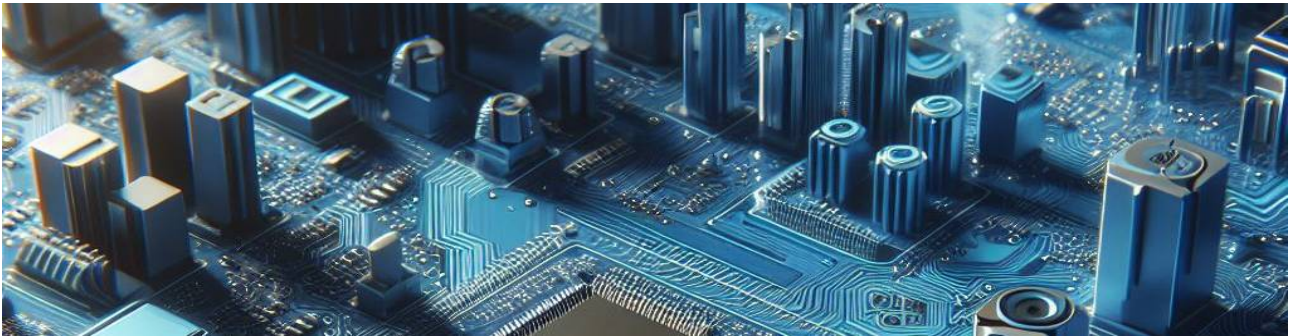


eForth Windows

Manuel de référence

version 1.0 - 30 novembre 2023



Auteur

- Marc PETREMANN

petremann@arduino-forth.com

Table des matières

Auteur.....	1
forth.....	3
graphics.....	56
Mots FORTH par utilisation.....	58
arithmetic integer.....	58
arithmetic real.....	58
block edit list.....	59
chars strings.....	59
comparaison logical.....	59
definition words.....	59
display.....	60
files words.....	61
loop and branch.....	61
memory access.....	61
stack manipulation.....	62

forth

! **n addr --**

Stocke une valeur entière n à l'adresse addr.

```
VARIABLE TEMPERATURE
32 TEMPERATURE !
```

**d1 -- d2**

Effectue une division modulo la base numérique courante et transforme le reste de la division en chaîne de caractère. Le caractère est déposé dans le tampon défini à l'exécution de **<#**

```
: hh ( c -- adr len)
  base @ >r hex
  <# # # #>
  r> base !
;
3 hh type \ display 03
26 hh type \ display 1a
```

#! **--**

Se comporte comme **** pour ESP32forth.

Sert d'en-tête de fichier texte pour indiquer au système d'exploitation (de type Unix) que ce fichier n'est pas un fichier binaire mais un script (ensemble de commandes). Sur la même ligne est précisé l'interpréteur permettant d'exécuter ce script.

```
#!/usr/bin/env ueforth
```

#> **n -- addr len**

Dépile n. Rend la chaîne de sortie numérique mise en forme sous forme de chaîne de caractères. *addr* et *len* spécifient la chaîne de caractères résultante.

```
\ display address in format: NNNN-NNNN
: DUMPaddr ( n -- )
  <# # # # # [char] - hold # # # # #>
  type
;
;
```

#FS **r --**

Convertit un nombre réel en chaîne de caractères. Utilisé par **f**.

#s d1 -- d=0

Convertit le reste de d1 en chaîne de caractères dans la chaîne de caractères initiée par <#.

```
: EUROS ( d1 --- str len)
  <#
  # #          \ convert € cents
  [char] , hold \ add char "," to str buffer
  #s #>        \ convert rest after ","
;
15630. EUROS type \ display 156,30 ok
```

#tib -- n

Nombre de caractères reçus dans le tampon d'entrée du terminal.

' exec: <space>name -- xt

Recherche <name> et laisse son code d'exécution (adresse).

En interprétation, ' xyz EXECUTE équivaut à xyz.

```
defer xEmit
: vxEmit ( c ---)
  1+ emit ;
' vxEmit is xEmit
```

'tib -- addr

Pointeur vers le tampon d'entrée du terminal.

(local) a n --

Mot utilisé pour gérer la création des variables locales.

* n1 n2 -- n3

Multiplication entière de deux nombres.

```
6 3 * \ push 18 operation 6*3
7 3 * \ push 21 operation 7*3
-7 3 * \ push -21
7 -3 * \ push -21
-7 -3 * \ push 21
```

*/ n1 n2 n3 -- n4

Multiplie n1 par n2 produisant le résultat intermédiaire à double précision d. Divise d par n3 en donnant le quotient entier n4.

```
5000 1000 4000 */ . \ display 1250
```

***/MOD** **n1 n2 n3 -- n4 n5**

Multiplie n1 par n2 produisant le résultat intermédiaire à double précision d. Divise d par n3 produisant le reste entier n4 et le quotient entier n5.

```
50000 10 4001 */MOD . \ display 124 3876
```

+ **n1 n2 -- n3**

Laisse la somme de n1 et n2 sur la pile.

```
7 15 + \ leave 22 on stack
```

+! **n addr --**

Incrémente le contenu de l'adresse mémoire pointé par addr.

```
variable valX
15 valX !
1 valX +!
valX ? \ display 16
```

+loop **n --**

Incrémente l'index de boucle de n.

Marque la fin d'une boucle **n1 0 do ... n2 +loop**.

```
: loopTest
  100 0 do
    i .
    5 +loop
  ;
loopTest \ display 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95
```

+to **n --- <valname>**

incrémente de n le contenu de *valname*

```
5 value FINAL-SCORE
1 +to FINAL-SCORE \ increment content of FINAL-SCORE
FINAL-SCORE . \ display 6
```

, **x --**

Ajoute x à la section de données actuelle.

- **n1 n2 -- n1-n2**

Soustraction de deux entiers.

```
6 3 - . \ display 3
```

```
-6 3 - . \ display -9
```

-rot **n1 n2 n3 -- n3 n1 n2**

Rotation inverse de la pile. Action similaire à **rot rot**

. **n --**

Dépile la valeur au sommet de la pile et l'affiche en tant qu'entier simple précision signé.

```
1 . \ display 1
1 2 . \ display 2 leave 1 on stack
1 2 + . \ display 3 addition 1 and 2, leave nothing on the
stack
6 3 * . \ display 18
7 3 * 6 3 * + . \ display 39 operation (7*3)+(6*3)
```

." **-- <string>**

Le mot **."** est utilisable exclusivement dans une définition compilée.

A l'exécution, il affiche le texte compris entre ce mot et le caractère **"** délimitant la fin de chaîne de caractères.

```
: TITLE
  ."        GENERAL MENU" CR
  ."        =====" ;
: line1
  ." 1.. Enter datas" ;
: line2
  ." 2.. Display datas" ;
: last-line
  ." F.. end program" ;
: MENU ( ---)
  title cr cr cr
  line1 cr cr
  line2 cr cr
  last-line ;
```

.s **--**

Affiche le contenu de la pile de données, sans action sur le contenu de cette pile.

/ **n1 n2 -- n3**

Division de deux entiers. Laisse le quotient entier sur la pile.

```
6 3 / . \ display 2 opération 6/3
7 3 / . \ display 2 opération 7/3
8 3 / . \ display 2 opération 8/3
9 3 / . \ display 3 opération 9/3
```

/mod **n1 n2 -- n3 n4**

Divise n1 par n2, donnant le reste entier n3 et le quotient entier n4.

```
22 7 /MOD . . \ display 3 1
```

0< **x1 --- fl**

Teste si x1 est inférieur à zéro.

0<> **n -- fl**

Empile -1 si n <> 0

0= **x -- fl**

Teste si l'entier simple précision situé au sommet de la pile est nul.

```
5 0= \ push FALSE on stack
0 0= \ push TRUE on stack
```

1+ **n -- n+1**

Incrémente la valeur située au sommet de la pile.

1- **n -- n-1**

Décrémente la valeur située au sommet de la pile.

1/F **r -- r'**

Effectue une opération 1/r.

```
12e 1/F f. \ display 0.083333 (op: 1/12)
```

2! **d addr --**

Stocke la valeur double précision d à l'adresse addr.

2* **n -- n*2**

Multiplie n par deux.

2/ **n -- n/2**

Divise n par deux.

n/2 est le résultat du décalage de n d'un bit vers le bit le moins significatif, laissant le bit le plus significatif inchangé.

```
24 2/ . \ display 12
25 2/ . \ display 12
```

```
26 2/ . \ display 13
```

2@ **addr -- d**

Empile la valeur double précision d stockée à l'adresse addr.

2drop **n1 n2 n3 n4 -- n1 n2**

Retire la valeur double précision du sommet de la pile de données.

```
1 2 3 4 2drop \ leave 1 2 on top of stack
```

2dup **n1 n2 -- n1 n2 n1 n2**

Duplique la valeur double précision n1 n2.

```
1 2 2dup \ leave 1 2 1 2 on stack
```

4* **n -- n*4**

Multiplie n par quatre.

4/ **n -- n/4**

Divise n par quatre.

: **comp: -- <word> | exec: --**

Ignore les délimiteurs d'espace de début. Analyse le nom délimité par un espace. Crée une définition pour le , appelée "définition deux-points". Entre dans l'état de compilation et démarre la définition actuelle.

L'exécution ultérieure de **NOM** réalise l'enchaînement d'exécution des mots compilés dans sa définition "deux-points".

Après **:** **NOM**, l'interpréteur entre en mode compilation. Tous les mots non immédiats sont compilés dans la définition, les nombres sont compilés sous forme littérale. Seuls les mots immédiats ou placés entre crochets (mots **[** et **]**) sont exécutés pendant la compilation pour permettre de contrôler celle-ci.

Une définition "deux-points" reste invalide, c'est à dire non inscrite dans le vocabulaire courant, tant que l'interpréteur n'a pas exécuté **;** (point-virgule).

```
: NAME nomex1 nomex2 ... nomexn ;  
NAME \ execute NAME
```

:noname **-- cfa-addr**

Définit un code FORTH sans en-tête. cfa-addr est l'adresse d'exécution d'une définition.


```

:noname s" Saterdag" ;
:noname s" Friday" ;
:noname s" Thursday" ;
:noname s" Wednesday" ;
:noname s" Tuesday" ;
:noname s" Monday" ;
:noname s" Sunday" ;

create (ENday) ( --- addr)
      , , , , , , ,

:noname s" Samedi" ;
:noname s" Vendredi" ;
:noname s" Jeudi" ;
:noname s" Mercredi" ;
:noname s" Mardi" ;
:noname s" Lundi" ;
:noname s" Dimanche" ;

create (FRday) ( --- addr)
      , , , , , , ,

defer (day)

: ENdays
  ['] (ENday) is (day) ;

: FRdays
  ['] (FRday) is (day) ;

3 value dayLength
: .day
  (day)
  swap cell *
  + @ execute
  dayLength ?dup if
    min
  then
  type
;
ENdays
0 .day \ display Sun
1 .day \ display Mon
2 .day \ display Tue
FRdays ok
0 .day \ display Dim
1 .day \ display Lun
2 .day \ display Mar

```

; --

Mot d'exécution immédiate terminant habituellement la compilation d'une définition "deux-points".

```

: NAME
  nomex1 nomex2
  nomexn ;

```

< **n1 n2 -- fl**

Laisse fl vrai si $n1 < n2$

```
4 10 <= \ leave -1 on stack
4 4 <= \ leave 0 on stack
4 3 <= \ leave 0 on stack
```

<# **n --**

Marque le début de la conversion d'un nombre entier en chaîne de caractères.

```
\ display address in format: NNNN-NNNN
: DUMPaddr ( n -- )
  <# # # # # [char] - hold # # # # #>
  type
;

\ display byte in format: NN
: DUMPbyte ( c -- )
  <# # # #>
  type
;
```

<= **n1 n2 -- fl**

Laisse fl vrai si $n1 \leq n2$

```
4 10 <= \ leave -1 on stack
4 4 <= \ leave -1 on stack
4 3 <= \ leave 0 on stack
```

<> **x1 x2 -- fl**

Teste si l'entier simple précision x1 n'est pas égal à x2.

```
5 5 <> \ push FALSE on stack
5 4 <> \ push TRUE on stack
```

= **n1 n2 -- fl**

Laisse fl vrai si $n1 = n2$

```
4 10 = \ leave 0 on stack
4 4 = \ leave -1 on stack
```

> **x1 x2 -- fl**

Teste si x1 est supérieur à x2.

>= **x1 x2 -- fl**

Teste si l'entier simple précision x1 est égal à x2.

```
5 5 >=      \ push  FALSE on stack
5 4 >=      \ push  TRUE  on stack
```

>body **cfa -- pfa**

convertit l'adresse cfa en adresse pfa (Parameter Fields Address)

>flags **xt -- flags**

Convertit l'adresse cfa en adresse des flags.

>in **-- addr**

Nombre de caractères consommés depuis TIB

```
tib >in @ type
\ display:
tib >in @
```

>link **cfa -- cfa2**

Convertit l'adresse cfa du mot courant en adresse cfa du mot précédemment défini dans le dictionnaire.

```
' dup >link    \ get cfa from word defined before dup
>name type     \ display "XOR"
```

>link& **cfa -- lfa**

Transforme l'adresse d'exécution du mot courant en adresse de lien de ce mot. Cette adresse de lien pointe vers le cfa du mot défini avant ce mot.

Utilisé par **>link**

>name **cfa -- nfa len**

trouve l'adresse du champ de nom d'un mot à partir de son adresse de champ de code cfa.

```
' words      \ push cfa of 'words' on stack
>name        \ convert cfa of 'words' in nfa field followed by len
type         \ display 'words'
```

>name-length **cfa -- n**

Transforme une adresse cfa en longueur du nom du mot de cette adresse cfa. Mot utilisé par **vlist**

>r S: n -- R: n

Transfère n vers la pile de retour.

Cette opération doit toujours être équilibrée avec **r>**

```
\ display n in binary format
: b. ( n -- )
  base @ >r
  binary .
  r> base !
;
```

? addr -- c

Affiche le contenu d'une variable ou d'une adresse quelconque.

?do n1 n2 --

Exécute une boucle **do loop** ou **do +loop** si n1 est strictement supérieur à n2.

```
DECIMAL
: qd ?DO I LOOP ;
  789 789 qd \
-9876 -9876 qd \
  5 0 qd \ display: 0 1 2 3 4
```

?dup n -- n | n n

Duplique n si n n'est pas nul.

@ addr -- n

Récupère la valeur entière n stockée à l'adresse addr.

```
TEMPERATURE @
```

abort --

Génère une exception et interrompt l'exécution du mot et rend la main à l'interpréteur.

abort" comp: --

Affiche un message d'erreur et interrompt toute exécution FORTH en cours.

```
: abort-test
  if
    abort" stop program"
  then
    ." continue program"
;

0 abort-test \ display: continue program
1 abort-test \ display: stop program ERROR
```

abs **n -- n'**

Renvoie la valeur absolue de n.

```
-7 abs .        \ display 7
```

accept **addr n -- n**

Accepte n caractères depuis le clavier (port série) et les stocke dans la zone mémoire pointée par addr.

```
create myBuffer 100 allot
myBuffer 100 accept        \ on prompt, enter: This is an example
myBuffer swap type        \ display: This is an example
```

afliteral **r:r --**

Compile un nombre réel. Utilisé par **fliteral**

aft **--**

Saute à THEN dans une boucle FOR-AFT-THEN-NEXT lors de la première itération.

```
: test-aft1 ( n -- )
  FOR
    ." for "        \ first iteration
    AFT
    ." aft "        \ following iterations
    THEN
    I .             \ all iterations
  NEXT ;
3 test-aft1
\ display for 3 aft 2 aft 1 aft 0
```

again **--**

Marque la fin d'une boucle infinie de type **begin ... again**

```
: test ( -- )
  begin
    ." Diamonds are forever" cr
  again
;
```

align **--**

Aligne le pointeur du dictionnaire de la section de données actuelle sur la limite de la cellule.

aligned **addr1 -- addr2**

addr2 est la première adresse alignée plus grande ou égale à addr1.

allot **n** --

Réserve n adresses dans l'espace de données.

also --

Duplique le vocabulaire au sommet de la pile des vocabulaires.

analogRead **pin** -- **n**

Lecture analogique, intervalle 0-4095.

Utilisé pour lire la valeur analogique. **analogRead** n'a qu'un seul argument qui est un numéro de broche du canal analogique que vous souhaitez utiliser.

```
\ solar cell connected on pin G34
34 constant SOLAR_CELL

: init-solar-cell ( -- )
  SOLAR_CELL input pinMode
;

: solar-cell-read ( -- n )
  SOLAR_CELL analogRead
;
```

AND **n1 n2** --- **n3**

Effectue un ET logique.

Les mots **AND**, **OR** et **XOR** effectuent des opérations logiques binaires **bit à bit** sur les entiers simple précision situés au sommet de la pile de données.

```
0 0 and . \ display 0
0 -1 and . \ display 0
-1 0 and . \ display 0
-1 -1 and . \ display -1
```

ansi --

Sélectionne le vocabulaire **ansi**.

ARSHIFT **x1 u** -- **x2**

Décalage arithmétique à droite de u fois

asm --

Sélectionne le vocabulaire **asm**.

assembler --

Alias pour **asm**.

Sélectionne le vocabulaire **asm**.

assert **fl** **--**

Pour tests et assertions.

at-xy **x y** **--**

Positionne le curseur aux coordonnées x y.

```
: menu ( -- )
  page
  10 4 at-xy
  0 bg 7 fg ." Your choice, press: " normal
  12 5 at-xy ." A - accept"
  12 6 at-xy ." D - deny"
;
```

base **-- addr**

Variable simple précision déterminant la base numérique courante.

La variable **BASE** contient la valeur 10 (décimal) au démarrage de FORTH.

```
DECIMAL      \ select decimal base
2 BASE !     \ select binary base

\ other example
: GN2 \ ( -- 16 10 )
  BASE @ >R HEX BASE @ DECIMAL BASE @ R> BASE !
;
```

begin **--**

Marque le début d'une structure **begin..until**, **begin..again** ou **begin..while..repeat**

```
: endless ( -- )
  0
  begin
    dup . 1+
  again
;
```

bg **color[0..255]** **--**

Sélectionne la couleur d'affichage en arrière plan. La couleur est dans l'intervalle 0..255 en décimal.

```
: testBG ( -- )
  normal
  256 0 do
    i bg ." X"
  loop ;
```

BIN **mode -- mode'**

Modifie une méthode d'accès au fichier pour inclure BINARY.

BINARY **--**

Sélectionne la base numérique binaire.

```
255 BINARY . \ display 11111111
DECIMAL      \ return to decimal base
```

bl **-- 32**

Dépose 32 sur la pile de données.

```
\ definition of bl
: bl ( -- 32 )
  32
;
```

blank **addr len --**

Si len est supérieur à zéro, range un caractère de code \$20 (espace) dans toute la zone de longueur len à l'adresse mémoire commençant à addr.

block **n -- addr**

Récupère l'adresse d'un bloc n de 1024 octets.

block-fid **-- n**

Flag indiquant l'état d'un fichier de blocs.

block-id **-- n**

Pointeur vers un fichier de blocs.

buffer **n - addr**

Obtient un bloc de 1024 octets sans tenir compte de l'ancien contenu.

bye **--**

Mot défini par **defer**.

Exécute par défaut **esp32-by** (dans voc. internals).

c! **c addr --**

Stocke une valeur 8 bits c à l'adresse addr.

```
36 constant DDRB \ data direction register for PORT B on Arduino
```



```
32 DDRB c! \ same as 35 32 c!
```

c, **c --**

Ajoute c à la section de données actuelle.

```
create myDatas
    36 c, 42 c, 24 c, 12 c,
myDatas 1+ c@ \ push 42 on stack
```

c@ **addr -- c**

Récupère la valeur 8 bits c stockée à l'adresse addr.

```
35 constant PINB \ adresse registre données PIN de PORT B sur Arduino
PINB c@ \ empile contenu registre pointé par PINB
```

camera-server **--**

Sélectionne le vocabulaire **camera-server**.

CASE **--**

Marque le début d'une structure **CASE OF ENDOF ENDCASE**

```
: day ( n -- addr len )
    CASE
        0 OF s" Sunday"      ENDOF
        1 OF s" Monday"      ENDOF
        2 OF s" Tuesday"     ENDOF
        3 OF s" Wednesday"   ENDOF
        4 OF s" Thursday"    ENDOF
        5 OF s" Friday"      ENDOF
        6 OF s" Saturday"    ENDOF
    ENDCASE
;
```

cat **-- <path>**

Affiche le contenu du fichier.

```
cat /spiffs/dumpTool.txt
\ display content of file dumpTool.txt
\ if this file was edited and saved in /spiffs/ file system
```

catch **cfa -- fl**

Initialise une action à réaliser en cas d'exception déclenchée par **throw**.

cell **-- 4**

Retourne le nombre d'octets pour un entier 32 bits.

cell+ **n -- n'**

Incrémente contenu de **CELL**.

cell/ **n -- n'**

Divise contenu de **CELL**.

cells **n -- n'**

Multiplie contenu de **CELL**.

Permet de se positionner dans un tableau d'entiers.

```
create table ( -- addr)
  1 , 5 , 10 , 50 , 100 , 500 ,
\ get values indexed 0 and 3 from table
table 0 cells + @ . \ display 1
table 3 cells + @ . \ display 50
```

char **-- <string>**

Mot utilisable en interprétation seulement.

Empile le premier caractère de la chaîne qui suit ce mot.

```
char v . \ display: 118 (ascii code for "v")
char house . \ display: 104 - code for "h"
```

CLOSE-FILE **fileid -- ior**

Ferme un fichier ouvert.

cmove **c-addr1 c-addr2 u --**

Si u est supérieur à zéro, copier u caractères consécutifs de l'espace de données commençant à c-addr1 vers celui commençant à c-addr2, en procédant caractère par caractère des adresses inférieures aux adresses supérieures.

code **-- <:name>**

Définit un mot dont la définition est écrite en assembleur.

```
code my2*
  a1 32 ENTRY,
  a8 a2 0 L32I.N,
  a8 a8 1 SLLI,
  a8 a2 0 S32I.N,
  RETW.N,
end-code
```


CREATE-FILE **a n mode -- fh ior**

Crée un fichier sur le disque, renvoyant un 0 ior en cas de succès et un identifiant de fichier.

current **-- cfa**

Pointeur vers le pointeur du dernier mot du vocabulaire actuel

```
: test ( -- )
  ." only for test" ;
current @ @ >name type   \ display test
```

DECIMAL **--**

Sélectionne la base numérique décimale. C'est la base numérique par défaut au démarrage de FORTH.

```
HEX
FF DECIMAL .   \ display 255
```

default-key **-- c**

Execute **serial-key**.

default-key? **-- fl**

Execute **serial-key?**.

default-type **addr len --**

Execute **serial-type**.

defer **-- <vec-name>**

Définit un vecteur d'exécution différée.

vec-name exécute le mot dont le code d'exécution est stocké dans l'espace de données de vec-name.

```
defer xEmit
: vxEmit ( c ---)
  1+ emit ;
' vxEmit is xEmit
```

DEFINED? **-- <word>**

Renvoie une valeur non nulle si le mot est défini.

```
DEFINED FORGET      \ push non null value on stack
DEFINED LotusBlue   \ push 0 value on stack if LotusBlue don't defined
```

```
\ other example:
DEFINED? --DAout [if] forget --DAout [then]
create --DAout
```

definitions --

Rend courant le premier vocabulaire de contexte. Tout mot compilé est chaîné à un vocabulaire de contexte. Initialement, ce vocabulaire est **FORTH**

```
VOCABULARY LOGO          \ create vocabulary LOGO
LOGO DEFINITIONS        \ will set LOGO context vocabulary
: EFFACE
  page ;                 \ create word EFFACE in LOGO vocabulary
```

DELETE-FILE a n -- ior

Supprime un fichier nommé du disque et renvoie ior=0 en cas de succès.

depth -- n

n est le nombre de valeurs de cellule unique contenues dans la pile de données avant que n ne soit placé sur la pile.

```
\ test this after reset:
depth          \ leave 0 on stack
10 32 25
depth          \ leave 3 on stack
```

digitalWrite pin value --

Défini l'état du pin GPIO.

```
17 constant TRIGGER_ON    \ green LED
16 constant TRIGGER_OFF   \ red LED

: init-trigger-state ( -- )
  TRIGGER_ON output pinMode
  TRIGGER_OFF output pinMode
;

TRIGGER_ON HIGH digitalWrite
```

do n1 n2 --

Configure les paramètres de contrôle de boucle avec l'index n2 et la limite n1.

```
: testLoop
  256 32 do
    I emit
  loop
;
```

DOES> **comp:** -- | **exec:** -- **addr**

Le mot **CREATE** peut être utilisé dans un nouveau mot de création de mots...

Associé à **DOES>**, on peut définir des mots qui disent comment un mot est créé puis exécuté.

drop **n** --

Enlève du sommet de la pile de données le nombre entier simple précision qui s'y trouvait.

```
2 5 8 drop \ leave 2 and 5 on stack
```

dump **a n** --

Visualise une zone mémoire.

Cette version est peu intéressante. Préférez cette version:

[DUMP tool for ESP32Forth](#)

dump-file **addr len addr2 len2** --

Transfère le contenu d'une chaîne texte **addr len** vers le fichier pointé par **addr2 len2**

Le contenu du fichier */spiffs/autoexec.fs* est automatiquement interprété et/ou compilé au démarrage de ESP32Forth.

Cette fonctionnalité peut être exploitée pour paramétrer l'accès WiFi au démarrage de ESP32Forth en injectant les paramètres d'accès comme ceci:

```
r| z" NETWORK-NAME" z" PASSWORD" webui |  
s" /spiffs/autoexec.fs"  
dump-file
```

dup **n** -- **n n**

Duplique le nombre entier simple précision situé au sommet de la pile de données.

```
: SQUARE ( n --- nE2)  
  DUP * ;  
5 SQUARE . \ display 25  
10 SQUARE . \ display 100
```

echo -- **addr**

Variable. Contient -1 par défaut. Si 0, les commandes ne sont pas affichées.

```
: serial2-type ( a n -- )  
  Serial2.write drop ;  
  
: typeToLoRa ( -- )  
  0 echo ! \ disable display echo from terminal
```

```

    ['] serial2-type is type
;

: typeToTerm ( -- )
    ['] default-type is type
    -1 echo !    \ enable display echo from terminal
;

```

editor --

Sélectionne le vocabulaire **editor**.

- **l** liste le contenu du bloc courant
- **n** sélectionne le bloc suivant
- **p** sélectionne le bloc précédent
- **wipe** vide le contenu du bloc courant
- **d** efface la ligne n. Le numéro de ligne doit être dans l'intervalle 0..14. Les lignes qui suivent remontent vers le haut.

Exemple: **3 D** efface le contenu de la ligne 3 et fait remonter le contenu des lignes 4 à 15.

- **e** efface le contenu de la ligne n. Le numéro de ligne doit être dans l'intervalle 0..15. Les autres lignes ne remontent pas.
- **a** insère une ligne n. Le numéro de ligne doit être dans l'intervalle 0..14. Les lignes situées après la ligne insérées redescendent.

Exemple: **3 A test** insère **test** à la ligne 3 et fait descendre le contenu des lignes 4 à 15.

- **r** remplace le contenu de la ligne n.

Exemple: **3 R test** remplace le contenu de la ligne 3 par **test**

else --

Mot d'exécution immédiate et utilisé en compilation seulement. Marque une alternative dans une structure de contrôle du type **IF ... ELSE ... THEN**

```

: TEST ( ---)
  CR ." Press a key " KEY
  DUP 65 122 BETWEEN
  IF
    CR 3 SPACES ." is a letter "
  ELSE
    DUP 48 57 BETWEEN
    IF
      CR 3 SPACES ." is a digit "

```

```

ELSE
    CR 3 SPACES ." is a special character "
THEN
THEN
DROP ;

```

emit x --

Si x est un caractère graphique dans le jeu de caractères défini par l'implémentation, affiche x.

L'effet d'**EMIT** pour toutes les autres valeurs de x est défini par l'implémentation.

Lors du passage d'un caractère dont les bits de définition de caractère ont une valeur comprise entre hex 20 et 7E inclus, le caractère standard correspondant s'affiche. Étant donné que différents périphériques de sortie peuvent répondre différemment aux caractères de contrôle, les programmes qui utilisent des caractères de contrôle pour exécuter des fonctions spécifiques ont une dépendance environnementale. Chaque **EMIT** ne traite qu'avec un seul caractère.

```

65 emit \ display A
66 emit \ display B

```

empty-buffers --

Vide tous les tampons.

ENDCASE --

Marque la fin d'une structure **CASE OF ENDOF ENDCASE**

```

: day ( n -- addr len )
  CASE
    0 OF s" Sunday"      ENDOF
    1 OF s" Monday"      ENDOF
    2 OF s" Tuesday"     ENDOF
    3 OF s" Wednesday"   ENDOF
    4 OF s" Thursday"    ENDOF
    5 OF s" Friday"      ENDOF
    6 OF s" Saturday"    ENDOF
  ENDCASE
;

```

ENDOF --

Marque la fin d'un choix **OF .. ENDOF** dans la structure de contrôle entre **CASE ENDCASE**.

```

: day ( n -- addr len )
  CASE
    0 OF s" Sunday"      ENDOF
    1 OF s" Monday"      ENDOF
    2 OF s" Tuesday"     ENDOF

```



```

3 OF s" Wednesday"  ENDOF
4 OF s" Thursday"    ENDOF
5 OF s" Friday"      ENDOF
6 OF s" Saturday"    ENDOF
ENDCASE
;

```

erase **addr len --**

Si len est supérieur à zéro, range un caractère de code \$00 dans toute la zone de longueur len à l'adresse mémoire commençant à addr.

ESP32-C3? **-- -1|0**

Empile -1 si la carte est ESP32-C3.

ESP32-S2? **-- -1|0**

Empile -1 si la carte est ESP32-S2.

ESP32-S3? **-- -1|0**

Empile -1 si la carte est ESP32-S3.

ESP32? **-- -1|0**

Empile -1 si la carte est ESP32.

evaluate **addr len --**

Évalue le contenu d'une chaîne de caractères.

```

s" words"
evaluate  \ execute the content of the string, here: words

```

EXECUTE **addr --**

Exécute le mot pointé par addr.

Prenez l'adresse d'exécution de la pile de données et exécute ce jeton. Ce mot puissant vous permet d'exécuter n'importe quel jeton qui ne fait pas partie d'une liste de jetons.

exit **--**

Interrompt l'exécution d'un mot et rend la main au mot appelant.

Utilisation typique: **: X ... test IF ... EXIT THEN ... ;**

En exécution, le mot **EXIT** aura le même effet que le mot **;**

extract **n base -- n c**

Extrait le digit de poids faible de n. Laisse sur la pile le quotient de n/base et le caractère ASCII de ce digit.

F* **r1 r2 -- r3**

Multiplication de deux nombres réels.

```
1.35e 2.2e F*
F. \ display 2.969999
```

F** **r_val r_exp -- r**

Elève un réel r_val à la puissance r_exp.

```
2e 3e f** f. \ display 8.000000
2e 4e f** f. \ display 16.000000
10e 1.5e f** f. \ display 31.622776
```

F+ **r1 r2 -- r3**

Addition de deux nombres réels.

```
3.75e 5.21e F+
F. \ display 8.960000
```

F- **r1 r2 -- r3**

Soustraction de deux nombres réels.

```
10.02e 5.35e F-
F. \ display 4.670000
```

f. **r --**

Affiche un nombre réel. Le nombre réel doit venir de la pile des réels.

```
pi f. \ display 3.141592
```

f.s **--**

Affiche le contenu de la pile des réels.

```
2.35e
36.512e
f.s \ display: <2> 2.350000 36.511996
```

F/ **r1 r2 -- r3**

Division de deux nombres réels.

```
22e 7e F/    \ PI approximation
F.           \ display 3.142857
```

F0< **r -- fl**

Teste si un nombre réel est inférieur à zéro.

```
5e F0<      \ leave 0 on stack
-3e F0<     \ leave -1 on stack
```

F0= **r -- fl**

Indique vrai si le réel est nul.

```
3e 3e F- F0= . \ display -1
```

f< **r1 r2 -- fl**

fl est vrai si $r1 < r2$.

```
3.2e 5.25e f<
. \ display -1
```

f<= **r1 r2 -- fl**

fl est vrai si $r1 \leq r2$.

```
3.2e 5.25e f<=
. \ display -1
5.25e 5.25e f<=
. \ display -1
8.3e 5.25e f<=
. \ display 0
```

f<> **r1 r2 -- fl**

fl est vrai si $r1 \neq r2$.

```
3.2e 5.25e f<>
. \ display -1
5.25e 5.25e f<>
. \ display 0
```

f= **r1 r2 -- fl**

fl est vrai si $r1 = r2$.

```
3.2e 5.25e f=
. \ display 0
5.25e 5.25e f=
. \ display -1
```

f> **r1 r2 -- fl**

fl est vrai si $r1 > r2$.

```
3.2e 5.25e f>
. \ display 0
```

f>= **r1 r2 -- fl**

fl est vrai si $r1 \geq r2$.

```
3.2e 5.25e f>=
. \ display 0
5.25e 5.25e f>=
. \ display -1
8.3e 5.25e f>=
. \ display -1
```

F>S **r -- n**

Convertit un réel en entier. Laisse sur la pile de données la partie entière si le réel a des parties décimales.

```
3.5e F>S . \ display 3
```

FABS **r1 -- r1'**

Délivre la valeur absolue d'un nombre réel.

```
-2e FABS F. \ display 2.000000
```

FATAN2 **r-tan -- r-rad**

Calcule l'angle en radian à partir de la tangente.

```
0.5e fatan2 f. \ display 1.325917
1e fatan2 f. \ display 0.785398
```

fconstant **comp: r -- <name> | exec: -- r**

Définit une constante de type réel.

```
9.80665e fconstant g \ gravitation constant on Earth
g f. \ display 9.806649
```

FCOS **r1 -- r2**

Calcule le cosinus d'un angle exprimé en radians.

```
pi 2e f/ \ calc angle 90 deg
FCOS F. \ display 0.000000
```

fdepth -- n

n est le nombre de réels dans la pile de réels.

FDROP r1 --

Enlève le nombre réel r1 du sommet de la pile des réels.

FDUP r1 -- r1 r1

Duplique le nombre réel r1 du sommet de la pile des réels.

FEXP ln-r -- r

Calcule le réel correspondant à e EXP r

```
4.605170e FEXP F.      \ display 100.000018
```

fg color[0..255] --

Sélectionne la couleur d'affichage du texte. La couleur est dans l'intervalle 0..255 en décimal.

```
: testFG ( -- )
  256 0 do
    i fg ." X"
  loop ;
```

file-exists? addr len --

Teste si un fichier existe. Le fichier est désigné par une chaîne de caractères.

```
s" /spiffs/dumpTool.txt" file-exists?
```

FILE-POSITION fileid -- ud ior

Renvoie la position du fichier et renvoie ior=0 en cas de succès

FILE-SIZE fileid -- ud ior

Récupère la taille en octets d'un fichier ouvert sous la forme d'un nombre double et renvoie ior=0 en cas de succès.

fill addr len c --

Si len est supérieur à zéro, range c dans toute la zone de longueur len à l'adresse mémoire commençant à addr.

FIND addr len -- xt | 0

cherche un mot dans le dictionnaire.

```
32 string t$
s" vlist" t$ $!
t$ find \ push cfa of VLIST on stack
```

fliteral **r:r --**

Mot d'exécution immédiate. Compile un nombre réel.

FLN **r -- ln-r**

Calcule le logarithme naturel d'un nombre réel.

```
100e FLN f. \ display 4.605170
```

FLOOR **r1 -- r2**

Arrondi un réel à la valeur entière inférieure.

```
45.67e FLOOR F. \ display 45.000000
```

flush **--**

Enregistre et vide tous les tampons.

Après édition du contenu d'un fichier bloc, exécutez **flush** garantit que les modification du contenu des blocs sont sauvegardées.

FLUSH-FILE **fileid -- ior**

Essayez de forcer l'écriture de toute information mise en mémoire tampon dans le fichier référencé par fileid vers le stockage de masse. Si l'opération réussit, ior vaut zéro.

FMAX **r1 r2 -- r1|r2**

Laisse le plus grand réel de r1 ou r2.

```
3e 4e FMAX F. \ display 4.000000
```

FMIN **r1 r2 -- r1|r2**

Laisse le plus petit réel de r1 ou r2.

```
3e 4e FMIN F. \ display 3.000000
```

FNEGATE **r1 -- r1'**

Inverse le signe d'un nombre réel.

```
5e FNEGATE f. \ display -5.000000
-7e FNEGATE f. \ display 7.000000
```

FNIP **r1 r2 -- r2**

Supprime second élément sur la pile des réels.

```
2.5e 4.32e
fnip
f.s \ display: <1> 4.320000
```

for **n --**

Marque le début d'une boucle **for .. next**

ATTENTION: l'index de boucle sera traité dans l'intervalle [n..0], soit n+1 itérations, ce qui est contraire aux autres versions du langage FORTH implémentant FOR..NEXT (FlashForth).

```
: myLoop ( ---)
  10 for
    r@ . cr \ display loop index
  next
;
```

forget **-- <name>**

Cherche dans le dictionnaire le mot qui suit. Si c'est un mot valide, supprime tous les mots définis jusqu'à ce mot. Affiche un message d'erreur si ce n'est pas un mot valide.

forth **--**

Sélectionne le vocabulaire **FORTH** dans l'ordre de recherche des mots pour exécuter ou compiler des mots.

forth-builtins **-- cfa**

Point d'entrée du vocabulaire **forth**.

FOVER **r1 r2 -- r1 r2 r1**

Duplique le second réel sur la pile des réels.

```
2.6e 3.4e fover
f.s \ display <3> 2.600000 3.400000 2.600000
```

fp0 **-- addr**

pointe vers le bas de la pile des réels de ESP32Forth (pile de données).

FP@ **-- addr**

Récupère l'adresse du pointeur de pile des réels.

freq **chan freq --**

définit la fréquence freq sur le canal chan.

Utilise **ledcWriteTone**

FSIN **r1 -- r2**

Calcule le sinus d'un angle exprimé en radians.

```
pi 2e f/      \ calc angle 90° deg
FSIN F.       \ display 1.000000
```

FSINCOS **r1 -- rcos rsin**

Calcule le cosinus et le sinus d'un angle exprimé en radians.

```
pi 4e f/
FSINCOS f. f.  \ display 0.707106 0.707106
pi 2e f/
FSINCOS f. f.  \ display 0.000000 1.000000
```

fsqrt **r1 -- r2**

Racine carrée d'un nombre réel.

```
64e fsqrt
F.       \ display 8.000000
```

FSWAP **r1 r2 -- r1 r2**

Inverse l'ordre des deux valeurs sur la pile des réels de ESP32Forth.

```
3.75e 5.21e FSWAP
F.  \ display 3.750000
F.  \ display 5.210000
```

fvariable **comp: -- <name> | exec: -- addr**

Définit une variable de type flottant.

```
fvariable arc
pi 0.5e F* \ angle 90° in radian -- PI/2
arc SF!
arc SF@ f. \ display 1.570796
```

graphics **--**

sélectionne le vocabulaire **graphics**.

handler -- addr

Ticket pour les interruptions.

here -- addr

Restitue l'adresse courante du pointeur de dictionnaire.

Le pointeur de dictionnaire s'incrémente au fur et à mesure de la compilation de mots et définition des variables et tableaux de données.

```
here u.      \ display 1073709120
: null ;
here u.      \ display 1073709144
```

HEX --

Sélectionne la base numérique hexadécimale.

```
255 HEX .    \ display FF
DECIMAL      \ return to decimal base
```

hld -- addr

Pointeur vers le tampon de texte pour la sortie numérique.

hold c --

Insère le code ASCII d'un caractère ASCII dans la chaîne de caractères initiée par <#.

i -- n

n est une copie de l'index de boucle actuel.

```
: mySingleLoop ( -- )
  cr
  10 0 do
    i .
  loop
;
mySingleLoop
\ display 0 1 2 3 4 5 6 7 8 9
```

if fl --

Le mot **IF** est d'exécution immédiate.

IF marque le début d'une structure de contrôle de type **IF . . THEN** ou **IF . . ELSE . . THEN**.

Lors de l'exécution, la partie de définition située entre **IF** et **THEN** ou entre **IF** et **ELSE** est exécutée si le flag booléen situé au sommet de la pile de données est vrai (f<>0).

Dans le cas contraire, si le flag booléen est faux (f=0), c'est la partie de définition située entre **ELSE** et **THEN** qui sera exécutée. S'il n'y a pas de **ELSE**, l'exécution se poursuit après **THEN**.

```
: WEATHER? ( f1 ---)
  IF
    ." Nice weather "
  ELSE
    ." Bad weather "
  THEN ;
1 WEATHER?      \ display: Nice weather
0 WEATHER?      \ display: Bad weather
```

immediate --

Rend la définition la plus récente comme mot immédiat.

Définit le bit de lexique de compilation uniquement dans le champ de nom du nouveau mot compilé. Lorsque l'interpréteur rencontre un mot avec ce bit défini, il ne l'exécutera pas, mais transmet un message d'erreur. Ce bit empêche l'exécution des mots de structure en dehors d'une définition de mot.

include -- <:name>

Charge le contenu d'un fichier désigné par <name>.

Le mot **include** n'est utilisable que depuis le terminal.

Pour charger le contenu d'un fichier depuis un autre fichier, utiliser le mot **included**.

```
include /spiffs/dumpTool.txt
\ load content of dump.txt

\ to include a file from an other file, use included
s" /spiffs/dumpTool.txt" included
```

included addr len --

Charge le contenu d'un fichier depuis le système de fichiers SPIFFS, désigné par une chaîne de caractères.

Le mot **included** peut être utilisé dans un listing FORTH stocké dans le système de fichiers SPIFFS.

Pour cette raison, le nom de fichier à charger doit toujours être précédé de */spiffs/*

```
s" /spiffs/dumpTool.txt" included
```

included? addr len -- f

Teste si le fichier désigné dans la chaîne de caractères a déjà été compilé.

INPUT -- 1

Constante. Valeur 1. Définit le sens d'utilisation d'un registre GPIO comme entrée.

internalized --

sélectionne le vocabulaire **internalized**.

internals --

Sélectionne le vocabulaire **internals**.

invert x1 -- x2

Complément à un de x1. Agit sur 16 ou 32 bits selon les versions FORTH.

```
1 invert . \ display -2
```

is --

Assigns the execution code of a word to a vectorized execution word.

```
defer xEmit
: vxEmit ( c ---)
  1+ emit ;
' vxEmit is xEmit
```

j -- n

n est une copie de l'index de boucle externe suivant.

```
: myDoubleLoop ( -- )
  cr
  10 0 do
    cr
    10 0 do
      i 1+ j 1+ * .
    loop
  loop
;
myDoubleLoop
\ display:
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
```

k -- **n**

n est la copie en 3ème niveau dans une boucle do do..loop.

```
: myTripleLoop ( -- )
  cr
  5 0 do
    cr
    5 0 do
      cr
      5 0 do
        i 1+ j 1+ k 1+ * * .
      loop
    loop
  loop
;
myTripleLoop
```

key -- **char**

Attend l'appui sur une touche. L'appui sur une touche renvoie son code ASCII.

```
key . \ display 97 if key "a" is active
key . \ affiche 65 if key "A" is active
```

key? -- **fl**

Renvoie *vrai* si une touche est appuyée.

```
: keyLoop
  begin
  key? until
;
```

L! **n addr** --

Enregistre une valeur n.

```
hex
3ff44004 constant GPIO_OUT_REG

: led-off ( -- )
  0 GPIO_OUT_REG 1!
;

: led-on ( -- )
  4 GPIO_OUT_REG 1!
;
```

latestxt -- **xt**

Empile l'adresse du code d'exécution (cfa) du dernier mot compilé.

```
: txtxtx ;
latest
```

```
>name type \ display txttxtx
```

leave --

Termine prématurément l'action d'une boucle **do..loop**.

```
256 string LoRaRX
s" +RCV=55,27,this is a transmission test,-36,40" LoRaRX $!

: scan$ ( char addr len -- )
  0 do
    2dup i + c@ = if
      i cr .
      leave
    then
  loop
  2drop
;

char , LoRaRX scan$
```

LED -- 2

Valeur pin 2 pour LED sur la carte. Ne fonctionne pas avec toutes les cartes.

list n --

Affiche le contenu du bloc n.

literal x --

Compile la valeur x comme valeur littérale.

```
: valueReg ( --- n)
  [ 36 2 * ] literal ;

\ equivalent to:
: valueReg ( --- n)
  72 ;
```

load n --

Charge et interprète le contenu d'un bloc.

load précédé du numéro du bloc que vous souhaitez exécuter et/ou compiler le contenu.

Pour compiler le contenu de notre bloc 0, nous allons exécuter **0 load**

loop --

Ajoute un à l'index de la boucle. Si l'index de boucle est alors égal à la limite de boucle, supprime les paramètres de boucle et poursuit l'exécution immédiatement après la boucle. Sinon, continue l'exécution au début de la boucle.

ls -- "path"

Affiche le contenu d'un chemin de fichiers.

```
ls /spiffs/ \ display:  
dump.txt
```

LSHIFT x1 u -- x2

Décalage vers la gauche de u bits de la valeur x1.

```
8 2 lshift . \ display 32
```

max n1 n2 -- n1|n2

Laisse le plus grand non signé de u1 et u2.

MDNS.begin name-z -- fl

Démarre le DNS multidiffusion.

```
z" forth" MDNS.begin
```

min n1 n2 -- n1|n2

Laisse min de n1 et n2

mod n1 n2 -- n3

Divise n1 par n2, laisse le reste simple précision n3.

La fonction modulo peut servir à déterminer la divisibilité d'un nombre par un autre.

```
21 7 mod . \ display 0  
22 7 mod . \ display 1  
23 7 mod . \ display 2  
24 7 mod . \ display 3  
  
: DIV? ( n1 n2 ---)  
  OVER OVER MOD CR  
  IF  
    SWAP . ." is not "  
  ELSE  
    SWAP . ." is "  
  THEN  
    ." divisible by " .  
;
```

ms n --

Attente en millisencondes.

Pour les attentes longues, définir un mot d'attente en secondes.

```
500 ms \ delay for 1/2 second

: seconds ( n --)
  0
  for
    1000 ms
  next
;
12 seconds \ delay for 12 seconds
```

MS-TICKS -- n

Impulsions système. Une impulsion par milliseconde.

Utile pour mesurer le temps d'exécution d'une définition.

mv -- "src" "dest"

Renommez le fichier "src" en "dst".

n. n --

Affiche toute valeur n sous sa forme décimale.

negate n -- -n'

Le complément à deux de n.

```
5 negate . \ display -5
```

next --

Marque la fin d'une boucle **for .. next**

nip n1 n2 -- n2

Enlève n1 de la pile.

nl -- 10

Dépose 10 sur la pile de données.

normal --

Désactive les couleurs sélectionnées pour l'affichage.

OCTAL --

Sélectionne la base numérique octale.

```
255 OCTAL . \ display 377
DECIMAL \ return to decimal base
```

OF n --

Marque un choix **OF .. ENDOF** dans la structure de contrôle entre **CASE ENDCASE**

Si la valeur testée est égale à celle qui précède **OF**, la partie de code située entre **OF ENDOF** sera exécutée.

```
: day ( n -- addr len )
  CASE
    0 OF s" Sunday"      ENDOF
    1 OF s" Monday"      ENDOF
    2 OF s" Tuesday"     ENDOF
    3 OF s" Wednesday"   ENDOF
    4 OF s" Thursday"    ENDOF
    5 OF s" Friday"      ENDOF
    6 OF s" Saturday"    ENDOF
  ENDCASE
;
```

ok --

Affiche la version du langage FORTH ESP32forth.

```
ok
\ display: ESP32forth v7.0.6.10 - rev 17c8b34289028a5c731d
```

only --

Réinitialise la pile de contexte à un élément, le dictionnaire FORTH

Non standard, car il n'y a pas de vocabulaire ONLY distinct

open-blocks addr len --

Ouvre un fichier de blocs. Le fichier de blocs par défaut est *blocks.fb*

OPEN-FILE addr n opt -- n

Ouvre un fichier.

opt est une valeur parmi **R/O** ou **R/W** ou **W/O**.

```
s" myFile" r/o open-file
```

OR n1 n2 -- n3

Effectue un OU logique.

Les mots **AND**, **OR** et **XOR** effectuent des opérations logiques binaires **bit à bit** sur les entiers simple précision situés au sommet de la pile de données.

```
0 -1 or . \ display 0
0 -1 or . \ display -1
```



```
-1 0 or . \ display -1
-1 -1 or . \ display -1
```

order --

Affiche l'ordre de recherche de vocabulaire.

```
Serial
order \ display Serial
```

over **n1 n2** -- **n1 n2 n1**

Place une copie de n1 au sommet de la pile.

```
2 5 OVER \ duplicate 2 on top of the stack
```

page --

Efface l'écran.

PARSE **c "string"** -- **addr count**

Analyse le mot suivant dans le flux d'entrée, se terminant au caractère c. Laissez l'adresse et le nombre de caractères du mot. Si la zone d'analyse était vide, alors count=0.

pause --

Passe la main aux autres tâches.

PI -- **r**

Constante PI.

```
pi
F. \ display 3.141592
\ perimeter of a circle, for r = 5.2 --- P = 2 π R
5.2e 2e F* pi F*
F. \ display 32.672560
```

pinMode **pin mode** --

Sélectionne le mode d'utilisation de la borne GPIO

MODE = INPUT | OUTPUT

```
04 input pinmode \ G04 as an input
15 input pinmode \ G15 as an input
```

precision -- **n**

Pseudo constante déterminant la précision d'affichage des nombres réels.

Valeur initiale 6.

Si on réduit la précision d'affichage des nombres réels en dessous de 6, les calculs seront quand même réalisés avec une précision à 6 décimales.

```
precision . \ display 6
pi f.      \ display 3.141592
4 set-precision
precision . \ display 4
pi f.      \ display 3.1415
```

prompt --

Affiche un texte de disponibilité de l'interpréteur. Affiche par défaut:

ok

PSRAM? -- -1|0

Empile -1 si la mémoire PSRAM est disponible.

r" **comp:** -- <string> | **exec:** addr len

Crée une chaîne temporaire terminée par "

R/O -- 0

Constante système. Empile 0.

R/W -- 2

Constante système. Empile 2.

r> **R: n** -- **S: n**

Transfère n depuis la pile de retour.

Cette opération doit toujours être équilibrée avec **>r**

```
\ display n in binary format
: b. ( n -- )
  base @ >r
  binary .
  r> base !
;
```

R@ -- n

Copie sur la pile de données le contenu du sommet de la pile de retour.

rdrop **S:** -- **R: n** --

Jete l'élément supérieur de la pile de retour.

READ-FILE **a n fh -- n ior**

Lit les données d'un fichier. Le nombre de caractères réellement lus est renvoyé sous la forme u2, et ior est renvoyé 0 pour une lecture réussie.

recurse --

Ajoute un lien d'exécution correspondant à la définition actuelle.

L'exemple habituel est le codage de la fonction factorielle.

```
: FACTORIAL ( +n1 -- +n2)
  DUP 2 < IF DROP 1 EXIT THEN
  DUP 1- RECURSE *
;
```

remaining -- n

Indique l'espace restant pour vos définitions.

```
remaining .      \ display 76652
: t ;
remaining .      \ display 76632
```

remember --

Sauvegarde un instantané dans le fichier par défaut (./myforth or /spiffs/myforth on ESP32).

Le mot **REMEMBER** vous permet de *geler* le code compilé. Si vous avez compilé une application, exécutez **REMEMBER**. Débranchez la carte ESP32. Rebranchez-là. Vous devriez retrouver votre application.

Utilisez **STARTUP** : pour définir le mot de votre application à exécuter au démarrage.

repeat --

Achève une boucle indéfinie **begin.. while.. repeat**

REPOSITION-FILE **ud fileid -- ior**

Définir la position du fichier et renvoyer ior=0 en cas de succès

required **addr len --**

Charge le contenu du fichier désigné dans la chaîne de caractères s'il n'a pas déjà été chargé.

```
s" /spiffs/dumpTool.txt" required
```

reset --

Supprime le nom de fichier par défaut.

RESIZE-FILE **ud fileid -- ior**

Définit la taille du fichier par ud, un nombre double non signé. Après avoir utilisé **RESIZE-FILE**, le résultat renvoyé par **FILE-POSITION** peut être invalide

restore -- **<:name>**

Restaure un instantané à partir d'un fichier.

revive --

Restaure le nom de fichier par défaut.

rm -- **"path"**

Efface le fichier indiqué.

rot **n1 n2 n3 -- n2 n3 n1**

Rotation des trois valeurs au sommet de la pile.

rp0 -- **addr**

pointe vers le bas de la pile de retour de Forth (pile de données).

RSHIFT **x1 u -- x2**

Décalage vers la droite de u bits de la valeur x1.

```
64 2 rshift . \ display 16
```

r| **comp: -- <string> | exec: addr len**

Crée une chaîne temporaire terminée par |

s" **comp: -- <string> | exec: addr len**

En interprétation, laisse sur la pile de données la chaîne délimitée par "

En compilation, compile la chaîne délimitée par "

Lors de l'exécution du mot compilé, restitue l'adresse et la longueur de la chaîne...

```
\ header for DUMP
: headDump
  s" --addr----- 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F"
;
headDump \ push addr len on stack
```

```
headDump type \ display: --addr----- 00 01 02 03 04 05 06 07 08 09 0A 0B 0C
0D 0E 0F
```

S>F **n -- r: r**

Convertit un nombre entier en nombre réel et transfère ce réel sur la pile des réels.

```
35 S>F
F. \ display 35.000000
```

s>z **a n -- z**

Convertir une chaîne addr len en chaîne terminée par zéro.

save **-- <:name>**

Enregistre un instantané du dictionnaire actuel dans un fichier.

save-buffers **--**

Sauvegarde tous les tampons.

SCR **-- addr**

Variable pointant sur le bloc en cours d'édition.

see **-- name>**

Décompile une définition FORTH.

```
see include
: include bl PARSE included ;

see space
: space bl emit ;
```

set-precision **n --**

Modifie la précision d'affichage des nombres Réels.

```
pi f. \ display 3.141592
2 set-precision
pi f. \ display 3.14
```

SF! **r addr --**

Stocke un réel préalablement déposé sur la pile des réels à l'adresse mémoire addr.

sf, **r --**

Compile un nombre réel.

SF@ **addr -- r**

Récupère le nombre réel stocké à l'adresse **addr**, en général une variable définie par **fvariable**.

sfloat **-- 4**

Constante. Valeur 4.

sfloat+ **addr -- addr+4**

Incrémente une adresse mémoire de la longueur d'un réel.

sfloats **n -- n*4**

Calcule l'espace nécessaire pour **n** réels.

sp0 **-- addr**

pointe vers le bas de la pile de données de Forth (pile de données).

SP@ **-- addr**

Dépose l'adresse du pointeur de pile sur la pile.

```
\ return number cells used on stack
: stackSize ( -- n )
  SP@ SP0 - CELL/
;
```

space **--**

Affiche un caractère espace.

```
\ definition of space
: space ( -- )
  bl emit
;
```

spaces **n --**

Affiche **n** fois le caractère espace.

Défini depuis la version 7.071

SPI **--**

Sélectionne le vocabulaire **SPI**.

Liste des mots du vocabulaire **SPI**:

**SPI.begin SPI.end SPI.setHwCs SPI.setBitOrder SPI.setDataMode
SPI.setFrequency**

**SPI.setClockDivider SPI.getClockDivider SPI.transfer SPI.transfer8
SPI.transfer16
SPI.transfer32 SPI.transferBytes SPI.transferBits SPI.write SPI.write16
SPI.write32 SPI.writeBytes SPI.writePixels SPI.writePattern SPI-builtins**

startup: -- <name>

Indique le mot qui doit s'exécuter au démarrage de ESP32forth après initialisation de l'environnement général.

Ici on a défini le mot **myBoot** qui affiche un texte au démarrage.

Pour tester la bonne exécution, vous pouvez taper **bye**, ce qui redémarre ESP32forth.

Vous pouvez aussi débrancher la carte ESP32 et la rebrancher. C'est ce test qui a été effectué. Voici le résultat dans le terminal.

```
: myBoot ( -- )
  ." This is a text displayed from boot" ;
startup: myBoot

\ on restart:
--> This is a text displayed from bootESP32forth v7.0.5 - rev
33cf8aaa6fe3e0bc4a
bf3e4cd5c496a3071b9171
ok
ok
```

state -- fl

Etat de compilation. L'état ne peut être modifié que par [et].

-1 pour compilateur, 0 pour interpréteur

str n -- addr len

Transforme en chaîne alphanumérique toute valeur n, ce dans la base numérique courante.

str= addr1 len1 addr2 len2 -- fl

Compare deux chaînes de caractères. Empile vrai si elles sont identiques.

```
s" 123"    s" 124"
str = .    \ display 0
s" 156"    s" 156"
str= .     \ display -1
```

streams --

Sélectionne le vocabulaire **streams**.

structures --

Sélectionne le vocabulary **structures**.

swap n1 n2 -- n2 n1

Echange les valeurs situées au sommet de la pile.

```
2 5 SWAP
. \ display 2
. \ display 5
```

task comp: xt dsz rsz -- <name> | exec: -- task

Créer une nouvelle tâche avec taille dsz pour la pile de données et rsz pour la pile de retour.

```
tasks
: hi    begin ." Time is: " ms-ticks . cr 1000 ms again ;
' hi 100 100 task my-counter
my-counter start-task
```

tasks --

Sélectionne le vocabulaire **tasks**.

then --

Mot d'exécution immédiate utilisé en compilation seulement. Marque la fin d'une structure de contrôle de type **IF..THEN** ou **IF..ELSE..THEN**.

throw n --

Génère une erreur si n pas égal à zéro.

Si les bits de n ne sont pas nuls, extraie l'exception en tête de la pile d'exceptions, ainsi que tout ce qui se trouve sur la pile de retour au-dessus de ce cadre. Ensuite, restaure la spécification de la source d'entrée utilisée avant le CATCH correspondant et ajuste les profondeurs de toutes les piles définies par cette norme afin qu'elles soient identiques aux profondeurs enregistrées dans le cadre d'exception (i est le même nombre que le i dans les arguments d'entrée au CATCH correspondant), place n au-dessus de la pile de données et transfère le contrôle à un point juste après le CATCH qui a poussé ce cadre d'exception.

```
: could-fail ( -- char )
  KEY DUP [CHAR] Q = IF 1 THROW THEN ;

: do-it ( a b -- c ) 2DROP could-fail ;

: try-it ( --)
  1 2 ['] do-it CATCH IF
  ( x1 x2 ) 2DROP ." There was an exception" CR
  ELSE ." The character was " EMIT CR
```



```

    THEN
;

: retry-it ( -- )
    BEGIN 1 2 ['] do-it CATCH WHILE
    ( x1 x2) 2DROP ." Exception, keep trying" CR
    REPEAT ( char )
    ." The character was " EMIT CR
;

```

thru **n1 n2 --**

Charge le contenu d'un fichier de blocs, du bloc n1 au bloc n2.

tib **-- addr**

renvoie l'adresse du tampon d'entrée du terminal où la chaîne de texte d'entrée est conservée.

```

tib >in @ type
\ display:
tib >in @

```

to **n --- <valname>**

to affecte une nouvelle valeur à *valname*

tone **chan freq --**

définit la fréquence freq sur le canal chan.

Utilise **ledcWriteTone**

touch **-- "path"**

Créez un chemin de fichier "path" s'il n'existe pas.

type **addr c --**

Affiche la chaîne de caractères sur c octets.

```

: hello ( --- addr c)
s" Hello world" ;
hello type           \ display: Hello world
hello drop 5 type    \ display: Hello

```

u. **n --**

Dépile la valeur au sommet de la pile et l'affiche en tant qu'entier simple précision non signé.

```

1 U.      \ display 1

```

```
-1 U. \ display 65535
```

U/MOD **u1 u2 -- rem quot**

division int/int->int non signée.

UL@ **addr -- un**

Récupère une valeur non signée.

ATTENTION: les précédentes versions de ESP32forth utilisaient le mot **L@**.

unloop **--**

Arrête une action do..loop. Utiliser **unloop** avant **exit** seulement dans une structure do..loop.

```
: example ( -- )
  100 0 do
    cr i .
    key bl = if
      unloop exit
    then
  loop
;
```

until **fl --**

Ferme une structure **begin.. until**.

```
: myTestLoop ( -- )
  begin
    key dup .
    [char] A =
  until
;
myTestLoop \ end loop if key A pressed
```

update **--**

Utilisé pour l'édition de blocs. Force le bloc courant à l'état modifié.

use **-- <name>**

Utilise "name" comme fichier de blocs.

```
USE /spiffs/foo
```

used **-- n**

Indique l'espace pris par les définitions utilisateur. Ceci inclue les mots déjà définis du dictionnaire FORTH.

UW@ **addr** -- **un[2exp0..2exp16-1]**

Extrait la partie poids faible 16 bits d'une zone mémoire pointée par son adresse 32 bits non signée.

```
variable valX
hex 10204080 valX !
valX UW@ .      \ display 4080
valX 2 + UW@ .  \ display 1020
```

value **comp: n** -- **<valname>** | **exec: -- n**

Crée un mot de type *value*

valname empile la valeur.

Un mot défini par **value** est semblable à une constante, mais dont la valeur peut être modifiée.

```
12 value APPLES      \ Define APPLES with an initial value of 12
34 to APPLES          \ Change the value of APPLES. to is a parsing word
APPLES                \ puts 34 on the top of the stack
```

variable **comp: -- <name>** | **exec: -- addr**

Mot de création. Définit une variable simple précision.

```
variable speed
75 speed !      \ store 75 in speed
speed @ .      \ display 75
```

visual --

Sélectionne le vocabulaire **visual**.

vlist --

Affiche tous les mots d'un vocabulaire.

```
Serial vlist \ display content of Serial vocabulary
```

vocabulary **comp: -- <name>** | **exec: --**

Mot de définition d'un nouveau vocabulaire. En 83-STANDARD, les vocabulaires ne sont plus déclarés d'exécution immédiate.

```
\ create new vocabulary FPACK
VOCABULARY FPACK
```

W/O -- **1**

Constante système. Empile 1.

web --

Sélectionne le vocabulaire **web**.

while fl --

Marque la partie d'exécution conditionnelle d'une structure **begin..while..repeat**

```
\ logarithmus dualis of n1>0, rounded down to the next integer
: log2 ( +n1 -- n2 )
  2/ 0 begin
    over 0 >
  while
    1+ swap 2/ swap
  repeat
  nip
;
7 log2 .      \ display 2
100 log2 .    \ display 6
```

windows --

sélectionne le vocabulaire **windows**.

words --

Répertorie les noms de définition dans la première liste de mots de l'ordre de recherche.
Le format de l'affichage dépend de l'implémentation.

WRITE-FILE a n fh -- ior

Écrire un bloc de mémoire dans un fichier.

XOR n1 n2 -- n3

Effectue un OU eXclusif logique.

Les mots **AND**, **OR** et **XOR** effectuent des opérations logiques binaires **bit à bit** sur les entiers simple précision situés au sommet de la pile de données.

```
0 -1 xor .      \ display 0
0 -1  xor .      \ display -1
-1 0  xor .      \ display -1
-1 0  xor .      \ display 0
```

xtensa-assembler --

Charge et installe le vocabulaire **xtensa**.

Ce mot doit être exécuté une seule fois avant la définition de mots en assembleur XTENSA.

```
xtensa-assembler
```

```
code my2*
  a1 32 ENTRY,
  a8 a2 0 L32I.N,
  a8 a8 1 SLLI,
  a8 a2 0 S32I.N,
  RETW.N,
end-code
```

Xtensa? -- -1|0

Empile -1 si le processeur est XTENSA.

z" **comp:** -- <string> | **exec:** -- addr

Compile une chaîne terminée par valeur 0 dans la définition.

ATTENTION: ces chaînes de caractères marquées par **z"** ne sont à exploiter que pour des fonctions spécifiques, réseau par exemple.

```
z" mySSID"
z" myPASSWORD" Wifi.begin
```

z>s **z -- a n**

Convertit une chaîne terminée par zéro en chaîne addr len.

[--

Entre en mode interprétation. **[** est un mot d'exécution immédiate.

```
\ source for [
: [
  0 state !
; immediate
```

['] **comp:** -- <name> | **exec:** -- addr

Utilisable en compilation seulement. Exécution immédiate.

Compile le cfa de <name>

```
serial \ Select Serial vocabulary

: serial2-type ( a n -- )
  Serial2.write drop ;

: typeToLoRa ( -- )
  0 echo ! \ disable display echo from terminal
  ['] serial2-type is type
;

: typeToTerm ( -- )
  ['] default-type is type
```

```
-1 echo ! \ enable display echo from terminal
;
```

[char] **comp:** -- <spaces>name | **exec:** -- xchar

En compilation, enregistre le code ASCII du caractère indiqué après ce mot.

En exécution, le code xchar est déposé sur la pile de données.

```
: GC1 [CHAR] X      ;
: GC2 [CHAR] HELLO ;
GC1 \ empile 58
GC2 \ empile 48
```

[ELSE] --

Marque la partie de code d'une séquence **[IF]** ... **[ELSE]** ... **[THEN]**.

[IF] **fi** --

Commence une séquence conditionnelle de type **[IF]** ... **[ELSE]** ou **[IF]** ... **[ELSE]** ... **[THEN]**.

Si l'indicateur est 'TRUE', ne fait rien (et exécute donc les mots suivants normalement). Si l'indicateur est 'FALSE', analyse et supprime les mots de la zone d'analyse, y compris les instances imbriquées de **[IF]**.. **[ELSE]**.. **[THEN]** et **[IF]**.. **[THEN]** jusqu'à l'équilibrage **[ELSE]** ou **[THEN]** a été analysé et supprimé.

```
DEFINED? mclr invert [IF]
: mclr ( mask addr -- )
  dup >r c@ swap invert and r> c!
  ;
[THEN]
```

[THEN] --

Termine une séquence conditionnelle de type **[IF]** ... **[ELSE]** or **[IF]** ... **[ELSE]** ... **[THEN]**.

```
DEFINED? mclr [IF]
: mclr ( mask addr -- )
  dup >r c@ swap invert and r> c!
  ;
[THEN]
```

] --

Retour en mode compilation. **]** est un mot immédiat.

```
\ Load constant $1234 to top of stack
: a-number ( -- 1234 )
```

```
dup                \ Make space for new TOS value
[ R24 $34 ldi, ]
[ R25 $12 ldi, ]
;
```

{ -- <names..>

Marque le début de la définition de variables locales. Ces variables locales se comportent comme des pseudo-constantes.

Les variables locales sont une alternative intéressante à la manipulation des données de la pile. Elles rendent le code plus lisible.

```
: summ { n1 n2 }
  n1 n2 + . ;
3 5 summ \ display 8
```

graphics

Mots définis dans le vocabulaire **graphics**

```
flip poll wait window heart vertical-flip viewport scale translate }g g{
screen>g box color pressed? pixel height width event last-char last-key
mouse-y mouse-x RIGHT-BUTTON MIDDLE-BUTTON LEFT-BUTTON FINISHED TYPED RELEASED
PRESSED MOTION EXPOSED RESIZED IDLE internals
```

color -- **n**

Définit la couleur. Valeur par défaut: 0

```
\ Pen in red color:
$ff0000 to color \ $rrggbb
```

event -- **0**

Constante. Valeur par défaut 0

EXPOSED -- **2**

Constante. Valeur 2

FINISHED -- **7**

Constante. Valeur 7

height -- **0**

Value. Valeur par défaut 0

IDLE -- **0**

Constante. Valeur 0

last-char -- **0**

Constante. Valeur par défaut 0

last-key -- **0**

Constante. Valeur par défaut 0

LEFT-BUTTON -- **255**

Constante. Valeur 255

MIDDLE-BUTTON -- **254**

Constante. Valeur 254

MOTION -- 3

Constante. Valeur 3

mouse-x -- 0

Constante. Valeur par défaut 0

mouse-y -- 0

Constante. Valeur par défaut 0

pixel w h --

Trace un pixel en position w h

PRESSED -- 4

Constante. Valeur 4

RELEASED -- 5

Constante. Valeur 5

RESIZED -- 1

Constante. Valeur 1

RIGHT-BUTTON -- 253

Constante. Valeur 253

TYPED -- 6

Constante. Valeur 6

width -- 0

Value. Valeur par défaut 0

Mots FORTH par utilisation

arithmetic integer

* (n1 n2 -- n3)
*/ (n1 n2 n3 -- n4)
*/MOD (n1 n2 n3 -- n4 n5)
+ (n1 n2 -- n3)
- (n1 n2 -- n1-n2)
/mod (n1 n2 -- n3 n4)
1+ (n -- n+1)
1- (n -- n-1)
2* (n -- n*2)
2/ (n -- n/2)
4* (n -- n*4)
4/ (n -- n/4)
ARSHIFT (x1 u -- x2)
mod (n1 n2 -- n3)
negate (n -- -n')

FNEGATE (r1 -- r1')
FSIN (r1 -- r2)
FSINCOS (r1 -- rcos rsin)
fsqrt (r1 -- r2)
pi (-- r)
S>F (n -- r: r)

arithmetic real

#f+s (r:r)
1/F (r -- r')
F* (r1 r2 -- r3)
F** (r_val r_exp -- r)
F+ (r1 r2 -- r3)
F- (r1 r2 -- r3)
F/ (r1 r2 -- r3)
F0< (r -- fl)
F0= (r -- fl)
F>S (r -- n)
FABS (r1 -- r1')
FATAN2 (r-tan -- r-rad)
fconstant (comp: r -- <name> | exec: --
r)
FCOS (r1 -- r2)
FEXP (ln-r -- r)
FLN (r -- ln-r)
FLOOR (r1 -- r2)
FMAX (r1 r2 -- r1|r2)
FMIN (r1 r2 -- r1|r2)

block edit list

a (n --)
copy (from to --)
d (n --)
e (n --)
editor (--)
flush (--)
list (n --)
load (n --)
n (--)
open-blocks (addr len --)
p (--)
r (n --)
thru (n1 n2 --)
update (--)
use (-- <name>)
wipe (--)

chars strings

(n1 -- n2)
#FS (r:r --)
#s (n1 -- n=0)
<# (n --)
extract (n base -- n c)
F>NUMBER? (addr len -- real:r fl)
hold (c --)
r| (comp: -- <string> | exec: addr len)
s" (comp: -- <string> | exec: addr len)
s>z (a n -- z)
str (n -- addr len)
str= (addr1 len1 addr2 len2 -- fl)
z" (comp: -- <string> | exec: -- addr)
z>s (z -- a n)
[char] (comp: -- name | exec: -- xchar)

comparaison logical

0< (x1 --- fl)
0<> (n -- fl)
0= (x -- fl)
< (n1 n2 -- fl)
<= (n1 n2 -- fl)
<> (x1 x2 -- fl)
= (n1 n2 -- fl)
> (x1 x2 -- fl)
>= (x1 x2 -- fl)
f< (r1 r2 -- fl)
f<= (r1 r2 -- fl)
f<> (r1 r2 -- fl)
f= (r1 r2 -- fl)
f> (r1 r2 -- fl)
f>= (r1 r2 -- fl)
invert (x1 -- x2)
max (n1 n2 -- n1|n2)
min (n1 n2 -- n1|n2)
OR (n1 n2 -- n3)
XOR (n1 n2 -- n3)

definition words

: (comp: -- <word> | exec: --)
:noname (-- cfa-addr)
; (--)
constant (comp: n -- <name> | exec: -- n)
CREATE (comp: -- <name> | exec: -- addr)
defer (-- <vec-name>)
DOES> (comp: -- | exec: -- addr)
fvariable (comp: -- <name> | exec: -- addr)
value (comp: n -- <valname> | exec: -- n)
variable (comp: -- <name> | exec: -- addr)
vocabulary (comp: -- <name> | exec: --)

display

. (n --)
." (-- <string>)
.s (--)
? (addr -- c)
cr (--)
emit (x --)
esc (--)
f. (r --)
f.s (--)
ip. (--)
n. (n --)
normal (--)
ok (--)
prompt (--)
see (-- name>)
space (--)
spaces (n --)
type (addr c --)
u. (n --)
vlist (--)
words (--)

files words

BIN (mode -- mode')
block (n -- addr)
block-fid (-- n)
block-id (-- n)
cat (-- <path>)
CLOSE-FILE (fileid -- ior)
common-default-use (--)
cp (-- "src" "dst")
CREATE-FILE (a n mode -- fh ior)
DELETE-FILE (a n -- ior)
dump-file (addr len addr2 len2 --)
edit (-- <filename>)
file-exists? (addr len --)
FILE-POSITION (fileid -- ud ior)
FILE-SIZE (fileid -- ud ior)
FLUSH-FILE (fileid -- ior)
include (-- <:name>)
included? (addr len -- f)
ls (-- "path")
mv (-- "src" "dest")
OPEN-FILE (addr n opt -- n)
R/O (-- 0)
R/W (-- 2)
READ-FILE (a n fh -- n ior)
REPOSITION-FILE (ud fileid -- ior)
required (addr len --)
RESIZE-FILE (ud fileid -- ior)
rm (-- "path")
save-buffers (--)
touch (-- "path")
W/O (-- 1)
WRITE-FILE (a n fh -- ior)

loop and branch

+loop (n --)
?do (n1 n2 --)
aft (--)
begin (--)
CASE (--)
else (--)
ENDCASE (--)
ENDOF (--)
for (n --)
if (fl --)
loop (--)
next (--)
OF (n --)
repeat (--)
then (--)
unloop (--)
until (fl --)
while (fl --)
[ELSE] (--)
[IF] (fl --)
[THEN] (--)

memory access

! (n addr --)
2! (n1 n2 addr --)
2@ (addr -- d)
@ (addr -- n)
c! (c addr --)
c@ (addr -- c)
FP@ (-- addr)
m! (val shift mask addr --)
m@ (shift mask addr -- val)
UL@ (addr -- un)
UW@ (addr -- un[2exp0..2exp16-1])

stack manipulation

-rot (n1 n2 n3 -- n3 n1 n2)
2drop (n1 n2 n3 n4 -- n1 n2)
2dup (n1 n2 -- n1 n2 n1 n2)
>r (S: n -- R: n)
?dup (n -- n | n n)
drop (n --)
dup (n -- n n)
FDROP (r1 --)
FDUP (r1 -- r1 r1)
FNIP (r1 r2 -- r2)
FOVER (r1 r2 -- r1 r2 r1)
FSWAP (r1 r2 -- r1 r2)
nip (n1 n2 -- n2)
over (n1 n2 -- n1 n2 n1)
r> (R: n -- S: n)
R@ (-- n)
rdrop (S: -- R: n --)
swap (n1 n2 -- n2 n1)

