



noForth for MSP430 documentation

(*	?DNEGATE	MSEE
><	DU.	?NEGATE
@+	DU*S	NOFORTH\
?ABORT	DU/S	OK
ADR	DU2/	?PAIR
APP	'EMIT	R0
BEYOND	?EXIT	RDROP
*BIC	EXTRA V	ROM!
**BIC	FLYER	ROMC!
*BIS	FRESH V	ROMMOVE
**BIS	FREEZE	ROMTEST
BIT*	FROZEN	ROUTINE
BIT**	>GIE	RTYPE
*BIX	GIE>	S<>
**BIX	HOR	S0
BL-WORD	HOT	SCAN
BN	HX	SEE
BOUNDS	IB	SHIELD
CELL	#IB	SKIP
CELL-	INCR	TIB
CH	INSIDE V	TIB/
CHERE	IVECS	+T0
COLD	IWORDS C	UPPER
DAS	'KEY	VALUE
>DIG	'KEY?	VEC!
DIG?	LFA>	.VOC V
DIVE	LFA>N	X!
DM	M,	X@
DMP	MANY	XC!
DN	MDAS	XC@

**Standard words are not documented here.
In this text you find information about:**

1. Cold start data
2. Memory maps
3. Only in noForth C
4. Only in noForth V
5. Prefixes, Number input
6. Values, more prefixes
7. Program flow
8. Utilities
9. Extended memory access
10. System values
11. Bit manipulation
12. Parsing
13. ROM / RAM
14. Strings
15. Interrupt vectors
16. Double unsigned
17. Miscellaneous
18. Word headers in noForth
19. Error messages

noForth is case insensitive

1. Cold start data

FROZEN → HOT

FROZEN is the address of a ROM info-block with noForth system data.

When noForth starts, these data are copied into RAM at address **HOT** where noForth can use it and change it.

HOT → FROZEN

FREEZE copies the actual RAM data into the ROM info block.

Cold start data in the FROZEN block

Type this in the terminal: `frozen msec`

What you see is a list of the cold start data. (Compare this to: `hot msec`)

This is what you could see:

```
frozen msec
1800 EB18 ( Top words of the dictionary threads (usually 8 cells)
1802 ( EB28 (
1804 B EB42 (
1806 E8BC (
1808 E808 (
180A l E96C (
180C E7BE (
180E E988 (
1810 DAF4 PFX-LINK value, contains top word of the prefix-list
1812 DF84 ( ASSEMBLER ) WID-LINK value, (only in noForth-V, not in noForth-C)
1814 C89E EMIT ) 'EMIT value, contains token of emit action, default EMIT)
1816 C8B4 KEY?) 'KEY? value, contains token of key? action, default KEY?)
1818 C8A8 KEY) 'KEY value, contains token of key action, default KEY)
181A C40C NOOP APP value, contains application token, default NOOP
181C X EB58 ROMP value, ROMhere
181E J 204A HERE value, RAMhere
1820 . 2EB0 TIB value, terminal input buffer
1822 / 2F00 TIB/ value, end of TIB and bottom of data stack
1824 / 2F80 S0 value, top of data stack and bottom of return stack
1826 7 OK value, set bits activate prompt functions
1828 10 BASE variable
182A FFFF not used
182C FFFF "
182E FFFF "
1830 FFFF "
1832 7CF ( 14 bytes of processor dependent hardware configuration data.
1834 600 ( See the readme file in the zip files with the noForth binaries
1836 F4 ( for the different MSP430 processors.
1838 800 (
183A 201 (
183C 4 (
183E QU 5551 (
```

- **BASE** (ram address) is in the last cell of the cold start data.
STATE is in the first RAM address after **BASE**, so the length in bytes of the cold data area is: `STATE HOT -`
- For **MARKER** and **SHIELD** it is necessary that **PFX-LINK** (and **WID-LINK**) remain positioned immediately after the dictionary threads.



2. Memory maps

	----- RAM -----				----- INFO block -----	----- FROM/FRAM -----									
	HOT	TIB	TIB/	S0	R0	FROZEN		ORIGIN	IVECS						
	HERE	HERE				*		CHERE	**						
G2553	[0200	..	0330	0380	03C0]0400	[1080	10B2]10C0	-	[C000	..	FFDE]10000	-
G2955	[1100	..	1FB0	2000	2080]2100	[1080	10B2]10C0	-	[2100	..	FFDE]10000	-
F149	[0200	..	08B0	0900	0980]0A00	[1000	1072]1080	-	[1100	..	FFDE]10000	-
FR5739	[1C00	..	1F30	1F80	1FC0]2000	[1800	1872]1880	-	[C200	..	FF7E]10000	-
FR59X9	[1C00	..	22B0	2300	2380]2400	[1900	1972]1980	-	[4400	..	FF7E	10000]14000
FR5994	[1C00	..	2AB0	2B00	2B80]2C00	[1900	1972]1980	-	[4000	..	FF7E	10000]24000
FR2433	[2000	..	2EB0	2F00	2F80]3000	[1800	1832	1840]1A00	[C400	..	FF7E]10000	-

RAM configuration: **TIB**, **TIB/** and **S0** are values, so you can move them somewhat up or down. Do not change their order!

RAM

- HOT to HERE contains noForth data, links, values, variables.
- HERE -- end of allotted space with noForth data. HERE moves up when new values or variables are defined (with ALLOT).
- HERE to TIB -- circular internal noForth buffer for WORD, S", FLYER, number-output, etc.
- TIB to TIB/ -- terminal input buffer.
- S0 down to TIB/ -- data stack.
- R0 down to S0 -- return stack.

INFO block

- FROZEN to * contains the cold start data which will be moved to HOT when noForth starts. FREEZE moves the data in the opposite direction, from HOT to FROZEN.
- * -- start of the CONFIG data (14 bytes). The rest of the INFO block contains noForth code (only for FR2433).

FROM or FRAM

- ORIGIN -- start of the noForth dictionary.
- CHERE -- end of the noForth dictionary. CHERE moves up when new definitions are added (with the comma-words).
- IVECS -- address of a cell containing the RETI command. The cells thereafter contain the interrupt vectors. The last one, the RESET vector is at FFFE. Empty interrupt vectors point to IVECS (RETI = return from interrupt).
- ** -- start of extended memory, if present.



3. Only in noForth C

IWORDS shows the inside words (hidden auxiliary words).

WORDS shows all words except the inside words.

All words can be found normally.



4. Only in noForth V

EXTRA is a vocabulary with non-standard useful words.

INSIDE is a vocabulary with internal words.

FRESH is defined as:

```
: 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



5. 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
```



6. Values, more prefixes

A **VALUE** ('name' --) in noForth 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.

```
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.

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

Use CHAR when the character does not follow immediately.



7. 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 only need to define the compile time action.

Example:

```
: S" flyer postpone s"  
ch " parse dup c, m,  
align ; immediate
```

Execution of S" :

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.

COLD (--) \ Restart noForth.

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.



8. Utilities

These 5 commands print only one line. Press space bar for next line, press [enter] to leave.

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:

```
b1 hex ↵ OK
dup emit dup . 1+ many ↵ 20 !21 "22 #23 $24 etc.
```



9. Extended memory access

X! (x da --)

X@ (da -- x)

XC! (ch da --)

XC@ (da -- ch)

All noForth MSP430 FRAM versions with extended memory above FFFF provide these four commands. From february 2017 these commands take a double number as address.

Example:

```
hex 40 dn 12345 xc!
```



10. System values

IB (-- a) \ Address of actual input buffer

#IB (-- n) \ Length of actual input (contents)

See also [memory maps](#).

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 the highest bit 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).



11. Bit manipulation

***BIC** (mask addr --) \ AND byte in addr with inverted mask

***BIS** (mask addr --) \ OR byte in addr with mask

***BIX** (mask addr --) \ XOR byte in addr with mask

BIT* (mask addr -- x) \ AND mask with byte in addr

The 16 bits versions are: ****BIC** ****BIS** ****BIX** **BIT****

GIE> (-- 8 | 0) \ Save Gie-bit on stack and disable interrupts

<GIE (x --) \ Restore Gie-bit (x 8 and) in Status Register



12. Parsing

BL-WORD \ 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 *****)

Both **(*** and *****) must be the first word on a line!



13. ROM / RAM

HERE (-- a) \ RAMhere in data-space

ALLOT (n --) \ Reserve n byte at RAMhere

CHERE (-- a) \ ROMhere (you should not need it)

ROMTEST (-- a) \ Detect CHERE (not in FRAM versions)

! 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, is a special noForth word for the MOVE to ROM function:

M, (a n --) \ Compile the string a,n at CHERE

Constant string in ROM? Use the comma-words

```
create LOG01
s" noForth" dup c, M, align
logo1 count type ↵ noForth OK
```

Changeable string in RAM? Use ALLOT

```
create LOG02 10 allot
s" noForth" logo2 2dup c! 1+ swap move
logo2 count type ↵ noForth OK
```



14. Strings

S<> (a1 n1 a2 n2 -- t|f) \ Compare strings, true → not equal

UPPER (a n --) \ Capitalize characters in string a,n in RAM

The value **HOR** holds the number of characters sent by **EMIT**. After a CR it is zero.

```
: 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 **WORD** and **PARSE**



15. Interrupt vectors

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

a = address of interrupt routine, ia = location in interrupt vector table

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

ROUTINE \ This word starts the assembler definition of a interrupt routine. When you type the name of the routine it will put its address on the stack so you can store it easily in a vector. Use **RETI** in stead of **NEXT**.

```
routine INTERRUPT ...assembler... reti end-code
```

When you end with **RP)+ PC MOVE (= RET)** in stead of **NEXT** you can use it together with **CALL** as a normal subroutine.

```
routine SBR-ONE ...assembler... rp )+ pc mov end-code
code TEST-ONE ... sbr-one # call ... next end-code
```

-->



16. Double unsigned

DU. (du --)

DU*S (du u -- dprod) \ Unsigned

DU/S (du u -- dquot rest) \ Unsigned, rest in tos!

DU2/ (du -- du/2) \ Logical drshift



17. Miscellaneous

RDROP (--) \ Short for R> DROP

<> (x -- y) \ Byte swap, hex 12AB → AB12

```
: @+ ( a -- a+2 a@ ) dup cell+ swap @ ;  
\ 16bit variant of count
```

```
: ?PAIR ( x y -- ) <> ?abort ;  
: ?NEGATE ( x y -- x2 ) 0< if negate then ;  
: ?DNEGATE ( dx y -- dx2 ) 0< if dnegate then ;
```

Number conversion:

>DIG (n -- char)

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

```
: CELL ( -- 2 ) 2 ;  
: CELL- ( a -- a-2 ) 2 - ;
```

LFA> (lfa -- cfa)

LFA>N (lfa -- nfa)



18. Word headers in noForth

- In noForth an LFA points to an LFA.
- Each of the first (8) cells at HOT (RAM) and FROZEN (ROM) points to the LFA in the newest word of that dictionary thread.
- Headers in noForth C variants differ from headers in V variants.

C headers

```
linkfield  one byte
icnt       one byte
name       name(+FF if alignment is needed)
code field one cell (indirect threaded)
```

```
icnt       = ij0n,nnnn (bits)
  i=0      → immediate word
  j=1      → inside word
  n,nnnn   → name length
```

A link field (LFA) contains the distance (backwards) in cells to the LFA in the preceding word header of that dictionary thread. When the distance is too large to fit in a byte the LFA contains zero and the real address is in the cell preceding the LFA.

```
: LNK@ ( lfa1 -- lfa2 )
  dup c@ ?dup if 2* - exit then
  cell- @ ;
```

V headers

```
link field one cell
hvoc       one byte
icnt       one byte
name       name(+FF if alignment is needed)
code field one cell (indirect threaded)
```

```
icnt       = i00n,nnnn (bits)
  i=0      → immediate word
  n,nnnn   → length of the name
```

```
hvoc       = ivvv,vvvv (bits)
  i=1      → name was unique
  i=0      → name already existed
  vvv,vvvv → the wordlist (wid) to which the word belongs
            (wid = 0..127)
```

A link field (LFA) points to the LFA in the preceding word header of that dictionary thread.

```
: LNK@ ( lfa1 -- lfa2 ) @ ;
```



19. 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
>FHRE	Not enough RAM space
ALLOT	Data space full
ALSO	Search order overflow [V]
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
PREVIOUS	Only one vocabulary in search order [V]
RECURSE	RECURSE not possible after DOES>
ROM!	Write action did not succeed
ROMC!	Write action did not succeed
SET-ORDER	Search order overflow [V]
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

RECURSE and SET-ORDER are in the file "more standard words".



*