ccc2 .. do .. key? 0= while .. loop .. else .. unloop then .. ;
: ccc3 .. for .. key? 0= while .. next .. else .. rdrop then .. ;
```

`NEXT` is state-smart:
In a colon definition the `NEXT` of For-Next is compiled.
In assembler the `NEXT` of the inner interpreter is assembled. ♒

⇧

11. Bit manipulation

`**BIC` (mask addr --) \ AND cell in addr with inverted mask
`**BIS` (mask addr --) \ OR cell in addr with mask
`**BIX` (mask addr --) \ XOR cell in addr with mask
`BIT**` (mask addr -- x) \ AND mask with cell in addr
The same words with one star operate on the lower half of a cell: `*BIC` `*BIS` `*BIX` `BIT*`

Avoiding name conflicts, only in noForth m (MSP) assembler:

> `BIA` is the name for MSP430 assembler AND
> `BIX` is the name for MSP430 assembler XOR

⇧

12. ROM / RAM (not for noForth t)

In noForth FRAM or Flash is treated as FROM.
HERE (-- a) \ RAMhere in data-space
ALLOT (n --) \ Reserve n byte at RAMhere
CHERE (-- a) \ ROMhere

`! C! +! MOVE` cannot be used with a ROM destination.
The words ROM! ROMC! ROMMOVE do exist, but you should not need them.
Use `, C, M,` instead.
M, (multi-c, or memory,) is a noForth word for the MOVE to ROM function:

```
: M, ( a n -- ) for count c, next ; \ Compile the string a,n at CHERE
```

Constant string to ROM? Use the comma-words

```
create LOGO1
s" noForth"  dup c, M, align
logo1 count type ↵  noForth  OK
```

Changeable string to RAM? Use ALLOT

```
create LOGO2   10 allot
s" noForth" logo2  2dup c!  1+ swap move
\ or
s" noForth" logo2  place
logo2 count type ↵  noForth  OK
```

Only for noForth r RISC-V

> ROMH! (16b a --) \ write 16 bits to address a
> Writing to Flash goes in portions of 16 bits, so ROMC! does not exist in noForth r. Yet C, is possible in noForth r because sequential bytes are written pairwise. This means:
> • C, and M, can be freely mixed but at the end an ALIGN is needed.
> • The dictionary pointer CHERE is updated after each pair of bytes written, so it is never odd.

⇧

13. Strings

S<> (a1 n1 a2 n2 -- t|f) \ Compare strings, true → not equal
UPPER (a n --) \ Capitalize characters in string a,n in RAM

```
: RTYPE ( a n r -- ) 2dup min - spaces type ;
: BOUNDS ( addr len -- enda addr ) over + swap ;
```

: SKIP (endaddr addr1 ch -- endaddr addr2) \ First char<>ch is at addr2.
: SCAN (endaddr addr1 ch -- endaddr addr2) \ First char=ch found at addr2.
When 'endaddr' = 'addr2' → Character is not found.
SKIP and SCAN are used in BL-WORD and PARSE

⇧

14. Double numbers

`DU.` (du --)
`DU*S` (du u -- dprod) \ Unsigned
`DU/S` (du u -- dquot rest) \ Unsigned, rest in tos!
`DU2/` (du -- du/2) \ Logical drshift

Number>String
`D.STR` (dn -- adr len)
`DU.STR` (du -- adr len)
The string adr/len has a very short life. Parsing the next word will overwrite the string, so you can not use these words interactively.

⇧

15. Interrupt vectors

`ROUTINE` starts the code definition for a interrupt routine or a subroutine. When executed it does not start the routine but it puts the address of the routine on the stack, so you can use it for a call or put it in a vector. Use 'return from interrupt' or 'return from subroutine' instead of NEXT.

```
routine INTERRUPT  ..assembler code..  MRET/RETI end-code
```

`VEC!` (a ia --) \ Write vector into interrupt vector table.

a = address of interrupt routine, ia = location in interrupt vector table
Only for noForth m (MSP430):

> `IVECS` (-- a) \ The address of the cell just below the vector table. It contains a return from interrupt.
> Empty vectors should point to `IVECS`

⇧

16. Extended memory (MSP430)

Only for noForth m (MSP430)

> `X!` (x da --) \ da = double number address
> `X@` (da -- x)
> `XC!` (ch da --)
> `XC@` (da -- ch)
> All noForth MSP430 FRAM versions with extended memory above FFFF provide these four commands. These commands take a double number as address. Example:
>
> ```
> hex 40 dn 12345 xc!
> ```

⇧

17. Miscellaneous

```
: CELL   1 cells ;
: CELL-  1 cells - ;
: @+       ( a -- a+cell a@ ) dup cell+ swap @ ;
: ?PAIR    ( x y -- )         <> ?abort ;
: ?NEGATE  ( x y -- x2 )      0< if  negate then ;
: ?DNEGATE ( dx y -- dx2 )    0< if dnegate then ;
```

UMIN (u1 u2 -- u3) \ Unsigned MIN
UMAX (u1 u2 -- u3) \ Unsigned MAX
ARSHIFT (x n -- x2) \ Arihmetical rshift
RDROP (--) \ Short for R> DROP
LFA> (lfa -- cfa)
LFA>N (lfa -- nfa)

Number conversion

>DIG (n -- char)
DIG? (char base -- n true | char false)

Swap bytes

>< (x -- y) \ hex 1234 -- 3412

Join or separate parts of a cell

B+B (x y -- z) \ hex 12 34 -- 3412
B-B (z -- x y) \ hex 1234 -- 34 12

not in noForth m

> H+H (x y -- z) \ hex 1234 5678 -- 56781234
> H-H (z -- x y) \ hex 12345678 -- 5678 1234

⇧

18. Error messages

Msg from	Meaning
?BASE	Base is reset, was not in [2,42)
?COMP	Only compiling
?COND	Invalid condition (assembler)
?PAIR	Unstructured code
?STACK	Stack underflow or stack overflow
'	Name not found
(*	*) not found
>FHERE	Not enough RAM space
ALLOT	Data space full
UALLOT	Udata space full
ALSO	Search order overflow
BEYOND	Could not refill
CHAR	End of input stream
CHERE?	Dictionary full
DIST	Distance too large in control structure
DN	Not a number
DST	Invalid destination address (assembler)
DN	Not a double number
HEADER	Name length not in [1,32)
HX	What's this?
INTERPRET	What's this?
MPU	Trying to write to protected memory
POSTPONE	Name not found
PREVIOUS	Only one vocabulary in search order
RECURSE	RECURSE not possible after DOES>
ROM!	Write action did not succeed
ROMC!	Write action did not succeed
SET-ORDER	Search order overflow
SRC	Invalid source address (assembler)
STOP?	Interrupted by user
THROW	No catch-frame found
TO	Prefix not accepted
VEC!	Could not install interrupt vector
[']	POSTPONE could not find name

⇧

***