

# ESP32forth

## Manuel de référence

version 1.1 - 20 novembre 2023



### Auteur

- Marc PETREMANN [petremann@arduino-forth.com](mailto:petremann@arduino-forth.com)

# Table des matières

<b>Auteur.....</b>	<b>1</b>
<b>Mots FORTH par utilisation.....</b>	<b>4</b>
arithmetic integer.....	4
arithmetic real.....	4
assembler.....	5
block edit list.....	5
chars strings.....	5
comparaison logical.....	5
definition words.....	6
display.....	6
files words.....	7
input output.....	7
ledc (PWM).....	8
loop and branch.....	8
memory access.....	8
serial communication.....	8
stack manipulation.....	9
wire (I2S).....	9
XTENSA assembler.....	9
<b>forth.....</b>	<b>12</b>
<b>ansi.....</b>	<b>69</b>
<b>asm.....</b>	<b>70</b>
<b>bluetooth.....</b>	<b>72</b>
<b>editor.....</b>	<b>74</b>
<b>ESP.....</b>	<b>76</b>
<b>httpd.....</b>	<b>78</b>
<b>internalized.....</b>	<b>81</b>
<b>internals.....</b>	<b>82</b>
<b>ledc.....</b>	<b>89</b>
<b>oled.....</b>	<b>91</b>
<b>registers.....</b>	<b>96</b>
<b>riscv.....</b>	<b>97</b>
<b>rmt.....</b>	<b>100</b>
<b>serial.....</b>	<b>101</b>
<b>sockets.....</b>	<b>103</b>
<b>SPI.....</b>	<b>106</b>
<b>SPIFFS.....</b>	<b>110</b>

**tasks.....111**

**telnetd.....112**

**web-interface.....113**

**WiFi.....114**

**wire.....117**

**xtensa.....121**

# 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 )

## assembler

asm ( -- )  
assembler ( -- )  
C.LWSP, ( rd imm -- )  
chere ( -- addr )  
code ( -- <:name> )  
end-code ( -- )  
riscv-assembler ( -- )  
xtensa ( -- )  
xtensa-assembler ( -- )

## 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: -- )

u. ( n -- )

vlist ( -- )

words ( -- )

## display

. ( n -- )  
." ( -- <string> )  
.s ( -- )  
? ( addr -- c )  
at-xy ( x y -- )  
bg ( color[0..255] -- )  
cr ( -- )  
emit ( x -- )  
esc ( -- )  
f. ( r -- )  
f.s ( -- )  
fg ( color[0..255] -- )  
ip. ( -- )  
n. ( n -- )  
normal ( -- )  
ok ( -- )  
page ( -- )  
prompt ( -- )  
see ( -- name> )  
space ( -- )  
spaces ( n -- )  
type ( addr c -- )

## 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 )

## input output

accept ( addr n -- n )  
analogRead ( pin -- n )  
digitalWrite ( pin value -- )  
key ( -- char )  
key? ( -- fl )  
pinMode ( pin mode -- )

## ledc (PWM)

```
ledcAttachPin ( pin channel -- )
ledcDetachPin ( pin -- )
ledcRead ( channel -- n )
ledcReadFreq ( channel -- freq )
ledcSetup ( channel freq resolution --
freq )
ledcWrite ( channel duty -- )
ledcWriteNote ( channel note octave --
freq )
ledcWriteTone ( channel freq --
freq*1000 )
```

## 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] )
```

## serial communication

```
serial-key ( -- c )
serial-key? ( -- c )
serial-type ( addr len -- )
Serial.available ( -- n )
Serial.begin ( baud -- )
Serial.end ( -- )
Serial.flush ( -- )
Serial.readBytes ( a n -- n )
Serial.write ( addr len -- )
Serial2.available ( -- n )
Serial2.begin ( baud -- )
Serial2.end ( -- )
Serial2.flush ( -- )
Serial2.readBytes ( a n -- n )
Serial2.write ( addr len -- )
```



## 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 )
```

## wire (I2S)

```
Wire.available ( -- of-read-bytes-available )
Wire.begin ( sdapin# sclpin# -- error# )
Wire.beginTransmission ( device-address --
)
Wire.busy ( -- busy-indicator )
Wire.endTransmission ( sendstop-option --
error )
Wire.flush ( -- )
Wire.getClock ( -- clockfrequency )
Wire.getErrorText ( error --
addresspointer-to-text )
Wire.getTimeout ( -- timeout )
Wire.lastError ( -- lasterror )
Wire.peek ( -- read-data-byte )
Wire.read ( -- read-data-byte )
Wire.readTransmission ( address-of-device
address-of-data-buffer of-bytes sendstop
address-of-count -- e )
Wire.requestFrom ( address-of-device of-
bytes sendstop-option -- flag )
Wire.setClock ( clockfrequency -- )
Wire.setTimeout ( timeout -- )
Wire.write ( address-of-data-buffer of-
bytes -- )
Wire.writeTransmission ( address-of-device
address-of-data-buffer of-bytes sendstop-
option -- flag )
```

## XTENSA assembler

```
a0 ( -- 0 )
a1 ( -- 1 )
a10 ( -- 10 )
a11 ( -- 11 )
a12 ( -- 12 )
a13 ( -- 13 )
a14 ( -- 14 )
a15 ( -- 15 )
a2 ( -- 2 )
a3 ( -- 3 )
```

a4 ( -- 4 )	MULL, ( ar as at -- )
a5 ( -- 5 )	NEG, ( at ar -- )
a6 ( -- 6 )	NOP, ( -- )
a7 ( -- 7 )	NOP.N, ( -- )
a8 ( -- 8 )	OR, ( ar as at -- )
a9 ( -- 0 )	QUOS, ( at as ar -- )
ABS, ( at ar -- )	REMS, ( at as ar -- )
ABS.S, ( fr fs -- )	RET, ( -- )
ADD, ( at as ar -- )	RET.N, ( -- )
ADD.N, ( at as ar -- )	RETW.N, ( -- )
ADD.S, ( fr fs ft -- )	RSR, ( at SR[0..255] -- )
ADDI.N, ( ar as imm -- )	S32I.N, ( at as 0..60 -- )
ADDMI, ( at as -32768..32512 )	SEXT, ( ar as 7..22 -- )
ADDX2, ( ar as at -- )	SRA, ( ar at -- )
ADDX4, ( ar as at -- )	SRAI, ( ar at 0..31 -- )
ADDX8, ( ar as at -- )	SRLI, ( ar at 0..15 -- )
ALL4, ( bt bs -- )	SSA8B, ( as -- )
ALL8, ( bt bs -- )	WSR, ( at SR[0..255] -- )
AND, ( at as ar -- )	XOR, ( at as ar -- )
ANDB, ( br bs bt -- )	
ANDBC, ( br bs bt )	
ANY4, ( bt bs -- )	
ANY8, ( bt bs -- )	
BALL, ( as at label -- )	
BEQ, ( as at -- )	
BEQZ, ( as label -- )	
CALL0, ( addr -- )	
ENTRY, ( as n -- )	
EXTW, ( -- )	
J, ( label -- )	
JX, ( as -- )	
L32I.N, ( at as 0..60 -- )	
L32R, ( at offset -- )	
LOOP, ( as label -- )	
MAX, ( ar as at -- )	
MAXU, ( ar as at -- )	
MIN, ( ar as at -- )	
MINU, ( ar as at -- )	
MOV, ( ar as -- )	
MOVF, ( bt as ar -- )	
MOVI, ( at n[-2048..2047] -- )	
MOVT, ( bt as ar -- )	



# forth

**! n addr --**

Stocke une valeur entière n à l'adresse addr.

```
VARIABLE TEMPERATURE
32 TEMPERATURE !
```

**# n1 -- n2**

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:r --**

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

```
: f.
```

```
<# #fs #> type space
;
```

### #s n1 -- n=0

Convertit le reste de n1 en chaîne de caractères dans la chaîne de caractères initiée par <#.

```
: EUROS ( n1 --- 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
```

### (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.

Stocke une valeur entière à l'adresse pointée par **here**, puis incrémente **here** de la valeur de **cell**.

Dans ESP32forth, cette incrémentation est de quatre octets.

```
create myDatas
  12 , 15 , 20 , 28 ,
```

### - 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.

```
3 0< . \ display 0
-2 0< . \ display -1
```

### **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.

```
4 1+ . \ display 5
-6 1+ . \ display -5
```

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

Décrémente la valeur située au sommet de la pile.

```
4 1- . \ display 4
-6 1- . \ display -6
```

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

Effectue une opération 1/r.

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



## **2! n1 n2 addr --**

Stocke le couple de valeurs n1 n2 à l'adresse addr.

## **2\* n -- n\*2**

Multiplie n par deux.

```
3 2* .      \ display 6
-4 2* .      \ display -8
```

## **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 <= 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 Fieds 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'
```

### >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.

## **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
```

## **adc** *n* -- *n*

Alias pour **analogRead**

## **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"
;
```

## **autoexec** **--**

Recherchez **autoexec.fs** et exécutez-le s'il est présent.

## **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.

## **bluetooth --**

Sélectionne le vocabulaire **bluetooth**.

## **bterm --**

Sélectionne le vocabulaire **bterm**.

## **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-bye** (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 --**

Sélectionne le vocabulaire **camera**.

## **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
```



**cp** -- "src" "dst"

Copie le fichier "src" dans "dst".

**cr** --

Affiche un retour à la ligne suivante.

```
: .result ( ---)
  ." Port analys result" cr
  . "pool detectors" cr ;
```

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

Le mot **CREATE** peut être utilisé seul.

Le mot situé après **CREATE** est créé dans le dictionnaire, ici **DATAS**. L'exécution du mot ainsi créé dépose sur la pile de données l'adresse mémoire de la zone de paramètres. Dans cet exemple, nous avons compilé 4 valeurs 8 bits. Pour les récupérer, il faudra incrémenter l'adresse empilée avec la valeur de décalage de la donnée à récupérer.

```
\ Peripherals accessed by the CPU via 0x3FF40000 ~ 0x3FF7FFFF address space
\ (DPORT address) can also be accessed via 0x60000000 ~ 0x6003FFFF
\ (AHB address). (0x3FF40000 + n) address and (0x60000000 + n)
\ address access the same content, where n = 0 ~ 0x3FFFF.
create uartAhbBase
    $60000000 ,
    $60010000 ,
    $6002E000 ,

: REG_UART_AHB_BASE { idx -- addr }      \ id=[0,1,2]
    uartAhbBase idx cell * + @
;
```

**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
```

**dacWrite n1 n0 --**

Ecrit sur DAC (Digital Analog Converter), n1=pin, n0=value [0..255]

L'ESP32 comporte deux convertisseurs numérique / analogique (DAC) à 8 bits, ce qui permet de produire un véritable signal analogique, c'est à dire une tension pouvant prendre n'importe quelle valeur située entre 0 et 3,3V.

Deux broches peuvent servir de sortie analogique: GPIO 25 et GPIO 26.

Par exemple, pour régler la broche GPIO 25 à une valeur de 1 volt, vous écrivez:

```
25 77 dacWrite \ 77 * 3,3 / 255 = 1.
```

## 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
```

## digitalRead n -- x

Lit l'état d'une borne GPIO.

```
17 input pinMode
: test
  ." pinvalue: "
  17 digitalRead . cr
;
```

## 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
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.

### **ESP** --

Sélectionne le vocabulaire **ESP**.

### **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.

## **free** a -- f

mémoire libre précédemment réservée par **allocate**

## **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
```

**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
```

**HIGH** -- **1**

Constante. Définit l'état actif d'un pin.

```
: ledon ( -- )
  HIGH LED pin
;
```

**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 <#.

## **httpd** --

Sélectionne le vocabulaire **httpd**.

## **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? ( fl ---)
  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.

### **internals** --

Sélectionne le vocabulaire **internals**.

### **interrupts** --

Sélectionne le vocabulaire **interrupts**.

### **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.

```
: txttxtx ;
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.

## ledc --

Sélectionne le vocabulaire **ledc**.

## **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**

## **login** **z1 z2** --

Accès au WiFi seulement.

```
\ connection to local WiFi LAN
: myWiFiConnect
  z" Mariloo"
  z" 1925144D91DXXXXXXXXXXXX959F"
  login
  ;
myWiFiConnect
\ display:
\ 192.168.1.8
\ MDNS started
```

## **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.

## **LOW** -- **0**

Constante. Définit l'état inactif d'un pin.

: ledoff ( -- )

LOW LED pin

;



**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" "dst"

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
```

## oled --

Sélectionne le vocabulaire **oled**.

## 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
```

### **OUTPUT -- 2**

Constante. Valeur 2. Définit le sens d'utilisation d'un registre GPIO comme sortie.

```
: ledsetup ( -- )
  LED OUTPUT pinMode
;
```

### **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
```

### **pad -- addr**

Empile l'adresse d'une zone de mémoire tampon.

### **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
```

**pin** **n** **pin#** --

alias de **digitalWrite**

**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
```

**postpone** --

Ignore les délimiteurs d'espaces de début. Analyse *name* délimité par un espace. Trouve *name*. Ajoutez la sémantique de compilation de *name* à la définition actuelle.

**POSTPONE** remplace la plupart des fonctionnalités de **COMPILE** et **[COMPILE]**. **COMPILE** et **[COMPILE]** sont utilisés dans le même but : reporter le comportement de compilation du mot suivant dans la zone d'analyse. **COMPILE** a été conçu pour être appliqué aux mots non immédiats et **[COMPILE]** aux mots immédiats.

```
DEFINED? esp_errors [IF]    \ test if esp_err.fs loaded
  0 to NODEBUG
[ELSE]
  : .esp_error ( error -- )
    POSTPONE drop ; immediate
[THEN]
\ init GPIO used by KY-022
: ky022.gpio.init ( -- )
  RMT_RX_CHANNEL RMT_MODE_RX IR_RECEIVE_PIN INVERT_SIG
  rmt_set_gpio .esp_error
;
\ if vocabulary esp_errors is not defined, the word
\ .esp_error compile drop
```

**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 *
;
```

## **registers** --

Sélectionne le vocabulaire **registers**.

## **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
```

## **rerun** t --

Relance la temporisation du timer t

**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.

**RISC-V?** -- **-1|0**

Empile -1 si le processeur est RISC-V.

**riscv-assembler** --

Charge et installe le vocabulaire **riscv**.

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

**rm** -- **"path"**

Efface le fichier indiqué.

**rmt** --

Sélectionne le vocabulaire **rmt**.

**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
```

**rtos** --

Sélectionne le vocabulaire **rtos**.



**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.

**SD --**

Sélectionne le vocabulaire **SD**.

**SD\_MMC --**

Sélectionne le vocabulaire **SD\_MMC**.

**see -- name>**

Décompile une définition FORTH.

```
see include
: include bl PARSE included ;

see space
: space bl emit ;
```

**Serial --**

Sélectionne le vocabulaire serial

**set-precision n --**

Modifie la précision d'affichage des nombres Réels.

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

**set-title addr len ---**

Donne un titre à la fenêtre du terminal VT-xxx

```
s" This my KY-022 project" set-title
```

**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.

## **SMUDGE -- 2**

Constante. Valeur 2.

## **sockets --**

Sélectionne le vocabulaire **sockets**.

## **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**

## **SPIFFS --**

Sélectionne le vocabulaire **SPIFFS**.

## **spi\_flash --**

Sélectionne le vocabulaire **spi\_flash**.

## **start-task task --**

Démarre une tâche.

## **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**.

**telnetd** --

Sélectionne le vocabulaire **telnetd**.

**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 @

```

### **timers --**

Sélectionne le vocabulaire **timers**.

### **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-interface** --

Sélectionne le vocabulaire **web-interface**.

**webui** **z1 z2** --

Accès au WiFi et démarrage de l'interface utilisateur web.

```
z" Mariloo"                \ SSID of your WiFi access
z" 1925144D91DXXXXXXXXXXXX959F" \ password for your WiFi access
webui

\ if WiFi connection ok, display:
\ 192.168.1.22 \ can different on your network
\ MDNS started
\ Listening on port 80
```

**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
```

**WiFi** --

Sélectionne le vocabulaire **WiFi**.

**Wire** --

Sélectionne le vocabulaire **Wire**.

## **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] fl --**

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
```

# ansi

Mots définis dans le vocabulaire **ansi**

```
terminal-restore terminal-save show hide scroll-up scroll-down clear-to-eol  
bel esc
```

**bel** --

Equivalent à **7 emit**

**esc** --

Equivalent à **27 emit**

## asm

Mots définis dans le vocabulaire **asm**

```
terminal-restore terminal-save show hide scroll-up scroll-down clear-to-eol  
bel esc
```

**>>1 n1 -- n2**

Décalage de 1 bit vers la droite de n1.

**chere -- addr**

Empile l'adresse du pointeur d'assemblage.

**disasm addr --**

Désassemble le code XTENSA.

```
code myL32R  
  a1 32          ENTRY,  
  a8 $fffe      L32R,  
  a8           arPUSH,  
              RETW.N,  
end-code  
  
hex  
' myL32R cell+ @ 5 disasm
```

**end-code --**

Termine une définition en langage assembleur.

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

**names n "names"\*n --**

Définit n mots comme constantes.

```
16 names a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15
```

**odd? n -- f**

Délivre flag vrai si n est impair.

**xtensa** --

Sélectionne le vocabulaire **xtensa**.

# bluetooth

Mots définis dans le vocabulaire **bluetooth**

```
SerialBT.new SerialBT.delete SerialBT.begin SerialBT.end SerialBT.available  
SerialBT.readBytes SerialBT.write SerialBT.flush SerialBT.hasClient  
SerialBT.enableSSP SerialBT.setPin SerialBT.unpairDevice SerialBT.connect  
SerialBT.connectAddr SerialBT.disconnect SerialBT.connected  
SerialBT.isReady bluetooth-builtins
```

## bluetooth-builtins --

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

### SerialBT.available **bt -- n**

Renvoie n #octets disponibles en entrée.

### SerialBT.begin **addr master bt -- f**

Démarre une instance bluetooth.

addr : Nom local de l'unité attribué par l'utilisateur. **z" ESP32Master1"** interrupteur principal: 1 si en mode maître/contrôleur 0 si en mode esclave/récepteur BT : adresse d'objet Bluetooth attribuée avec 'Nouveau'. L'unité sera désormais visible en tant que "nom attribué par l'utilisateur" Indicateur : 1 = succès, 0 = échec

### SerialBT.connectAddr **addr bt -- f**

Un mode de connexion OPTIONNEL.

Au lieu d'utiliser un nom d'appareil, l'adresse MAC réelle de l'appareil récepteur peut être utilisée. Il s'agit d'un mode de connexion plus sécurisé que l'utilisation d'un nom de destinataire. Plusieurs appareils peuvent avoir le même nom mais pas la même adresse MAC.

### SerialBT.connected **n bt -- f**

Recherche et attend une connexion pendant la durée spécifiée en N (ms).

Drapeau : 1 vrai si connecté sinon faux 0

### SerialBT.delete **bt --**

Libère un objet BT.

### SerialBT.disconnect **bt -- f**

Se déconnecte du client.



Renvoie Vrai (1) ou Faux (0)

### **SerialBT.enableSSP** `bt --`

Cette fonction définit simplement un indicateur pour que Bluetooth utilise SSP (Simple Serial Paring)

### **SerialBT.end** `bt --`

Termine la connexion Bluetooth et l'objet.

### **SerialBT.flush** `bt --`

A utiliser pour vous assurer que le tampon de transmission a été vidé.

### **SerialBT.hasClient** `bt -- f`

Vérifie si connecté à un client.

Drapeau : Vrai (1) si connecté sinon Faux (0) non connecté.

### **SerialBT.new** `-- bt`

Alloue un nouvel objet BT.

C'est la toute première commande à exécuter lors de l'initialisation du bluetooth.

### **SerialBT.readBytes** `Bufferaddr Buffer-Size bt -- n`

Renvoie n #bytes lus dans le tampon.

### **SerialBT.setPin** `addr bt -- f`

Définit la broche utilisée pour coupler un appareil.

Ceci est FACULTATIF et non nécessaire. La broche est sous la forme **z" 1234"**

### **SerialBT.unpairDevice** `addr -- f`

L'adresse distante de réception de la plongée est requise. voir : `esp_bt_dev_get_address`

### **SerialBT.write** `Bufferaddr Buffer-Size bt -- n`

Renvoie n #bytes écrits sur Bluetooth à partir du tampon.

0 renvoyé si une erreur s'est produite. Il est recommandé de terminer les chaînes de données sortantes par un crlf (0d0ah).

## editor

Mots définis dans le vocabulaire **editor**

```
a r d e wipe p n l
```

**a** n --

Insère la ligne n.

Le numéro de ligne doit être compris entre [0..14].

Les lignes situées après la ligne insérée descendent.

Exemple : **3 Un test** insère *test* sur la ligne 3 et déplace le contenu des lignes 4 à 15.

**d** n --

Efface la ligne n.

Le numéro de ligne doit être compris entre [0..14]. Les lignes suivantes montent.

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

**e** n --

Efface le contenu de la ligne n.

Le numéro de ligne doit être compris entre [0..15]. Les autres lignes ne montent pas.

**l** --

Liste le contenu du bloc en cours de traitement.

```
1 list
editor 1 \ display block 1 content
```

**n** --

Sélectionne le bloc suivant (n pour **N**ext)

```
1 LIST
editor n 1 \ display block 2 content
```

**p** --

Sélectionne le bloc **P**récedent

```
3 LIST
editor p 1 \ display block 2 content
```

**r n --**

Remplace le contenu de la ligne n.

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

**wipe --**

Nettoie le contenu du bloc courant.

# ESP

Mots définis dans le vocabulaire **ESP**

```
getHeapSize getFreeHeap getMaxAllocHeap getChipModel getChipCores  
getFlashChipSize getCpuFreqMHz getSketchSize deepSleep getEfuseMac  
esp_log_level_set ESP-builtins
```

## **deepSleep n --**

Met la carte ESP32 en veille profonde. Le paramètre n indique le délai de mise en sommeil. En sortie de veille, ESP32Forth redémarre.

```
20000000 ESP deepSleep \ sleep mode for 20 secs approx.
```

## **ESP-builtins -- addr**

point d'entrée du vocabulaire **ESP**.

## **getChipCores -- n**

Récupère le nombre de coeurs du processeur.

```
ESP  
getChipCores . \ display 2 or other value
```

## **getChipModel -- addr-z**

Récupère l'adresse de la chaîne terminée par octet 0 qui identifie le modèle de puce de la carte ESP32.

```
ESP  
getChipModel z>s type  
\ display ESP32-D0WDQ6 or other string
```

## **getCpuFreqMHz -- n**

Récupère la fréquence en Mhz du processeur.

```
ESP  
getCpuFreqMHz . \ display 240 = 240 Mhz
```

## **getFlashChipSize -- n**

Récupère la taille de la puce flash.

```
ESP  
getFlashChipSize .  
\ display 4194304 or other value  
\ 4194304 is 4Mb flash size
```

**getSketchSize -- n**

Empile la taille du croquis utilisé par ESP32Forth.

```
getSketchSize . \ display 915728 or other value
```

# httpd

Mots définis dans le vocabulaire **httpd**

```
notfound-response bad-response ok-response response send path method hasHeader  
handleClient read-headers completed? body content-length header crnl= eat  
skipover skipto in@<> end< goal# goal strcase= upper server client-cr  
client-emit  
client-read client-type client-len client httpd-port clientfd sockfd body-read  
body-1st-read body-chunk body-chunk-size chunk-filled chunk chunk-size  
max-connections
```

**bad-response** --

Send error 400.

**body** -- **addr len**

Corps de la demande.

**chunk** -- **addr**

Zone de données définie par **create**

**chunk-filled** -- **n**

Défini par **value**

**client** -- **addr**

Défini par **sockaddr**

**client-emit** **c** --

Transmet un caractère c au réseau.

```
: client-cr  
    13 client-emit  
    n1 client-emit  
;
```

**client-len** -- **addr**

Variable.

**goal** -- **addr**

Défini par **variable**

**goal#** -- **addr**

Défini par **variable**

**handleClient** --

Récupère la prochaine demande.

**hasHeader** **addr len** -- **fl**

???

**header** **addr len** -- **addr len**

Contenu de l'en-tête (ou chaîne vide).

**http-builtins** -- **addr**

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

**httpd-port** -- **addr**

Défini par **sockaddr**

**max-connections** -- **1**

Constante. Valeur 1.

**method** -- **addr len**

Méthode de requête, exemple GET

**notfound-response** --

Send error 404.

**ok-response** **mime\$** --

Envoie 200.

```
s" text/html" ok-response
```

**path** -- **addr len**

Chemin de requête, par ex. /foo

**response** **mime\$ result\$ status** --

Transmet une réponse réseau.

```
: notfound-response ( -- )
  s" text/plain"
  s" Not Found"
  404 response
;
```

**send** **addr len --**

Chemin de la requête, par ex. /foo

**sockfd --- n**

Mémoire socket courant.

```
AF_INET SOCK_STREAM 0 socket to sockfd
```



## internalized

Ce vocabulaire est un vocabulaire annexe de **internals**.

Mots définis dans le vocabulaire **internalized**

```
flags'or! LEAVE LOOP +LOOP ?DO DO NEXT FOR AFT REPEAT WHILE ELSE IF THEN  
AHEAD UNTIL AGAIN BEGIN cleave
```

# internals

Mots définis dans le vocabulaire **internals**

```
assembler-source xtensa-assembler-source MALLOC SYSFREE REALLOC
heap_caps_malloc
heap_caps_free heap_caps_realloc heap_caps_get_total_size
heap_caps_get_free_size
heap_caps_get_minimum_free_size heap_caps_get_largest_free_block RAW-YIELD
RAW-TERMINATE READDIR CALLCODE CALL0 CALL1 CALL2 CALL3 CALL4 CALL5 CALL6
CALL7 CALL8 CALL9 CALL10 CALL11 CALL12 CALL13 CALL14 CALL15 DOFLIT S>FLOAT?
fill32 'heap 'context 'latestxt 'notfound 'heap-start 'heap-size 'stack-cells
'boot 'boot-size 'tib 'argc 'argv 'runner 'throw-handler NOP BRANCH 0BRANCH
DONEXT DOLIT DOSET DOCOL DOCON DOVAR DOCREATE DODOES ALITERAL LONG-SIZE
S>NUMBER? 'SYS YIELD EVALUATE1 'builtins internals-builtins autoexec
arduino-remember-filename
arduino-default-use esp32-stats serial-key? serial-key serial-type yield-task
yield-step e' @line grow-blocks use?! common-default-use block-data block-dirty
clobber clobber-line include+ path-join included-files raw-included include-
file
sourcedirname sourcefilename! sourcefilename sourcefilename# sourcefilename&
starts../ starts./ dirname ends/ default-remember-filename remember-filename
restore-name save-name forth-wordlist setup-saving-base 'cold park-forth
park-heap saving-base crtype cremit cases (+to) (to) --? }? ?room scope-create
do-local scope-clear scope-exit local-op scope-depth local+! local! local@
<>locals locals-here locals-area locals-gap locals-capacity ?ins. ins.
vins. onlines line-pos line-width size-all size-vocabulary vocs. voc. voclist
voclist-from see-all >vocnext see-vocabulary nonvoc? see-xt ?see-flags
see-loop see-one indent+! icr see. indent mem= ARGS MARK -TAB +TAB NONAMED
BUILTIN_FORK SMUDGE IMMEDIATE MARK relinquish dump-line ca@ cell-shift
cell-base cell-mask MALLOC_CAP_RTCRAM MALLOC_CAP_RETENTION MALLOC_CAP_IRAM_8BIT
MALLOC_CAP_DEFAULT MALLOC_CAP_INTERNAL MALLOC_CAP_SPIRAM MALLOC_CAP_DMA
MALLOC_CAP_8BIT MALLOC_CAP_32BIT MALLOC_CAP_EXEC #f+s internalized BUILTIN_MARK
zplace $place free. boot-prompt raw-ok [SKIP]' [SKIP] ?stack sp-limit
input-limit
tib-setup raw.s $@ digit parse-quote leaving, leaving )leaving leaving(
value-bind evaluate&fill evaluate-buffer arrow ?arrow. ?echo input-buffer
immediate? eat-till-cr wascr *emit *key notfound last-vocabulary voc-stack-end
xt-transfer xt-hide xt-find& scope
```

**#f+s r:r**

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

**'cold -- addr**

Adresse du mot qui sera exécuté au démarrage. Si contient 0 n'exécute pas ce mot.

```
internals
'cold @ \ if content is 0, no word to execute at starting FORTH
```

**'notfound -- addr**

Jeton d'exécution d'un gestionnaire à appeler sur un mot introuvable

**'sys -- addr**

Adresse de base pour des variables système.

**'tib -- addr**

Pointeur vers le tampon d'entrée du terminal.

**(+to) xt --**

Partie **+to** pour les variables locales.

**(to) xt --**

Partie **to** pour les variables locales.

**block-data -- addr**

Zone tampon de 1024 octets. Utilisé par **editor**.

**block-dirty -- n**

Sert de flag pour indiquer si le bloc courant a été modifié.

**BRANCH --**

Succursale à l'adresse suivant BRANCH. BRANCH est compilé par AFT, ELSE, REPEAT et AGAIN.

**common-default-use --**

Ouvre par défaut le fichier *blocks.fb*

**default-use --**

Exécute par défaut **common-default-use**.

**digit n -- c**

Convertit un nombre en caractère ascii.

```
3 digit emit \ display 3
12 digit emit \ display C
```

**DOFLIT --**

Envoie un nombre flottant depuis la cellule suivante vers la pile des réels.

## **DOLIT -- n**

Empile la cellule suivante sur la pile de données en tant que littéral entier. Il permet aux nombres d'être compilés sous forme de littéraux en ligne, fournissant des données à la pile de données au moment de l'exécution.

## **DONEXT --**

Terminer une boucle FOR-NEXT. Le nombre de boucles a été poussé sur la pile de retour et est décrémenté par DONEXT. Si le compte n'est pas négatif, sautez à l'adresse qui suit DONEXT ; sinon, faites éclater la pile de retour du décompte et quittez la boucle. DONEXT est compilé par NEXT.

## **esp32-bye --**

Redémarre ESP32Forth.

## **esp32-stats --**

Affiche les paramètres propres au CPU de la carte ESP32 et certains paramètres de ESP32Forth.

```
internals esp32-stats
\ display:
\ ESP32-D0WDQ6   240 MHz   2 cores   4194304 bytes flash
\      System Heap: 201092 free + 345808 used = 546900 total (36% free)
\
\      97964 bytes max contiguous
```

## **grow-blocks n --**

Agrandit le fichier courant de n blocs.

## **immediate? cfa -- fl**

Teste si un mot est d'exécution immédiate.

```
internal
' if      immediate?  \ leave -1 on stack
' drop    immediate?  \ leave  0 on stack
```

## **included-files -- n**

Pointe vers un fichier inclus.

## **input-buffer -- addr**

Zone mémoire définie par CREATE. Laisse sur la pile l'adresse du tampon d'entrée. Taille 200.

**input-limit -- 200**

Constante. Contient 200. Détermine la taille du tampon d'entrée de l'interpréteur FORTH.

**last-vocabulary -- addr**

Variable pointant le dernier vocabulaire défini.

**line-pos -- 0**

valeur incrémentée à chaque affichage de mot par **words**.

**line-width -- 70**

Définit le nombre de caractères par ligne pour l'exécution de **words**

**locals-capacity -- 1024**

Constante. Capacité de l'espace dédié aux variables locales.

**long-size -- 4**

Pseudo-constante. Empile 4.

**MALLOC\_CAP\_32BIT -- 2**

Constante. Valeur 2.

**MALLOC\_CAP\_8BIT -- 4**

Constante. Valeur 4.

**MALLOC\_CAP\_DMA -- 8**

Constante. Valeur 8.

**MALLOC\_CAP\_EXEC 1**

Constante. Valeur 1.

**remember-filename -- addr len**

Mot différé spécifiant le nom de fichier d'instantané par défaut spécifique à la plate-forme.

**S>NUMBER? addr len -- n fl**

Evalue le contenu d'une chaîne de caractères et tente d'en transformer le contenu en nombre. Laisse la valeur n et -1 si l'évaluation est effectuée avec succès

```
s" 27" S>NUMBER? \ leave 27 -1 on stack
```

## save-name a n --

Enregistrez un instantané si le vocabulaire actuel dans un fichier.

## see-all --

Affiche l'ensemble des mots du dictionnaire. Si le mot est défini par : affiche la décompilation de ce mot.

```
internals
see-all
\ display:
\ VOCABULARY registers
\ -----
\ : m@ @ AND SWAP RSHIFT ;
\ : m! DUP >R @ OVER invert AND >R >R LSHIFT R> AND R> OR R> ! ;
\
\ VOCABULARY oled
\ -----
\ Built-in fork: oled-builtins
\ VOCABULARY bluetooth
\ -----
\ Built-in fork: bluetooth-builtins
\ VOCABULARY rtos
\ -----
\ Built-in fork: rtos-builtins
\ VOCABULARY rmt
\ -----
\ Built-in fork: rmt-builtins
\ VOCABULARY interrupts
\ -----
\ : pinchange DUP #GPIO_INTR_ANYEDGE gpio_set_intr_type throw SWAP 0
gpio_isr_handler_add throw ;
\ DOES>/CONSTANT: #GPIO_INTR_HIGH_LEVEL
\ DOES>/CONSTANT: #GPIO_INTR_LOW_LEVEL
\ DOES>/CONSTANT: #GPIO_INTR_ANYEDGE
\ DOES>/CONSTANT: #GPIO_INTR_NEGEDGE
\ DOES>/CONSTANT: #GPIO_INTR_POSEDGE
\ DOES>/CONSTANT: #GPIO_INTR_DISABLE
\ DOES>/CONSTANT: ESP_INTR_FLAG_INTRDISABLED
\ DOES>/CONSTANT: ESP_INTR_FLAG_IRAM
\ DOES>/CONSTANT: ESP_INTR_FLAG_EDGE
\ DOES>/CONSTANT: ESP_INTR_FLAG_SHARED
\ DOES>/CONSTANT: ESP_INTR_FLAG_NMI
\ : ESP_INTR_FLAG_LEVELn 1 SWAP LSHIFT ;
\ DOES>/CONSTANT: ESP_INTR_FLAG_DEFAULT
\ Built-in fork: interrupts-builtins
\ VOCABULARY sockets
\ -----
\ : ->port! 2 + >R DUP 256 / R@ C! R> 1+ C! ;
\ : ->port@ 2 + >R R@ C@ 256 * R> 1+ C@ + ;
\ ...etc...
```

## see. xt --

Affiche le nom d'un mot FORTH depuis son code exécutable.

```
internals
```

```
' dup see. \ display DUP
```

### **serial-key -- c**

Récupère un caractère en attente dans le tampon UART0.

### **serial-key? -- c**

Execute **Serial.available**. Teste si un caractère est disponible depuis le port série UART0.

### **serial-type addr len --**

Execute **Serial.write**.

### **sourcefilename -- a n**

Empile l'adresse et la taille du nom de fichier pointés par **sourcefilename&** et **sourcefilename#**.

### **sourcefilename! a n --**

Enregistre l'adresse a et la taille n de la chaîne pointant un nom de fichier dans **sourcefilename#** et **sourcefilename&**.

### **sourcefilename# -- a**

Mémoire l'adresse de la chaîne pointant un nom de fichier.

### **sourcefilename& -- n**

Mémoire la taille de la chaîne pointant un nom de fichier.

### **voc. VOC --**

Utilisé par **vocs**.

### **voclist --**

Affiche la liste de tous les vocabulaires disponibles.

```
\ on version v7.0.6.16
internals voclist \ display:
registers
oled
bluetooth
rtos
rmt
interrupts
sockets
Serial
ledc
```

```
SPIFFS
spi_flash
SD_MMC
SD
WiFi
Wire
ESP
editor
streams
tasks
structures
internals
FORTH
```

**VOCs.** **VOC** --

Utilisé par **order**

**[SKIP]** --

Mot d'exécution différée. Exécute **[SKIP]** '

**[SKIP]'** --

Boucle qui teste les mots entre **[IF]** **[ELSE]** **{THEN}**.



# ledc

Mots définis dans le vocabulaire **ledc**

```
ledcSetup ledcAttachPin ledcDetachPin ledcRead ledcReadFreq ledcWrite  
ledcWriteTone ledcWriteNote ledc-builtins
```

## **ledc-builtins** -- addr

Point d'entrée du vocabulaire **ledc**

## **ledcAttachPin** pin channel --

Reçoit en entrée le GPIO et le canal.

Les canaux sont numérotés de 0 à 15. Pour produire un signal PWM sur une broche, il faut associer cette broche à un des 16 canaux.

```
0 value Channel  
ledc  
25 Channel ledcAttachPin \ attach GPIO25 to channel 0
```

## **ledcDetachPin** pin --

Détache le GPIO du canal.

## **ledcRead** channel -- n

Obtient la valeur du signal PWM du canal

## **ledcReadFreq** channel -- freq

Récupère la fréquence (x 1,000,000)

Renvoie la fréquence actuelle du canal spécifié (cette méthode renvoie 0 si le rapport cyclique actuel est 0).

## **ledcSetup** channel freq resolution -- freq

Initialise un canal PWM.

Définit la fréquence et le nombre de comptage (résolution du cycle de service) correspondant au canal LEDC. Renvoie la fréquence finale.

## **ledcWrite** channel duty --

Contrôle PWM

## **ledcWriteNote** channel note octave -- freq

Joue une note.

## **ledcWriteTone channel freq -- freq\*1000**

Écrit la fréquence d'un ton (x 1000)

```
0 constant CHANNEL0      \ define PWM channel 0
25 constant BUZZER        \ buzzer connected to GPI25

ledc      \ select ledc vocabulary
: initTones ( -- )
    BUZZER CHANNEL0 ledcAttachPin
    ;
initTones
CHANNEL0 400 1000 * ledcWriteTone drop
200 ms
CHANNEL0 340 1000 * ledcWriteTone drop
200 ms
CHANNEL0 230 1000 * ledcWriteTone drop
```

# oled

Mots définis dans le vocabulaire **oled**

```
OledInit SSD1306_SWITCHCAPVCC SSD1306_EXTERNALVCC WHITE BLACK
OledReset HEIGHT WIDTH OledAddr OledNew OledDelete OledBegin OledHOME OledCLS
OledTextc OledPrintln OledNumln OledNum OledDisplay OledPrint OledInvert
OledTextsize OledSetCursor OledPixel OledDrawL OledFastHLine OledFastVLine
OledCirc OledCircF OledRect OledRectF OledRectR OledRectRF oled-builtins
```

## BLACK -- 0

Constante, valeur 0

## HEIGHT -- 64

Constante, valeur 64

Indique la hauteur, en pixels d'un afficheur OLED SSD1306.

## oled-builtins -- addr

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

## OledAddr -- addr

Variable.

## OledBegin 2 I2Caddr -- fl

Démarre la gestion d'un afficheur OLED SSD1306.

```
SSD1306_SWITCHCAPVCC $3C OledBegin drop
```

## OledCirc x y radius color --

Trace un cercle centré à x y, de rayon radius et de couleur color (0|1)

```
: test-circle
  oledCLS OledDisplay
  10 10 10 white OledCirc drop
  OledDisplay
;
```

## OledCircF x y radius color --

Trace un cercle plein centré à x y, de rayon radius et de couleur color (0|1)

```
: test-circle
  oledCLS OledDisplay
  10 10 10 white OledCircF drop
  OledDisplay
```

```
;
```

## **OledCLS** --

Efface le contenu de l'écran OLED.

```
oledCLS OledDisplay
```

## **OledDelete** --

Interrompt la gestion de l'afficheur OLED SSD1306.

## **OledDisplay** --

transmet à l'afficheur OLED les commandes en attente d'affichage.

```
z" efgh" OledPrintln OledDisplay
```

## **OledDrawL** x0 y0 x1 y1 color --

Trace un trait depuis x0 y0 jusqu'à x1 y1.

## **OledFastHLine** x y length color --

Trace un trait horizontal depuis x y de dimension length et de couleur color.

```
OledCLS OledDisplay  
5 5 40 WHITE OledFastHLine OledDisplay
```

## **OledFastVLine** x y length color --

Trace un trait vertical depuis x y de dimension length et de couleur color.

```
oled  
OledCLS OledDisplay  
5 5 40 WHITE OledFastVLine OledDisplay
```

## **OledHOME** --

Positionne le curseur en ligne 0, colonne 0 sur l'afficheur OLED

## **OledInit** --

Initialise la communication avec l'afficheur OLED SSD1306.

```
oled  
128 to WIDTH  
32 to HEIGHT  
OledInit
```

## **OledInvert** --

Inverse l'affichage de l'écran OLED

## **OledNew** **width height** **OledReset** --

Instancie un nouvel afficheur OLED SSD1306 avec les paramètres **WIDTH HEIGHT** et **OledReset**.

## **OledNum** **n** --

Affiche le nombre n sous forme de chaîne sur l'écran OLED.

```
oledcls oleddisplay
0 0 oledsetcursor
1234 olednum oleddisplay
```

## **olednumln** **n** --

Affiche un nombre entier sur l'afficheur OLED et passe à la ligne suivante.

```
56 olednumln oleddisplay
\ display 56 on OLED screen
```

## **OledPixel** **x y color** -- **fl**

Active un pixel à la position x y. Le paramètre color détermine la couleur du pixel.

```
: testPixel ( -- )
  10 0 do
    i i white oledpixel drop
  loop
  oleddisplay
;
\ display 45° line begining at x y = 0 0
```

## **OledPrint** **z-string** --

Affiche un texte z-string sur l'écran OLED.

```
z" test" OledPrint OledDisplay
```

## **OledPrintln** **z-string** --

Transmet une chaîne z-string vers l'afficheur OLED. La transmission s'achève par un retour à la ligne suivante.

```
z" my string" OledPrintln OledDisplay
```

## **OledRect** **x y width height color** --

Trace un rectangle vide depuis la position x y de taille width height et de couleur color.

```
: test-rect
  oledCLS OledDisplay
  10 4 30 5 white OledRect OledDisplay
;
```

### **OledRectF** *x y width height color --*

Trace un rectangle vide depuis la position *x y* de taille *width height* et de couleur *color*.

```
: test-rect
  oledCLS OledDisplay
  10 4 30 5 white OledRectF drop
  OledDisplay
;
```

### **OledRectR** *x y width height radius color --*

Trace un rectangle à coins arrondis, depuis la position *x y*, de dimension *width height*, dans la couleur *color*, avec un rayon *radius*.

```
oled
OledCLS OledDisplay
0 0 80 30 8 WHITE OledRectR OledDisplay
```

### **OledRectRF** *x y width height radius color --*

Trace un rectangle plein à coins arrondis, depuis la position *x y*, de dimension *width height*, dans la couleur *color*, avec un rayon *radius*.

```
oled
OledCLS OledDisplay
0 0 80 30 8 WHITE OledRectRF OledDisplay
```

### **OledReset** *-- -1*

Constante, valeur -1

### **OledSetCursor** *x y --*

Définit la position du curseur.

```
0 0 OledSetCursor \ Start at top-left corner
```

### **OledTextc** *color --*

Définit la couleur du texte à afficher.

```
WHITE OledTextc \ Draw white text
```

## **OledTextsize n --**

Définit la taille du texte à afficher sur l'écran OLED. La valeur de n doit être dans l'intervalle [1..3]

Pour un texte de taille normale, n=1. Si vous dépassez la valeur 4, le texte sera tronqué sur un afficheur 4 lignes.

```
: dispText ( n -- )
  oledCLS
  OledTextsize
  WHITE OledTextc ( Draw white text )
  0 0 OledSetCursor ( Start at top-left corner )
  z" test" OledPrintln OledDisplay
;
1 dispText \ display "test" at normal size
2 dispText \ display "test" at double size
3 dispText \ display "test" at triple size
```

## **SSD1306\_EXTERNALVCC -- 1**

Constante, valeur 1

## **SSD1306\_SWITCHCAPVCC -- 2**

Constante, valeur 2

## **WHITE -- 1**

Constante, valeur 1

Permet de sélectionner la couleur des pixels à afficher.

## **WIDTH -- 128**

Constante, valeur 128

Indique la largeur, en pixels d'un afficheur OLED SSD1306.

# registers

Mots définis dans le vocabulaire **registers**

```
m@ m!
```

**m!** val shift mask addr --

Modifie le contenu d'un registre pointé par addr, applique un masque logique avec mask et décale de n bits val en fonction de shift.

```
\ Registers set for DAC control
$3FF48484 defREG: RTCIO_PAD_DAC1_REG          \ DAC1 configuration register

\ PAD DAC1 input/output value. (R/W)
$ff 19 defMASK: mRTCIO_PAD_PDACn_DAC

registers
: DAC1! ( c -- )
    mRTCIO_PAD_PDACn_DAC RTCIO_PAD_DAC1_REG m!
;
```

**m@** shift mask addr -- val

Lit le contenu d'un registre pointé par addr, applique un masque logique avec mask et décale de n bits en fonction de shift.

```
\ Registers set for DAC control
$3FF48484 defREG: RTCIO_PAD_DAC1_REG          \ DAC1 configuration register

\ PAD DAC1 input/output value. (R/W)
$ff 19 defMASK: mRTCIO_PAD_PDACn_DAC

registers
: DAC1@ ( -- c )
    mRTCIO_PAD_PDACn_DAC RTCIO_PAD_DAC1_REG m@
;
```



# riscv

Mots définis dans le vocabulaire **riscv**

```
C.FSWSP, C.SWSP, C.FSDSP, C.ADD, C.JALR, C.EBREAK, C.MV, C.JR, C.FLWSP,
C.LWSP, C.FLDSP, C.SLLI, BNEZ, BEQZ, C.J, C.ADDW, C.SUBW, C.AND, C.OR,
C.XOR, C.SUB, C.ANDI, C.SRAI, C.SRLI, C.LUI, C.LI, C.JAL, C.ADDI, C.NOP,
C.FSW, C.SW, C.FSD, C.FLW, C.LW, C.FLD, C.ADDI4SP, C.ILL, EBREAK, ECALL,
AND, OR, SRA, SRL, XOR, SLTU, SLT, SLL, SUB, ADD, SRAI, SRLI, SLLI, ANDI,
ORI, XORI, SLTIU, SLTI, ADDI, SW, SH, SB, LHU, LBU, LW, LH, LB, BGEU, BLTU,
BGE, BLT, BNE, BEQ, JALR, JAL, AUIPC, LUI, J-TYPE U-TYPE B-TYPE S-TYPE
I-TYPE R-TYPE rs2' rs2#' rs2 rs2# rs1' rs1#' rs1 rs1# rd' rd#' rd rd# offset
ofs ofs. >ofs iiii i numeric register' reg'. reg>reg' register reg. nop
x31 x30 x29 x28 x27 x26 x25 x24 x23 x22 x21 x20 x19 x18 x17 x16 x15 x14
x13 x12 x11 x10 x9 x8 x7 x6 x5 x4 x3 x2 x1 zero
```

## C.LWSP, rd imm --

Charge une valeur 32 bits de la mémoire dans le registre rd. Calcule une adresse effective en ajoutant le décalage étendu par zéro, mis à l'échelle par 4, au pointeur de pile, x2.

**x1 -- 1**

Empile 1.

**x10 -- 10**

Empile 10.

**x11 -- 11**

Empile 11.

**x12 -- 12**

Empile 12.

**x13 -- 13**

Empile 13.

**x14 -- 14**

Empile 14.

**x15 -- 15**

Empile 15.

**x16 -- 16**

Empile 16.

**x17 -- 17**

Empile 17.

**x18 -- 18**

Empile 18.

**x19 -- 19**

Empile 19.

**x2 -- 2**

Empile 2.

**x20 -- 20**

Empile 20.

**x21 -- 21**

Empile 21.

**x22 -- 22**

Empile 22.

**x23 -- 23**

Empile 23.

**x24 -- 24**

Empile 24.

**x25 -- 25**

Empile 25.

**x26 -- 26**

Empile 26.

**x27 -- 27**

Empile 27.

**x28 -- 28**

Empile 28.

**x29 -- 29**

Empile 29.

**x3 -- 3**

Empile 3.

**x30 -- 30**

Empile 30.

**x31 -- 31**

Empile 31.

**x4 -- 4**

Empile 4.

**x5 -- 5**

Empile 5.

**x6 -- 6**

Empile 6.

**x7 -- 7**

Empile 7.

**x8 -- 8**

Empile 8.

**x9 -- 9**

Empile 9.

**zero -- 0**

Empile 0.

# rmt

Mots définis dans le vocabulaire **rmt**

```
rmt_set_clk_div rmt_get_clk_div rmt_set_rx_idle_thresh rmt_get_rx_idle_thresh  
rmt_set_mem_block_num rmt_get_mem_block_num rmt_set_tx_carrier rmt_set_mem_pd  
rmt_get_mem_pd rmt_tx_start rmt_tx_stop rmt_rx_start rmt_rx_stop  
rmt_tx_memory_reset  
rmt_rx_memory_reset rmt_set_memory_owner rmt_get_memory_owner  
rmt_set_tx_loop_mode  
rmt_get_tx_loop_mode rmt_set_rx_filter rmt_set_source_clk rmt_get_source_clk  
rmt_set_idle_level rmt_get_idle_level rmt_get_status rmt_set_rx_intr_en  
rmt_set_err_intr_en rmt_set_tx_intr_en rmt_set_tx_thr_intr_en rmt_set_gpio  
rmt_config rmt_isr_register rmt_isr_deregister rmt_fill_tx_items  
rmt_driver_install  
rmt_driver_uninstall rmt_get_channel_status rmt_get_counter_clock  
rmt_write_items  
rmt_wait_tx_done rmt_get_ringbuf_handle rmt_translator_init  
rmt_translator_set_context  
rmt_translator_get_context rmt_write_sample rmt-builtins
```

**rmt-builtins** -- addr

point d'entrée du vocabulaire **rmt**.

**rmt\_driver\_uninstall** channel -- err

Désinstalle le driver RMT.

**rmt\_register\_tx\_end\_callback** --

NON SUPPORTE

**rmt\_set\_clk\_div** channel div8 -- err

Réglage du diviseur d'horloge RMT, l'horloge du canal est divisée depuis l'horloge source.

**rmt\_set\_gpio** channel mode gpio\_num invert\_signal -- err

Configurez le GPIO utilisé par le canal RMT.

**rmt\_set\_mem\_pd** channel fl -- err

Réglage de la mémoire RMT en mode faible consommation.

**rmt\_set\_pin** --

OBSOLETE, utilisez plutôt rmt\_set\_gpio

# serial

Mots définis dans le vocabulaire **Serial**

```
Serial.begin Serial.end Serial.available Serial.readBytes Serial.write  
Serial.flush Serial.setDebugOutput Serial2.begin Serial2.end Serial2.available  
Serial2.readBytes Serial2.write Serial2.flush Serial2.setDebugOutput  
serial-builtins
```

## Serial.available -- n

Délivre n|0 caractères disponibles dans le tampon de réception UART

```
115200 Serial.begin      \ initialise UART 0 at 115200 bauds  
Serial.available .      \ display 0  
S" AT" Serial.write drop \ send strint "AT" to UART  
Serial.available .      \ display 8
```

## Serial.begin baud --

Démarre le port série 0.

```
115200 Serial.begin      \ select 115200 baud rate
```

## Serial.end --

Désactive la communication série, permettant aux broches RX et TX d'être utilisées pour l'entrée et la sortie générales. Pour réactiver la communication série, utiliser

**Serial.begin.**

## Serial.flush --

Attend la fin de la transmission des données série sortantes.

## Serial.readBytes a n -- n

Lire les octets en série sur UART0, renvoyer le nombre de caractères obtenus

## Serial.write addr len --

Envoie la chaine pointée par addr|len vers UART

## Serial2.available -- n

Délivre n|0 caractères disponibles dans le tampon de réception UART 2

```
115200 Serial2.begin     \ initialise UART 2 at 115200 bauds  
Serial2.available .     \ display 0  
S" AT" Serial2.write drop \ send strint "AT" to UART  
Serial2.available .     \ display 8
```

### **Serial2.begin baud --**

Démarre le port série 2.

```
115200 Serial2.begin \ select 115200 baud rate
```

### **Serial2.end --**

Désactive la communication série, permettant aux broches RX et TX d'être utilisées pour l'entrée et la sortie générales. Pour réactiver la communication série, utiliser

**Serial2.begin.**

### **Serial2.flush --**

Attend la fin de la transmission des données série sortantes.

### **Serial2.readBytes a n -- n**

Lire les octets en série sur UART2, renvoyer le nombre de caractères obtenus

### **Serial2.write addr len --**

Envoie la chaîne pointée par addr len vers UART 2

```
\ set UART speed at 115200 baud
115200 Serial2.begin

\ select frequency 865.5 Mhz for LoRa transmission
32 string AT_BAND
s" AT+BAND=868500000" AT_BAND $! \ set frequency at 865.5 Mhz
$0a AT_BAND c+$!
$0d AT_BAND c+$! \ add CR LF code at end of command
AT_BAND Serial2.write drop
```

# sockets

Mots définis dans le vocabulaire **sockets**

```
ip. ip# ->h_addr ->addr! ->addr@ ->port! ->port@ sockaddr l, s, bs,  
SO_REUSEADDR  
SOL_SOCKET sizeof(sockaddr_in) AF_INET SOCK_RAW SOCK_DGRAM SOCK_STREAM  
socket setsockopt bind listen connect sockaccept select poll send sendto  
sendmsg recv recvfrom recvmsg gethostbyname errno sockets-builtins
```

**AF\_INET** -- 2

Constante. valeur 2

**bind** sock addr addrlen -- 0/err

Lie un nom à un socket.

**errno** -- n

Récupère la dernière erreur générée par un socket.

**gethostbyname** hostnamez -- hostent|0

Récupère une adresse *hostent* contenant des informations à partir d'un nom d'hôte.

```
WiFi  
\ connection to local WiFi LAN  
: myWiFiConnect  
  z" mySSID"  
  z" myLOGIN"  
  login ;  
Forth  
myWiFiConnect  
: hn ( -- addr0 )  
  s" arduino-forth.com" s>z  
  ;  
sockets  
hn gethostbyname ->h_addr ip.  
\ display: 54.36.91.62
```

**listen** sock connections -- 0/err

Attend une connexion sur un socket.

**setsockopt** sock level optname optval optlen -- 0/err

Définit la valeur actuelle d'une option de socket associée à un socket de n'importe quel type, dans n'importe quel état.

**sizeof(sockaddr\_in)** -- 16

Constante. valeur 16

**sockaccept** **sock addr addrlen -- sock/err**

Varie selon support bsd

**sockaddr** **-- <name>**

Crée une adresse de socket.

```
sockaddr httpd-port \ define socket address httpd-port
sockaddr client \ define socket address client
```

**socket** **family type protocol -- 0/err**

Initialise le socket.

- family représente la famille de protocole utilisé
- type indique le type de service
- protocol permet de spécifier un protocole permettant de fournir le service désiré

```
AF_INET SOCK_STREAM 0 socket to sockfd
```

**sockets-builtins** **-- addr**

point d'entrée du vocabulaire sockets.

**SOCK\_DGRAM** **-- 2**

Constante. valeur 2

Définit un type de protocole de socket: **SOCK\_DGRAM**

UDP utilise **SOCK\_DGRAM**.

```
: udp ( port -- )
  incoming ->port!
  AF_INET SOCK_DGRAM 0 socket to sockfd
  sockfd non-block throw
  sockfd incoming sizeof(sockaddr_in) bind throw
  reader-task start-task
;
```

**SOCK\_RAW** **-- 3**

Constante. valeur 3

Définit un type de protocole de socket: **SOCK\_RAW**

**SOCK\_STREAM** **-- 1**

Constante. valeur 1

Définit un type de protocole de socket: **SOCK\_STREAM**



**SOL\_SOCKET -- 1**

Constante. valeur 1

**SO\_REUSEADDR -- 2**

Constante. valeur 2

# SPI

Mots définis dans le 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.writel6 SPI.write32 SPI.writeBytes SPI.writePixels SPI.writePattern  
spi-builtins
```

## SPI-builtins -- addr

Point d'entrée du vocabulaire SPI.

## SPI.begin clk miso mosi cs --

Initialise un port SPI:

- clk est la broche à utiliser pour l'horloge
- miso est la broche à utiliser pour MISO
- mosi est la broche à utiliser pour MOSI
- cs est la broche à utiliser pour SS.

```
\ define VSPI pins  
19 constant VSPI_MISO  
23 constant VSPI_MOSI  
18 constant VSPI_SCLK  
05 constant VSPI_CS  
  
\ define SPI port frequency  
40000000 constant SPI_FREQ  
  
\ select SPI vocabulary  
SPI  
  
\ initialize HSPI port  
: init.VSPI ( -- )  
    VSPI_CS OUTPUT pinMode  
    VSPI_SCLK VSPI_MISO VSPI_MOSI VSPI_CS SPI.begin  
    SPI_FREQ SPI.setFrequency  
;
```

## SPI.end --

Ferme les ports SPI. Désactive le bus SPI et renvoie les broches à un port d'E/S général.

## SPI.getClockDivider -- divider

Récupère la valeur du diviseur d'horloge.

## **SPI.setBitOrder** `c --`

Définit l'ordre des bits.

Paramètres :

- Ordre des bits, LSBFIRST ou MSBFIRST. La valeur par défaut est MSBFIRST

```
\ definition of LSBFIRST and MSBFIRST constants
0 constant LSBFIRST
1 constant MSBFIRST
```

## **SPI.setClockDivider** `divider --`

Définit le diviseur d'horloge SPI par rapport à l'horloge système.

```
\ define differents divider values
$00101001 constant SPI_CLOCK_DIV2 \ 8 MHz
$004c1001 constant SPI_CLOCK_DIV4 \ 4 MHz
$004c1001 constant SPI_CLOCK_DIV8 \ 2 MHz
$009c1001 constant SPI_CLOCK_DIV16 \ 1 MHz
$013c1001 constant SPI_CLOCK_DIV32 \ 500 KHz
$027c1001 constant SPI_CLOCK_DIV64 \ 250 KHz
$04fc1001 constant SPI_CLOCK_DIV128 \ 125 KHz
```

## **SPI.setDataMode** `c --`

Définit le mode de données. C'est la polarité et la phase de l'horloge.

- SPI\_MODE0 : L'horloge est basse pendant l'inactivité, échantillonne les données sur le front montant
- SPI\_MODE1 : L'horloge est basse pendant l'inactivité, échantillonne les données au front descendant
- SPI\_MODE2 : L'horloge est haute pendant l'inactivité, échantillonne les données sur le front montant
- SPI\_MODE3 : L'horloge est haute pendant l'inactivité, échantillonne les données au front descendant

```
\ definition of SPI_MODE0..SPI_MODE3 constants
0 constant SPI_MODE0
1 constant SPI_MODE1
2 constant SPI_MODE2
3 constant SPI_MODE3
```

## **SPI.setFrequency** `n --`

Définit la valeur de fréquence.

```
\ define SPI port frequency
40000000 constant SPI_FREQ
SPI \ select SPI vocabulary
```

### **SPI.setHwCs** `fl --`

Définit la ligne SPI CS pour basculer à chaque octet.

La valeur 1 active le port SPI, 0 désactive le port SPI pour l'interface connecté à ce port SPI.

```
\ send two bytes to MAX7219 thru SPI port
: MAX7219.send ( c1 c2 -- )
  1 SPI.setHwCs
  swap 8 lshift + SPI.write16
  0 SPI.setHwCs
;
```

### **SPI.transfer** `c -- c`

Transfère un octet sur le bus SPI, à la fois en envoi et en réception.

### **SPI.transfer16** `n -- n'`

Transmet 16 bits sur le port SPI. Si le pin MISO est exploité, empile la valeur transmise par le périphérique connecté au port SPI.

### **SPI.transfer32** `n -- n'`

Transmet 32 bits sur le port SPI. Si le pin MISO est exploité, empile la valeur transmise par le périphérique connecté au port SPI.

### **SPI.transfer8** `n -- n'`

Transmet 8 bits sur le port SPI. Si le pin MISO est exploité, empile la valeur transmise par le périphérique connecté au port SPI.

### **SPI.transferBytes** `data out size --`

Transfère un tampon rempli de données.

### **SPI.write** `n --`

Transmet la partie 8 bits de poids faible de n vers le port SPI.

### **SPI.write16** `n --`

Transmet la partie 16 bits de poids faible de n vers le port SPI.

Le mot **SPI.write16** peut remplacer deux exécutions de **SPI.write**.

```
\ send two bytes to MAX7219 thru SPI port
\ : MAX7219.send ( c1 c2 -- )
```

```

\      MAX7219.select
\      swap SPI.write SPI.write
\      MAX7219.deselect
\      ;

\ send two bytes to MAX7219 thru SPI port
: MAX7219.send ( c1 c2 -- )
  MAX7219.select
  swap 8 lshift + SPI.writel6
  MAX7219.deselect
;

```

### **SPI.write32 n --**

Transmet la valeur 32 bits n vers le port SPI.

### **SPI.writeBytes datas size --**

Envoie un tampon de données et ignore le retour.

# SPIFFS

Mots définis dans le vocabulaire **SPIFFS**

```
SPIFFS.begin SPIFFS.end SPIFFS.format SPIFFS.totalBytes SPIFFS.usedBytes  
SPIFFS-builtins
```

## **SPIFFS.begin** **n2 c1 n0 -- fl**

Ce mot monte le système de fichiers SPIFFS et doit être appelée avant que n'importe quelle autre méthode FS ne soit utilisée. Renvoie true si le système de fichiers a été monté avec succès.

```
-1 z" /spiffs" 10 SPIFFS.begin drop
```

## **SPIFFS.end** **--**

Démonte le système de fichier.

## **SPIFFS.format** **-- fl**

Formate le système de fichier. Renvoie true si le formatage a réussi. Attention, si des fichiers sont présents dans la zone mémoire, ils seront supprimés de façon irréversible.

Retourne en sortie une valeur booléenne indiquant si la procédure a réussi (vrai) ou échoué (faux).

```
SPIFFS.format      \ take many seconds  
SPIFFS.usedBytes   . \ display 0  
SPIFFS.totalBytes  . \ display 1860161
```

## **SPIFFS.totalBytes** **-- n**

Renvoie le nombre total d'octets utilisés par le système de fichier SPIFFS.

## **SPIFFS.usedBytes** **-- n**

Renvoie l'espace utilisé par le fichier spécifié en octets.

## tasks

Mots définis dans le vocabulaire **tasks**

```
main-task .tasks task-list
```

### **SPIFFS.begin**    **n2 c1 n0 -- fl**

Ce mot monte le système de fichiers SPIFFS et doit être appelée avant que n'importe quelle autre méthode FS ne soit utilisée. Renvoie true si le système de fichiers a été monté avec succès.

```
-1 z" /spiffs" 10 SPIFFS.begin drop
```

### **SPIFFS.end**    **--**

Démonte le système de fichier.

### **SPIFFS.format**    **-- fl**

Formate le système de fichier. Renvoie true si le formatage a réussi. Attention, si des fichiers sont présents dans la zone mémoire, ils seront supprimés de façon irréversible.

Retourne en sortie une valeur booléenne indiquant si la procédure a réussi (vrai) ou échoué (faux).

```
SPIFFS.format      \ take many seconds  
SPIFFS.usedBytes   . \ display 0  
SPIFFS.totalBytes  . \ display 1860161
```

### **SPIFFS.totalBytes**    **-- n**

Renvoie le nombre total d'octets utilisés par le système de fichier SPIFFS.

### **SPIFFS.usedBytes**    **-- n**

Renvoie l'espace utilisé par le fichier spécifié en octets.

# telnetd

Mots définis dans le vocabulaire **telnetd**

```
server broker-connection wait-for-connection connection telnet-key telnet-type  
telnet-emit broker client-len client telnet-port clientfd sockfd
```

**server port --**

Lance le serveur telnet sur le port indiqué.

```
z" WiFiSSID"          \ SSID of your WiFi access  
z" password"          \ password of your WiFi access  
  login  
cr telnetd 552 server  \ activate TELNET server
```

**telnet-emit ch --**

Emet un caractère sur le port TELNET actif.

**telnet-key -- n**

Récupère un caractère depuis le port TELNET actif.

**telnet-type addr len --**

Transmet une chaîne de caractères sur le port TELNET actif.



## web-interface

Mots définis dans le vocabulaire **web-interface**

```
server webserver-task do-serve handle1 serve-key serve-type handle-input  
handle-index out-string output-stream input-stream out-size webserver index-  
html  
index-html#
```

### index-html -- addr

Marque l'adresse de la chaine de l'interface web.

```
index-html index-html# type
```

### index-html# -- addr

Marque la taille de la chaine de l'interface web.

```
index-html index-html# type
```

### ip# n -- n'

Affiche une partie de l'adresse IP. Utilisé par **ip.**

### ip. --

Affiche une adresse IP à partir de son adresse 32 bits.

```
WIFI_MODE_STA Wifi.mode  
z" toSSID" \ type SSID of your network=k  
z" passwordToSSID" \ type password of your network  
WiFi.status . \ display status  
\ if status = 3 you can get and display IP address  
WiFi.localIP ip. \ display local IP address
```

# WiFi

Mots définis dans le vocabulaire **WiFi**

```
WiFi_MODE_APSTA WiFi_MODE_AP WiFi_MODE_STA WiFi_MODE_NULL WiFi.config
WiFi.begin
WiFi.disconnect WiFi.status WiFi.macAddress WiFi.localIP WiFi.mode
WiFi.setTxPower
WiFi.getTxPower WiFi.softAP WiFi.softAPIP WiFi.softAPBroadcastIP
WiFi.softAPNetworkID
WiFi.softAPConfig WiFi.softAPdisconnect WiFi.softAPgetStationNum WiFi-builtins
```

## WiFi-builtins -- addr

Point d'entrée du vocabulaire **WiFi**

## Wifi.begin ssid-z password-z

Initialise les paramètres réseau de WiFi et fournit l'état actuel.

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

## WiFi.config ip dns gateway subnet --

**WiFi.config** vous permet de configurer une adresse IP statique ainsi que de modifier l'adresses DNS, passerelle et sous-réseau sur le module WiFi.

Contrairement à **WiFi.begin** qui configure automatiquement le module WiFi pour utiliser DHCP, **WiFi.config** permet de définir manuellement l'adresse réseau du module. Appeler **WiFi.config** avant **WiFi.begin** oblige à configurer le WiFi avec les adresses réseau spécifiées dans **WiFi.config**. Paramètres: • **ip**: l'adresse IP de l'appareil •

- **dns**: l'adresse d'un serveur DNS. •
- **passerelle**: l'adresse IP de la passerelle réseau •
- **sous-réseau**: le masque de sous-réseau du réseau<

## Wifi.disconnect --

Déconnecte le module WiFi du réseau actuel.

## WiFi.getTxPower -- powerx4

Récupère la puissance d'émission x4.

## WiFi.localIP -- ip

Récupère l'adresse IP locale.

## WiFi.macAddress -- a

Obtient l'adresse MAC de votre port WiFi ESP32.

```
create mac 6 allot  
mac WiFi WiFi.macAddress
```

## WiFi.mode mode --

Sélectionne le mode WiFi: WIFI\_MODE\_NULL WIFI\_MODE\_STA WIFI\_MODE\_AP  
WIFI\_MODE\_APSTA

## WiFi.setTxPower powerx4 --

Définit la puissance x4.

## WiFi.softAP ssid password/0 -- success

Définit le SSID et mot de passe pour l'accès en mode AP (Access Point).

## WiFi.status -- n

Renvoie l'état de la connexion.

### • 255 WL\_NO\_SHIELD

attribué lorsqu'aucun module WiFi n'est présent • 0 WL\_IDLE\_STATUS

il s'agit d'un statut temporaire attribué lorsque **WiFi.begin** est appelé et reste actif jusqu'à l'expiration du nombre de tentatives (résultant en WL\_CONNECT\_FAILED) ou une connexion est établie (résultant en WL\_CONNECTED) • 1 WL\_NO\_SSID\_AVAIL

attribué lorsqu'aucun SSID n'est disponible • 2 WL\_SCAN\_COMPLETED

attribué lorsque l'analyse des réseaux est terminée • 3 WL\_CONNECTED

attribué lorsqu'il est connecté à un réseau Wi-Fi • 4 WL\_CONNECT\_FAILED

attribué lorsque la connexion échoue pour toutes les tentatives •

5 WL\_CONNECTION\_LOST

attribué lorsque la connexion est perdue • 6 WL\_DISCONNECTED

attribué lors de la déconnexion d'un réseau

```
WiFi.status . \ display status
```

## WIFI\_MODE\_AP -- 2

Constante. Valeur 2

Mode AP : dans ce mode, initialisez les données AP internes, tandis que l'interface AP est prête pour les données Wi-Fi RX/TX. Ensuite, le pilote Wi-Fi commence à diffuser des balises et le point d'accès est prêt à se connecter à d'autres stations.

### **WIFI\_MODE\_APSTA -- 3**

Constante. Valeur 3

Mode de coexistence station-AP : dans ce mode, initialisera simultanément la station et l'AP. Cela se fait en mode station et en mode AP. Veuillez noter que le canal du point d'accès externe auquel la station ESP est connectée a une priorité plus élevée que le canal du point d'accès ESP.

### **WIFI\_MODE\_NULL -- 0**

Constante. Valeur 0

Dans ce mode, la structure de données interne n'est pas attribuée à la station et au point d'accès, tandis que les interfaces de la station et du point d'accès ne sont pas initialisées pour les données Wi-Fi RX/TX. Généralement, ce mode est utilisé pour Sniffer, ou lorsque vous souhaitez uniquement arrêter à la fois le STA et le point d'accès pour décharger l'ensemble du pilote Wi-Fi.

### **WIFI\_MODE\_STA -- 1**

Constante. Valeur 1

Mode station : dans ce mode, les données internes de la station seront initialisées, tandis que l'interface de la station est prête pour les données Wi-Fi RX et TX.

# wire

Mots définis dans le vocabulaire **wire**

```
Wire.begin Wire.setClock Wire.getClock Wire.setTimeout Wire.setTimeout  
Wire.beginTransaction Wire.endTransmission Wire.requestFrom Wire.write  
Wire.available Wire.read Wire.peek Wire.flush Wire-builtins
```

## Wire-builtins -- addr

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

## Wire.available -- of-read-bytes-available

Renvoie le nombre d'octets disponibles pour la récupération avec le mot `Wire.read`. Cela devrait être appelé sur un périphérique maître après l'utilisation du mot `Wire.requestFrom`.

## Wire.begin sdapin# sclpin# -- error#

Lance la bibliothèque `Wire` et rejoint le bus I2C en tant que maître en utilisant les broches spécifiées pour `sda` et `scl`. L'option esclave n'est pas disponible. Cela ne devrait normalement être appelé qu'une seule fois.

`sdapin#` et `sclpin#` spécifiant les numéros de broches utilisés pour `sda` et `scl`. `error#` 1 pour le succès et 0 pour l'échec. Le 1 renvoyé indique que le bus I2C a été démarré correctement.

```
\ activate the wire vocabulary  
wire  
\ start the I2C interface using pin 21 and 22 on ESP32 DEVKIT V1  
\ with 21 used as sda and 22 as scl.  
21 22 wire.begin
```

## Wire.beginTransaction device-address --

Commencer une transmission vers le périphérique esclave I2C avec l'adresse donnée. Ensuite, mettez les octets en file d'attente pour la transmission avec la fonction **Wire.write** et transmettez-les en appelant **Wire.endTransmission**.

`device-address` spécifiant l'adresse du périphérique esclave vers lequel transmettre.

```
Wire  
  
\ set adress of OLED SSD1306 display  
$3c constant addrSSD1306  
  
: toSSD1306 ( addr len -- )  
  addrSSD1306 Wire.beginTransaction  
  Wire.write drop  
  addrSSD1306 Wire.endTransmission drop  
;
```

## **Wire.busy -- busy-indicator**

Lit l'état du bus I2C. 1 pour occupé et 0 pour libre.

## **Wire.endTransmission sendstop-option -- error**

Termine une transmission vers un périphérique esclave qui a été commencée par beginTransmission et transmet les octets mis en file d'attente par l'écriture.

sendstop-option: paramètre modifiant son comportement pour compatibilité avec certains appareils I2C. Si vrai ou 1, endTransmission envoie un message d'arrêt après la transmission, libérant le bus I2C. Si false ou 0, endTransmission envoie un message de redémarrage après la transmission. Le bus ne sera pas libéré, ce qui empêche un autre appareil maître de transmettre entre les messages. Cela permet à un appareil maître d'envoyer plusieurs transmissions tout en contrôlant. La valeur par défaut est true. error : qui indique l'état de la transmission : 0 : succès 1 : données trop longues pour tenir dans le tampon de transmission 2 : NACK reçu lors de la transmission de l'adresse 3 : NACK reçu lors de la transmission des données 4 : autre erreur

```
Wire

\ set address of OLED SSD1306 display
$3c constant addrSSD1306

: toSSD1306 ( addr len -- )
  addrSSD1306 Wire.beginTransaction
  Wire.write drop
  addrSSD1306 Wire.endTransmission drop
;
```

## **Wire.flush --**

Libère le bus I2C.

## **Wire.getClock -- clockfrequency**

Obtient la fréquence d'horloge pour la communication I2C.

Les entrées suivantes effectuent les opérations suivantes : - active le vocabulaire filaire avec un 1 pour indiquer le succès, - démarre l'interface I2C en utilisant les broches 21 et 22 sur ESP32 DEVKIT V1 avec 21 utilisé comme sda et 22 comme scl, - affiche la vitesse d'horloge par défaut de 100000 Khz après Wire.begin, - règle la fréquence d'horloge à 400000 Khz, - confirme la vitesse d'horloge à 400000 Khz.

```
21 22 wire.begin      \ push 1 on stack
wire.getClock         \ push 100000 on stack
400000 wire.setclock
wire.getClock         \ push 400000 on stack
```

**Wire.getErrorText** *error -- addresspointer-to-text*

Obtient l'adresse du texte à terminaison nulle correspondant à l'erreur spécifiée.

**Wire.getTimeout** *-- timeout*

Obtient le délai d'attente en ms pour la communication I2C.

**Wire.lastError** *-- lasterror*

Récupère la dernière erreur de communication I2C.

**Wire.peek** *-- read-data-byte*

Lit un octet à partir d'un périphérique esclave précédemment adressé à l'aide du mot `requestFrom`. C'est la même chose qu'un `Wire.read` sauf que le pointeur de tampon de données reçu n'est pas incrémenté.

**Wire.read** *-- read-data-byte*

Lit un octet à partir d'un périphérique esclave précédemment adressé à l'aide du mot `Wire.requestFrom`.

**Wire.readTransmission** *address-of-device address-of-data-buffer of-bytes sendstop address-of-count -- e*

Utilisé par le maître pour demander un nombre spécifié d'octets dans un tampon de données à l'adresse du tampon de données au périphérique esclave avec l'adresse du périphérique, puis terminer la transmission ou non comme spécifié par l'option `sendstop`.

**Wire.requestFrom** *address-of-device of-bytes sendstop-option -- flag*

Utilisé par le maître pour demander des octets à un appareil esclave. Les octets peuvent ensuite être récupérés avec les mots `wire.available` et `wire.read`. Les octets sont réellement reçus du dispositif esclave adressé et stockés dans un tampon de données puis la connexion est terminée ou non si nécessaire. Le mot `wire.available` indique le nombre d'octets présents dans ce tampon de données et `wire.read` permet de lire les octets du tampon de données. `Wire.requestFrom` accepte un paramètre `sendstop-option` modifiant son comportement pour la compatibilité avec certains périphériques I2C.

**Wire.setClock** *clockfrequency --*

Modifie la fréquence d'horloge pour la communication I2C. Les appareils esclaves I2C n'ont pas d'horloge de travail minimale fréquence, cependant 100KHz est généralement la ligne de base.

`clockFrequency`: la valeur (en Hertz) de l'horloge de communication. Les valeurs acceptées sont 100000 (mode standard) et 400000 (mode rapide). Certains processeurs prennent également en charge 10000 (mode basse vitesse), 1000000 (mode rapide plus) et

3400000 (mode haute vitesse). Veuillez vous référer à la documentation spécifique du processeur pour vous assurer que le mode souhaité est pris en charge.

### **Wire.setTimeout** *timeout --*

Modifie le délai d'attente en ms pour la communication I2C. Le timeout est exprimé en ms. La valeur par défaut est 50 ms.

### **Wire.write** *address-of-data-buffer of-bytes --*

Files d'attente d'octets vers le tampon d'adresse de données pour la transmission d'un maître vers un périphérique esclave précédemment sélectionné (entre les mots

**Wire.beginTransaction** et **Wire.endTransmission**).

```
Wire
\ set adress of OLED SSD1306 display
$3c constant addrSSD1306

: toSSD1306 ( addr len -- )
  addrSSD1306 Wire.beginTransaction
  Wire.write drop
  addrSSD1306 Wire.endTransmission drop
;
```

### **Wire.writeTransmission** *address-of-device address-of-data-buffer of-bytes sendstop-option -- flag*

Utilisé par le maître pour envoyer un nombre spécifié d'octets à partir d'un tampon de données à l'adresse-du-tampon-de-données vers l'appareil esclave avec l'adresse-de-l'appareil, puis terminer la transmission. L'option sendstop est définie sur true.





**a10 -- 10**

Empile 10.

**a11 -- 11**

Empile 11.

**a12 -- 12**

Empile 12.

**a13 -- 13**

Empile 13.

**a14 -- 14**

Empile 14.

**a15 -- 15**

Empile 15.

**a2 -- 2**

Empile 2.

**a3 -- 3**

Empile 3.

**a4 -- 4**

Empile 4.

**a5 -- 5**

Empile 5.

**a6 -- 6**

Empile 6.

**a7 -- 7**

Empile 7.

**a8 -- 8**

Empile 8.

**a9 -- 0**

Empile 9.

**ABS, at ar --**

Valeur absolue. Format RRR

**ABS**, calcule la valeur absolue du contenu du registre d'adresse at et l'écrit dans le registre d'adresse ar. Le débordement arithmétique n'est pas détecté.

```
\ for macros, see:
\
https://github.com/MPETREMAN11/ESP32forth/blob/main/assembler/xtensaMacros.txt
code myABS ( n -- n' )
  a1 32      ENTRY,
  a7        arPOP,
  a7 a7      ABS,
  a7        arPUSH,
  a7 a2 0    S32I.N,
              RETW.N,
end-code

  4 myABS . \ display: 4
-4 myABS . \ display: 4
```

**ABS.S, fr fs --**

Valeur absolue unique. Mot d'instruction (RRR).

**ABS.S**, calcule la valeur absolue simple précision du contenu du registre à virgule flottante fs et écrit le résultat dans le registre à virgule flottante fr.

**ADD, at as ar --**

Addition. Mot d'instruction (RRR).

**ADD**, calcule la somme de 32 bits en complément à deux des registres d'adresses as et at. Les 32 bits de poids faible de la somme sont écrits dans le registre d'adresse ar. Le débordement arithmétique n'est pas détecté.

```
\ for marcos, see:
\
https://github.com/MPETREMAN11/ESP32forth/blob/main/assembler/xtensaMacros.txt
code my+
  a1 32      ENTRY,
  a7        arPOP,
  a8        arPOP,
  a7 a8 a9 ADD,
  a9        arPUSH,
              RETW.N,
end-code

5 2 my+
```

### ADD.N, at as ar --

Addition proche. Mot d'instruction (RRRN).  $AR[r] \leftarrow AR[s] + AR[t]$

Ceci effectue la même opération que l'instruction **ADD**, dans un codage 16 bits.

```
\ for macros, see:
\
https://github.com/MPETREMAN11/ESP32forth/blob/main/assembler/xtensaMacros.txt
code myADD ( n -- n' )
  a1 32      ENTRY,
  a7      arPOP,
  a8      arPOP,
  a7 a8 a9    ADD.N,
  a9      arPUSH,
              RETW.N,
end-code

 4 7  myADD . \ display: 11
-4 -7 myADD . \ display: -11
```

### ADD.S, fr fs ft --

Addition simple. Mot d'instruction (RRR).

**ADD.S**, calcule la somme simple précision IEEE754 du contenu des registres à virgule flottante fs et ft, et écrit le résultat dans le registre à virgule flottante fr.

### ADDI, at as -128..127

Addition Immédiate. Instruction de mots (RRI8).  $AR[t] \leftarrow AR[s] + (imm8724 || imm8)$

**ADDI**, calcule la somme 32 bits en complément à deux du registre d'adresse et d'une constante codée dans le champ imm8. Les 32 bits de poids faible de la somme sont écrits dans le registre d'adresses at. Le débordement arithmétique n'est pas détecté.

```
\ for macros, see:
\
https://github.com/MPETREMAN11/ESP32forth/blob/main/assembler/xtensaMacros.txt
code my1+ ( n -- n' )
  a1 32      ENTRY,
  a7      arPOP,
  a7 a7 1     ADDI,
  a7      arPUSH,
              RETW.N,
end-code

 5 my1+ . \ display: 6
-5 my1+ . \ display: -4
```

### ADDI.N, ar as imm --

Somme Immédiate proche. Mot d'instruction (RRRN).

**ADDI.N**, est similaire à **ADDI**, mais a un codage 16 bits et prend en charge une plus petite plage de valeurs d'opérandes immédiates codées dans le mot d'instruction. **ADDI.N**, calcule la somme de 32 bits en complément à deux du registre d'adresse as et d'un opérande codé dans le champ t. Les 32 bits de poids faible de la somme sont écrits dans le registre d'adresse ar. Le débordement arithmétique n'est pas détecté. L'opérande encodé dans l'instruction peut être -1 ou un à 15. Si t est zéro, alors une valeur de -1 est utilisée, sinon la valeur est l'extension zéro de t.

### **ADDMI, at as -32768..32512**

Addition Immédiate avec Décalage de 8. Mot d'instruction (RRI8).

**ADDMI**, étend la plage d'addition constante. Il est souvent utilisé en conjonction avec les instructions de chargement et de stockage pour étendre la portée de la base et compenser le calcul. **ADDMI** calcule la somme de 32 bits en complément à deux du registre d'adresse en tant qu'opérateur et encodée dans le champ imm8. Les 32 bits de poids faible de la somme sont écrits dans le registre d'adresses at. Le débordement arithmétique n'est pas détecté. L'opérande codé dans l'instruction peut avoir des valeurs multiples de 256 allant de -32768 à 32512. Il est décodé en étendant le signe imm8 et en décalant le résultat de huit bits vers la gauche.

### **ADDX2, ar as at --**

Ajouter avec Décalage de 1. Mot d'instruction (RRR).

**ADDX2**, calcule la somme de 32 bits du complément à deux du registre d'adresse décalé vers la gauche d'un bit et du registre d'adresse à. Les 32 bits de poids faible de la somme sont écrits dans le registre d'adresse ar. Le débordement arithmétique n'est pas détecté.

**ADDX2**, est fréquemment utilisé pour le calcul d'adresse et dans le cadre de séquences à multiplier par petites constantes.

### **ADDX4, ar as at --**

Ajouter avec Décalage de 1. Mot d'instruction (RRR).

**ADDX4**, calcule la somme de 32 bits du complément à deux du registre d'adresse décalé vers la gauche de deux bits et du registre d'adresse at. Les 32 bits de poids faible de la somme sont écrits dans le registre d'adresse ar. Le débordement arithmétique n'est pas détecté. **ADDX4**, est fréquemment utilisé pour le calcul d'adresse et dans le cadre de séquences à multiplier par petites constantes.

### **ADDX8, ar as at --**

Ajouter avec Décalage de 1. Mot d'instruction (RRR).

**ADDX8**, calcule la somme de 32 bits du complément à deux du registre d'adresse décalé vers la gauche de trois bits et du registre d'adresse at. Les 32 bits de poids faible de la

somme sont écrits dans le registre d'adresse ar. Le débordement arithmétique n'est pas détecté. **ADDX8**, est fréquemment utilisé pour le calcul d'adresse et dans le cadre de séquences à multiplier par petites constantes.

#### **ALL4, bt bs --**

Les 4 booléens sont vrais. Mot d'instruction (RRR).

**ALL4**, définit le registre booléen bt sur le et logique des quatre registres booléens bs+0, bs+1, bs+2 et bs+3. bs doit être un multiple de quatre (b0, b4, b8 ou b12) ; sinon le fonctionnement de cette instruction n'est pas défini. **ALL4**, réduit quatre résultats de test de sorte que le résultat est vrai si les quatre tests sont vrais. Lorsque le sens des booléens bs est inversé (0 → vrai, 1 → faux), utilisez **ANY4**, et un test inversé du résultat.

#### **ALL8, bt bs --**

Les 8 booléens sont vrais. Mot d'instruction (RRR).

**ALL8**, définit le registre booléen bt sur le registre logique et des huit registres booléens bs+0, bs+1, ... bs+6 et bs+7. bs doit être un multiple de huit (b0 ou b8) ; sinon le fonctionnement de cette instruction n'est pas défini. **ALL8** réduit huit résultats de test de sorte que le résultat est vrai si les huit tests sont vrais. Lorsque le sens des booléens bs est inversé (0 → vrai, 1 → faux), utilisez **ANY8**, et un test inversé du résultat.

#### **AND, at as ar --**

Et logique au niveau du bit. Mot d'instruction (RRR).  $AR[r] \leftarrow AR[s] \text{ and } AR[t]$

**AND**, calcule la logique bit à bit et des registres d'adresse au fur et à. Le résultat est écrit dans le registre d'adresse ar.

```
\ for macros, see:
\
https://github.com/MPETREMANN11/ESP32forth/blob/main/assembler/xtensaMacros.txt
code myAND ( n -- n' )
  a1 32      ENTRY,
  a7      arPOP,
  a8      arPOP,
  a7 a8 a9    AND,
  a9      arPUSH,
          RETW.N,
end-code

hex
$ff $73    myAND .    \ display: 73
$08 $04    myAND .    \ display: 0
decimal
```

#### **ANDB, br bs bt --**

Et booléen. Mot d'instruction (RRR).

**ANDB**, effectue le et logique des registres booléens bs et bt et écrit le résultat dans le registre booléen br.

### **ANDBC, br bs bt**

Et booléen avec complément. Mot d'instruction (RRR).

**ANDBC**, effectue le et logique du registre booléen bs avec le complément logique du registre booléen bt, et écrit le résultat dans le registre booléen br.

### **ANY4, bt bs --**

N'importe quel 4 booléens Vrai. Mot d'instruction (RRR).

**ANY4**, définit le registre booléen bt sur le ou logique des quatre registres booléens bs+0, bs+1, bs+2 et bs+3. bs doit être un multiple de quatre (b0, b4, b8 ou b12) ; sinon le fonctionnement de cette instruction n'est pas défini. **ANY4**, réduit quatre résultats de test de sorte que le résultat est vrai si l'un des quatre tests est vrai. Lorsque le sens des booléens bs est inversé (0 → vrai, 1 → faux), utilisez **ALL4**, et un test inversé du résultat.

### **ANY8, bt bs --**

N'importe quel 8 booléens Vrai. Mot d'instruction (RRR).

**ANY8**, définit le registre booléen bt sur le ou logique des huit registres booléens bs+0, bs+1, ... bs+6 et bs+7. bs doit être un multiple de huit (b0 ou b8) ; sinon le fonctionnement de cette instruction n'est pas défini. **ANY8**, réduit huit résultats de test de sorte que le résultat est vrai si l'un des huit tests est vrai. Lorsque le sens des booléens bs est inversé (0 → vrai, 1 → faux), utilisez **ALL8**, et un test inversé du résultat.

### **BALL, as at label --**

Branchement si tous les bits sont définis. Mot d'instruction (RRI8).

**BALL**, bifurque si tous les bits spécifiés par le masque dans le registre d'adresse at sont définis dans le registre d'adresse as. Le test est effectué en prenant le et logique au niveau du bit de à et le complément de as, et en testant si le résultat est zéro. L'adresse d'instruction cible de la branche est donnée par l'adresse de l'instruction **BALL**, , plus le champ imm8 de 8 bits étendu par signe de l'instruction plus quatre. Si l'un des bits masqués est effacé, l'exécution continue avec l'instruction séquentielle suivante. L'inverse de **BALL**, est **NBALL**, .

### **BEQ, as at --**

Branchement si égal.

**BEQ**, branche si les registres d'adresse as et at sont égaux. Il est conseillé d'utiliser ce branchement au travers de la macro instruction **<>**, .

```

: <>, ( as at -- )
  0 BEQ,
;

code my<> ( n1 n2 -- f1 )    \ f1=1 if n1 = n2
  a1 32      ENTRY,
  a8        arPOP,          \ a8 = n2
  a9        arPOP,          \ a9 = n1
  a7 0       MOVI,          \ a7 = 1
  a8 a9 <>, If,
    a7 1     MOVI,          \ a7 = 0
  Then,
  a7        arPUSH,
            RETW.N,
end-code

```

## BEQZ, as label --

Branchement si égal à zéro. Mot d'instruction BRI12.

**BEQZ**, effectue un saut si le registre d'adresse as est égal à zéro. **BEQZ**, fournit 12 bits de plage cible au lieu des huit bits disponibles dans la plupart des branchements conditionnels. L'adresse d'instruction cible de la branche est donnée par l'adresse de l'instruction **BEQZ**, plus le champ imm12 de 12 bits à signe étendu de l'instruction plus quatre. Si le registre **as** n'est pas égal à zéro, l'exécution continue avec l'instruction séquentielle suivante. L'inverse de **BEQZ**, est **BNEZ**,.

## CALL0, addr --

Réalise un appel vers un sous-programme pointé par addr.

```

forth definitions
asm xtensa

variable myVarTest
10 myVarTest !

\ for macros, see:
\
https://github.com/MPETREMANN11/ESP32forth/blob/main/assembler/xtensaMacros.txt
code __my@ ( -- SUBRaddr )    \ EXEC: leave subroutine address on stack
  a1 32      ENTRY,
  a8        arPOP,
  a9 a8 0    L32I.N,
  a9        arPUSH,
            RETW.N,
end-code

code my@ ( addr -- n )
  ' __my@ cell+ @
            CALL0,
            RETW.N,
end-code

myVarTest my@    \ display: 10

```



## **ENTRY, as n --**

Entrée de sous-programme. Mot d'instruction. (BRI12).

**ENTRY**, est destiné à être la première instruction de tous les sous-programmes appelés avec **CALL4**, **CALL8**, **CALL12**, **CALLX4**, **CALLX8**, ou **CALLX12**. Cette instruction n'est pas destinée à être utilisée par une routine appelée par **CALL0** ou **CALLX0**. **ENTRY**, a deux objectifs :

1. Tout d'abord, il incrémente le pointeur de la fenêtre de registre (WindowBase) du montant demandé par l'appelant (tel qu'enregistré dans le champ PS.CALLINC).
2. Deuxièmement, il copie le pointeur de pile de l'appelant à l'appelé et alloue le cadre de pile de l'appelé. L'opérande *as* spécifie le registre du pointeur de pile; il doit spécifier l'un des registres a0..a3 ou l'opération de **ENTRY**, n'est pas définie. Il est lu avant que la fenêtre ne soit déplacée, la taille du cadre de pile est soustraite, puis le registre *as* dans la fenêtre déplacée est écrit.

La taille du cadre de pile est spécifiée sous la forme du champ imm12 non signé de 12 bits en unités de huit octets. La taille est étendue à zéro, décalée vers la gauche de 3 et soustraite du pointeur de pile de l'appelant pour obtenir le pointeur de pile de l'appelé. Par conséquent, des trames de pile jusqu'à 32760 octets peuvent être spécifiées. La taille initiale du cadre de pile doit être une constante, mais par la suite, l'instruction MOVSP peut être utilisée pour allouer des objets de taille dynamique sur la pile ou pour étendre davantage un cadre de pile constant supérieur à 32760 octets.

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

## **EXTW, --**

Attente externe. Mot d'instruction. (RRR).

## **IDTLB, as --**

Invalide l'entrée de données TLB. Mot d'instruction. (RRR).

## **J, label --**

Saut inconditionnel. Mot d'instruction (APPEL).

**J**, effectue un branchement inconditionnel vers l'adresse cible. Il utilise un décalage relatif au PC 18 bits signé pour spécifier l'adresse cible. L'adresse cible est donnée par l'adresse de l'instruction **J**, plus le champ de décalage de 18 bits étendu par signe de l'instruction plus quatre, donnant une plage de -131068 à +131075 octets.

**JX, as --**

Registre de saut inconditionnel. Mot d'instruction (CALLX).

**JX**, effectue un saut inconditionnel à l'adresse dans le registre as.

**L32I.N, at as 0..60 --**

Chargement court 32 bits. Mot d'instruction (RRRN).

**L32I.N**, est un chargement 32 bits à partir de la mémoire. Il forme une adresse virtuelle en ajoutant le contenu du registre d'adresse as et une valeur constante étendue à zéro de 4 bits codée dans le mot d'instruction décalé de deux à gauche. Par conséquent, le décalage peut spécifier des multiples de quatre de zéro à 60. Trente-deux bits (quatre octets) sont lus à partir de l'adresse physique. Ces données sont ensuite écrites dans le registre d'adresses at.

```
forth
DEFINED? code invert [IF] xtensa-assembler [THEN]

forth definitions
asm xtensa

code my+
  a1 32 ENTRY,
    sp--,
  a7 a2 0      L32I.N,
  a8 a2 1      L32I.N,
  a7 a8 a9     ADD,
  a9 a2 0 S32I.N,
  RETW.N,
end-code

4 9 my+ . \ display 13
```

**L32R, at offset --**

Chargement 32 bits relatif par rapport à PC.

```
forth definitions

asm xtensa

: define: ( comp: n -- | exec: -- addr )
  chere value
  code4,
;

$87654321 define: val01

: addr2offset ( addr -- offset )
  chere - 4 /
;

code myL32R
  a1 32          ENTRY,
```

```

val01 addr2offset
a8 swap          L32R,
a8              arPUSH,
                RETW.N,
end-code

```

## **LOOP, ar as at --**

Gestion de boucle.

Instruction non utilisable directement. Voir les macros **For**, et **Next**,.

## **MAX, ar as at --**

Valeur maximum. Mot d'instruction (RRR).

**MAX**, calcule le maximum du contenu en complément à deux des registres d'adresses as et at et écrit le résultat dans le registre d'adresses ar.

## **MAXU, ar as at --**

Valeur maximum. Mot d'instruction (RRR).

**MAXU**, calcule le maximum du contenu non signé des registres d'adresse as et at et écrit le résultat dans le registre d'adresse ar.

## **MIN, ar as at --**

Valeur minimum. Mot d'instruction (RRR).

**MIN**, calcule le minimum du contenu en complément à deux des registres d'adresses as et at et écrit le résultat dans le registre d'adresses ar.

## **MINU, ar as at --**

Valeur minimum. Mot d'instruction (RRR).

**MINU**, calcule le minimum du contenu non signé des registres d'adresses as et at, et écrit le résultat dans le registre d'adresses ar.

## **MOV, ar as --**

Déplacement. Mot d'instruction (RRR).

**MOV**, est une macro assembleur qui utilise l'instruction OR, pour déplacer le contenu du registre d'adresse quant au registre d'adresse ar. L'entrée de l'assembleur **ar comme MOV**, s'étend dans **ar comme OU**, ar et as ne doivent pas spécifier le même registre en raison de la restriction **MOV.N**,.

## **MOVF, bt as ar --**

Déplace si faux.

**MOVF**, déplace le contenu du registre d'adresses **as** vers le registre d'adresses **ar** si le registre booléen **bt** est faux. Le registre d'adresse **ar** reste inchangé si le registre booléen **bt** est vrai. L'inverse de **MOVF**, est **MOVT**,.

## **MOVI, at n[-2048..2047] --**

Déplacement Immédiat.

**MOVI**, définit le registre d'adresse **at** avec une constante dans la plage -2048..2047 codée dans le mot d'instruction.

```
code mySRA ( n -- n' )
  a1 32      ENTRY,
  a8 1        MOVI,      \ a8 = 1
  a8 SAR      WSR,      \ SAR = 1
  a8          arPOP,     \ a8 = n
  a8 a8       SRA,      \ a8 = a8/2
  a8          arPUSH,    \ push result on stack
                      RETW.N,
end-code
```

## **MOVT, bt as ar --**

Déplace si vrai.

**MOVT**, déplace le contenu du registre d'adresses **as** vers le registre d'adresses **ar** si le registre booléen **bt** est vrai. Le registre d'adresse **ar** reste inchangé si le registre booléen **bt** est faux. L'inverse de **MOVT**, est **MOVF**,.

## **MULL, ar as at --**

Multiplication basse. Mot d'instruction (RRR).

**MULL**, effectue une multiplication sur 32 bits du contenu des registres d'adresse **as** et **at**, et écrit les 32 bits les moins significatifs du produit dans le registre d'adresse **ar**. Étant donné que les bits de produit les moins significatifs ne sont pas affectés par le multiplicande et le signe multiplicateur, **MULL**, est utile pour la multiplication signée et non signée.

```
forth
DEFINED? code invert [IF] xtensa-assembler [THEN]

code myEXP2
  a1 32 ENTRY,
  a8 a2 0 L32I.N,
  a8 a8 a8 MULL,
  a8 a2 0 S32I.N,
  RETW.N,
end-code
```

```
25 myEXP2 .  
\ display: 625
```

## NEG, at ar --

Négatif.  $AR[r] \leftarrow 0 - AR[t]$

**NEG**, calcule la négation du complément à deux du contenu du registre d'adresse at et l'écrit dans le registre d'adresse ar. Le débordement arithmétique n'est pas détecté.

```
\ for macros, see:  
\ https://github.com/MPETREMANN11/ESP32forth/blob/main/assembler/xtensaMacros.txt  
code myNEG  
  a1 32      ENTRY,  
  a8      arPOP,  
  a8 a9      NEG,  
  a9      arPUSH,  
          RETW.N,  
end-code  
  
10 myNEG .    \ display: -10  
-12 myNEG .   \ display: 12
```

## NOP, --

Pas d'opération Mot d'instruction (RRR)

Cette instruction n'effectue aucune opération. Elle est généralement utilisée pour l'alignement des instructions. **NOP**, est une instruction 24 bits. Pour une version 16 bits, voir **NOP.N**,.

## NOP.N, --

Pas d'opération Mot d'instruction (RRRN)

Cette instruction n'effectue aucune opération. Elle est généralement utilisée pour l'alignement des instructions. **NOP.N**, est une instruction 16 bits. Pour une version 24 bits, voir **NOP**,.

## OR, ar as at --

OU logique au niveau du bit. Mot d'instruction (RRR).

**OR**, calcule le ou logique au niveau du bit des registres d'adresse au fur et à. Le résultat est écrit dans le registre d'adresse ar.

## QUOS, at as ar --

Quotient signé. Mot d'instruction (RRR).

**QUOS**, effectue une division en complément à deux sur 32 bits du contenu du registre d'adresse as par le contenu du registre d'adresse at et écrit le quotient dans le registre d'adresse ar. L'ambiguïté qui existe lorsque le registre d'adresse as ou le registre d'adresse at est négatif est résolue en exigeant que le produit du quotient et du registre d'adresse at soit plus petit en valeur absolue que le registre d'adresse as. Si le contenu du registre d'adresse à est égal à zéro, QUOS lève une exception Integer Divide by Zero au lieu d'écrire un résultat. Le débordement (-2147483648 divisé par -1) n'est pas détecté.

```
\ for macros, see:
\
https://github.com/MPETREMANN11/ESP32forth/blob/main/assembler/xtensaMacros.txt
code my/MOD ( n1 n2 -- rem quot )
  a1 32      ENTRY,
  a7 arPOP,   \ divider in a7
  a8 arPOP,   \ value to divide in a8
  a7 a8 a9    REMS,   \ a9 = a8 MOD a7
  a9 arPUSH,
  a7 a8 a9    QUOS,   \ a9 = a8 / a7
  a9 arPUSH,
              RETW.N,
end-code
```

## **REMS**, at as ar --

Reste signé. Mot d'instruction (RRR).

**REMS**, effectue une division en complément à deux sur 32 bits du contenu du registre d'adresse as par le contenu du registre d'adresse at et écrit le reste dans le registre d'adresse ar. L'ambiguïté qui existe lorsque l'un ou l'autre registre d'adresse as ou registre d'adresse at est négatif est résolue en exigeant que le reste ait le même signe que registre d'adresse as. Si le contenu du registre d'adresse à est égal à zéro, REMS lève une exception Integer Divide by Zero au lieu d'écrire un résultat.

```
\ for macros, see:
\
https://github.com/MPETREMANN11/ESP32forth/blob/main/assembler/xtensaMacros.txt
code my/MOD ( n1 n2 -- rem quot )
  a1 32      ENTRY,
  a7 arPOP,   \ divider in a7
  a8 arPOP,   \ value to divide in a8
  a7 a8 a9    REMS,   \ a9 = a8 MOD a7
  a9 arPUSH,
  a7 a8 a9    QUOS,   \ a9 = a8 / a7
  a9 arPUSH,
              RETW.N,
end-code
```

## **RET**, --

Retour sans fenêtre. Mot d'instruction (CALLX).

**RET**, revient d'une routine appelée par **CALL0**, ou **CALLX0**. Elle est équivalente à l'instruction **JX**. **RET**, existe en tant qu'instruction distincte car certaines implémentations Xtensa ISA peuvent bénéficier d'avantages en termes de performances en traitant cette opération comme un cas particulier.

## **RET.N**, --

**RET.N**, est identique à **RET**, dans un codage 16 bits. **RET**, revient d'une routine appelée par **CALL0**, ou **CALLX0**.

## **RETW.N**, --

Retour fenêtré court. Mot d'instruction (RRRN).

**RETW.N**, est identique à **RETW**, dans un codage 16 bits. **RETW.N** renvoie d'un sous-programme appelé par **CALL4**, **CALL8**, **CALL12**, **CALLX4**, **CALLX8**, ou **CALLX12**, et qui avait **ENTRY**, comme première instruction.

```
forth
DEFINED? code invert [IF] xtensa-assembler [THEN]

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

## **ROUND.S**, ...

Arrondi simple à fixe.

**ROUND.S** convertit le contenu du registre à virgule flottante fs du format simple précision au format entier signé, en arrondissant vers le plus proche. La valeur simple précision est d'abord mise à l'échelle par une puissance de deux valeurs constantes codées dans le champ t, avec 0..15 représentant 1.0, 2.0, 4.0, ..., 32768.0. La mise à l'échelle permet une notation à virgule fixe où le point binaire est à l'extrémité droite de l'entier pour t = 0 et se déplace vers la gauche à mesure que t augmente jusqu'à ce que pour t = 15, il y ait 15 bits fractionnaires représentés dans le nombre à virgule fixe.

## **RSR**, at **SR[0..255]** --

Lecture registre spécial.

Le contenu du registre spécial désigné par 8 bits est écrit dans le registre d'adresse at.

```
forth definitions
asm xtensa

3 constant SAR      \ SAR = Shift-amount register - Special Register Number 3
```

```

\ for macros, see:
\
https://github.com/MPETREMAN11/ESP32forth/blob/main/assembler/xtensaMacros.txt
code writeSAR ( n[0..31] -- )
  a1 32      ENTRY,
  a8        arPOP,
  a8 SAR     WSR,
            RETW.N,
end-code

code readSAR ( -- n )
  a1 32      ENTRY,
  a8 SAR     RSR,
  a8        arPUSH,
            RETW.N,
end-code

2 writeSAR readSAR . \ display: 2
3 writeSAR readSAR . \ display: 3

```

### S32I.N, at as 0..60 --

Stockage proche 32 bits. Instruction de mots (RRRN).

**S32I.N**, est un stockage 32 bits en mémoire. Il forme une adresse virtuelle en ajoutant le contenu du registre d'adresse as et une valeur constante étendue à zéro de 4 bits codée dans le mot d'instruction décalé de deux à gauche. Par conséquent, le décalage peut spécifier des multiples de quatre de zéro à 60. Les données à stocker sont extraites du contenu du registre d'adresse at et écrites dans la mémoire à l'adresse physique.

```

forth
DEFINED? code invert [IF] xtensa-assembler [THEN]

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

```

### SEXT, ar as 7..22 --

Extension de signe. Mot d'instruction (RRR).

**SEXT**, prend le contenu du registre d'adresse as et réplique le bit spécifié par son opérande immédiat (dans la plage 7 à 22) sur les bits de poids fort et écrit le résultat dans le registre d'adresse ar. L'entrée peut être considérée comme une valeur imm + 1 bit avec les bits de poids fort non pertinents et cette instruction produit l'extension de signe 32 bits de cette valeur.



## SLLI, ar as 1..31 --

Décalage logique immédiat vers la gauche. Mot d'instruction (RRR).

Décale le contenu du registre d'adresse à gauche d'une quantité constante dans la plage 1..31 codée dans l'instruction.

```
\ Store 32 bits literal value in at register
: 32movi, { atReg 32imm -- }
  32imm
  $100 /mod      \ split 32 byte value in 4 bytes
  $100 /mod
  $100 /mod { b0 b1 b2 b3 }
  atReg atReg 32    SLLI,
  atReg atReg b3    ADDI,
  atReg atReg 8     SLLI,
  atReg atReg b2    ADDI,
  atReg atReg 8     SLLI,
  atReg atReg b1    ADDI,
  atReg atReg 8     SLLI,
  atReg atReg b0    ADDI,
;
\ Example:
\   variable SCORE
\
\ and in code definition:
\   a7 SCORE 32movi,
\ now a7 can used for memory pointer
```

## SRA, ar at --

Décalage arithmétique à droite.

**SRA**, décale arithmétiquement le contenu du registre d'adresse **a** à droite, en insérant le signe de **a** à gauche, du nombre de positions de bit spécifié par SAR (registre de quantité de décalage) et écrit le résultat dans le registre d'adresses **ar**.

```
forth definitions
asm xtensa

3 constant SAR      \ SAR = Shift-amount register - Special Register Number 3

code mySRA ( n -- n' )
  a1 32             ENTRY,
  a8 1              MOVI,          \ a8 = 1
  a8 SAR            WSR,          \ SAR = 1
  a8                arPOP,        \ a8 = n
  a8 a8             SRA,          \ a8 = a8/2
  a8                arPUSH,       \ push result on stack
                        RETW.N,
end-code

6 mySRA .          \ display 3
-8 mySRA .         \ display -4
```

## **SRAI, ar at 0..31 --**

Décalage Arithmétique à droite avec valeur Immédiate.

**SRAI**, décale arithmétiquement le contenu du registre d'adresse **at** à droite, en insérant le signe de **at** à gauche, d'une quantité constante codée dans la plage 0..31.

```
forth definitions
asm xtensa

\ calculate the average of two values
code AVG ( n1 n2 -- n3 )      \ n3 = ( n1 + n2 ) / 2
  a1 32          ENTRY,
  a8             arPOP,          \ a8 = n2
  a9             arPOP,          \ a9 = n1
  a8 a9 a8       ADD,
  a8 a8 1        SRAI,
  a8             arPUSH,
                RETW.N,
end-code

\ for tests
10 20 AVG .      \ display 15
-10 20 AVG .     \ display 5
-10 -20 AVG .    \ display -15
```

## **SRLI, ar at 0..15 --**

Décalage logique immédiat vers la droite. Mot d'instruction (RRR).

**SRLI**, décale le contenu du registre d'adresse à droite, en insérant des zéros à gauche, d'une quantité constante codée dans le mot d'instruction dans la plage 0..15. Il n'y a pas de **SRLI**, pour les équipes  $\geq 16$ .

## **SSA8B, as --**

Définissez la quantité de décalage pour BE Byte Shift. Mot d'instruction (RRR).

## **WSR, at SR[0..255] --**

Ecrit dans un registre spécial.

Le contenu du registre d'adresses at est écrit dans le registre spécial désigné par le champ SR défini dans l'intervalle 0..255.

```
forth definitions
asm xtensa

3 constant SAR      \ SAR = Shift-amount register - Special Register Number 3

\ for macros, see:
\
https://github.com/MPETREMANN11/ESP32forth/blob/main/assembler/xtensaMacros.txt
code writeSAR ( n[0..31] -- )
  a1 32          ENTRY,
```

```

    a8      arPOP,
    a8 SAR   WSR,
            RETW.N,
end-code

code readSAR ( -- n )
    a1 32    ENTRY,
    a8 SAR   RSR,
    a8      arPUSH,
            RETW.N,
end-code

2 writeSAR readSAR . \ display: 2
3 writeSAR readSAR . \ display: 3

```

## XOR, at as ar --

OU eXclusif booléen.

**XOR**, calcule le ou exclusif logique au niveau du bit des registres d'adresses **as** et **at**. Le résultat est écrit dans le registre d'adresses **ar**.