april 2015 - AN
Website: http://home.hccnet.nl/anij/nof/noforth.html

noForth C,V for MSP430 documentation

```
><          HOT
@+          HX
?ABORT      IB
APP         #IB
BEYOND      INCR
*BIC        INSIDE V
**BIC       IVECS
*BIS        IWORDS C
**BIS       'KEY
BIT*        'KEY?
BIT**       LFA>
*BIX        LFA>N
**BIX       M,
BL-WORD     MANY
BN          MDAS
BOUNDS      MSEE
CELL        ?NEGATE
CELL-       NOFORTH
CH          OK
CHERE       ?PAIR
COLD        R0
DAS         RDROP
>DIG        ROM!
DIG?        ROMC!
```

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

```
DIVE        ROMMOVE
DM          ROMTEST
DN          RTYPE
?DNEGATE    S<>
DUMP        S0
DU.         SCAN
DU*S        SEE
DU/S        SHIELD
DU2/        SKIP
'EMIT       STOP?
?EXIT       TIB
EXTRA V     TIB/
FLYER       +TO
FRESH V     UPPER
FREEZE      VALUE
FROZEN      VEC!
FFBASE      .VOC V
```


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

```
@)frozen msee
  1080     E7AA          \ Top words of the dictionary threads, usually 8.
  1082     E7BA          \
  1084     E7D2          \
  1086 .   E12E          \
  1088 d   E464          \
  108A J   E44A          \
  108C     E416          \
  108E     E3AE          \
  1090     C504          \ PFX-LINK  - variable, contains top word of the prefix-list
  1092     DC02          \ WID-LINK  - variable, (only in noForth-V, not in noForth-C)  ***
  1094 :   C23A EMIT)    \  'EMIT     - value, contains token of  emit action, default EMIT)
  1096     C200 KEY?)    \  'KEY?     - value, contains token of key? action, default KEY?)
  1098     C21C KEY)     \  'KEY      - value, contains token of key action, default KEY)
  109A >   C03E NOOP     \ APP        - value, contains application token, default NOOP
  109C     E7E8          \ ROMP       - value, ROMhere
  109E J    24A          \ HERE       - value, RAMhere
  10A0 0    330          \ TIB        - value, terminal input buffer
  10A2      380          \ TIB/       - value, end of TIB  and bottom of data stack
  10A4      3C0          \ S0         - value, top of data stack  and bottom of return stack
  10A6      1C0          \ MS)        - value, MS tuning factor
  10A8        7 #3       \ OK         - value, set bits activate prompt functions
  10AA       10          \ BASE       - variable
  10AC     FFFF #FFFF    \ ...        - not used
```

```
10AE     FFFF #FFFF    \
10B0     FFFF #FFFF
10B2     FFFF #FFFF
```

- 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 for 3 processors

```
MSP430      G2553       FR5739      F149
            ------      ------      ------

            FFFF        FFFF **)    FFFF                flashROM ↓
            FFE0, 20↑   FF80, 80↑   FFE0, 20↑ ir. vector table
IVECS       FFDE        FF7E        FFDE
CHERE       ...         ...         ...
ORIGIN      C000        C200        1100                flashROM ↑

            10C0        1900        1080
FROZEN      1080, 40↑   18C0, 40↑   1000, 80↑           info block ↑

R0          400, 40↓    2000, 40↓   A00, 80↓  return stack
            3FF         1FFF        9FF                 RAM ↓
S0    *)    3C0, 40↓    1FC0, 40↓   980, 80↓  data stack
TIB/ *)     380         1F80        900
TIB  *)     330, 50↑    1F30, 50↑   8B0, 50↑  input buffer
HERE        ...         ...        ...
HOT         200         1C00        200                 RAM ↑
```

*) RAM configuration: `TIB` `TIB/` and `S0` are values, so you can move them somewhat up or down. Do not change their order!
**) flashROM is flashRAM for MSP430 FR5739

⇧

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, `INSIDE TICK FIND-ALL` are no longer needed and are romoved from noForth C.

(In older noForth C versions, before march 2015:
Inside words cannot be found with `'` or `FIND`
With `TICK` or `FIND-ALL` all words can be found.

```
TICK ( 'name' -- xt )
FIND-ALL ( csa -- csa 0 | xt imm? ) \ csa = counted string address

INSIDE ( 'ccc' -- ) \ To be used before inside words.
inside ccc \ In definitions: ccc will be compiled.
inside ccc \ Executing: ccc will be executed.)
```

⇧

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. Number input, prefixes

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, they are not compiled.

Base prefixes

HX DM and BN are defined with the defining word FFBASE .

```
decimal
```

```
16 ffbase HX
10 ffbase DM
2 ffbase BN
```

They 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 they do number output, it will be in the prefixed base.

```
10 hx . ↩  A  OK
' noforth 8 hx dump ↩  ...
```

But

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

This last HX had no effect. Why? - Because base is 16 while '.' is compiled...

Double number prefix

DN makes double number input possible, both compiling and interpreting

```
dn 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' --) does not take an initial value from stack! 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

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

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 --) \ is short for IF EXIT THEN

?ABORT (flag --) \ If flag is not zero, the name of the definition 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 definition 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-smart-ness 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` (--) has the same effect as power off/on.
`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

SEE ('name' --) \ Decompile, starting at the CFA of 'name'.
MSEE (addr --) \ Decompile, starting at addr.
DAS ('name' --) \ Disassemble, starting at the CFA of 'name'.
MDAS (addr --) \ Disassemble, starting at addr.
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.
```

STOP? (-- x)
STOP? is used in words like DUMP SEE MSEE MANY that produce scrolling text. The effect of STOP? in these words is:

Space bar	→ pause or continue, (x = false)
Esc-key	→ abort
Any other key	→ stop, (x = true)

⇧

9. 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).

10. 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**

⇧

11. 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
```

⇧

12. 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 (max 4F chars)

Constant string in ROM? Use the comma-words!

```
create LOGO1
```

```
s" noForth"  dup c,  M,  align
logo1 count type ↩  noForth  OK
```

Changeable string in RAM? Use ALLOT !

```
create LOGO2   10 allot
s" noForth" logo2  2dup c!  1+ swap move
logo count type ↩  noForth  OK
```

⇧

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

⇧

14. 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

⇧

15. Double unsigned

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

⇧

16. 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)

⇧

17. Error messages

from which word	meaning
Msg from '	Name not found
Msg from ALSO	Search order overflow [V]
Msg from ?BASE	Base is reset, was not in [2,42)
Msg from BEYOND	Could not refill
Msg from CHAR	End of input stream
Msg from CHERE?	Dictionary full
Msg from ?COMP	Only compiling

Msg from ?COND	Invalid condition (assembler)
Msg from CREATE)	Name length not in [1,32)
Msg from DST	Invalid destination address (assembler)
Msg from DIST	Distance too large in control structure
Msg from !DOER	CF has already been written
Msg from DOPREFIX	Prefix not accepted
Msg from >FHERE	Not enough RAM space
Msg from INTERPRET	What's this?
Msg from M,	String longer than 79(x4F) chars
Msg from ?PAIR	Unstructured code
Msg from PARSE	Delimiter not found
Msg from POSTPONE	Name not found
Msg from PREVIOUS	Only one vocabulary in search order [V]
Msg from RECURSE	RECURSE not allowed after DOES>
Msg from .S	Stack error
Msg from SET-ORDER	Search order overflow [V]
Msg from SRC	Invalid source address (assembler)
Msg from ?STACK	Stack underflow or stack overflow
Msg from STOP?	Interrupted by user
Msg from THROW	No catch-frame found
Msg from VEC!	Could not install interrupt vector

⇧