# ESP32forth

# Reference manual

**version 1.0 - 17 novembre 2023**


## Author


- Marc PETREMANN petremann@arduino-forth.com

# Contents

# forth

### ! n addr --

Store n to address.

```
VARIABLE TEMPERATURE
32 TEMPERATURE !
```

### # d1 -- d2

Perform a division modulo the current numeric base and transform the rest of the division into a string of characters. The character is dropped in the buffer set to running <#

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

### #! --

Behaves like \ for ESP32forth.

Serves as a text file header to indicate to the operating system (Unix-like) that this file is not a binary file but a script (set of commands). On the same line is specified the interpreter allowing to execute this script.

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

### #> n -- addr len

Drop n. Make the pictured numeric output string available as a character string. *addr* and *len* specify the resulting character string.

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

### #FS r --

Converts a real number to a string. Used by f.

### #s d1 -- d=0

Converts the rest of d1 to a string in the character string initiated by <#.

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

### #tib -- n

Number of characters received in terminal input buffer.

### ' exec: <space>name -- xt

Skip leading space delimiters. Parse name delimited by a space. Find name and return xt, the execution token for name. When interpreting, `' xyz EXECUTE` is equivalent to `xyz`.

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

### (local) a n --

Word used to manage the creation of local variables.

### * n1 n2 -- n3

Integer multiplication of two numbers.

```
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

Multiply n1 by n2 producing the intermediate double-cell result d. Divide d by n3 giving the single-cell quotient n4.

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

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

Multiply n1 by n2 producing the intermediate double-cell result d. Divide d by n3 producing the single-cell remainder n4 and the single-cell quotient n5.

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

### + n1 n2 -- n3

Leave sum of n1 n2 on stack.

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

### +! n addr --

Increments the contents of the memory address pointed to by addr.

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

### +loop n --

Increment index loop with value n. Mark the end of a loop `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>

add n to the content of *valname*

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

### , x --

Append x to the current data section.

### - n1 n2 -- n1-n2

Subtract two integers.

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

### -rot n1 n2 n3 -- n3 n1 n2

Inverse stack rotation. Same action than `rot rot`

### . n --

Remove the value at the top of the stack and display it as a signed single precision integer.

```
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>

The word ." can only be used in a compiled definition. At runtime, it displays the text between this word and the delimiting " character end of string.

```
: 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 --

Displays the content of the data stack, with no action on the content of this stack.

### / n1 n2 -- n3

Divide n1 by n2, giving the single-cell quotient n3.

```
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

Divide n1 by n2, giving the single-cell remainder n3 and the single-cell quotient n4.

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

### 0< x1 --- fl

Test if x1 is less than zero.

### 0<>  n -- fl

Leave -1 if n <> 0

### 0=  x -- fl

flag is true if and only if x is equal to zero.

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

### 1+  n -- n+1

Increments the value at the top of the stack.

### 1-  n -- n-1

Decrements the value at the top of the stack.

### 1/F  r -- r'

Performs a 1/r operation.

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

### 2!  d addr --

Store double precision value in memory address addr.

### 2*  n -- n*2

Multiply n by two.

### 2/  n -- n/2

Divide n by two.

n/2 is the result of shifting n one bit toward the least-significant bit, leaving the most-significant bit unchanged

```
24 2/ .    \ display    12
25 2/ .    \ display    12
26 2/ .    \ display    13
```

### 2@  addr -- d

Leave on stack double precision value d stored at address addr.

### 2drop  n1 n2 n3 n4 -- n1 n2

Removes the double-precision value from the top of the data stack.

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

### 2dup n1 n2 -- n1 n2 n1 n2

Duplicates the double precision value n1 n2.

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

### 4* n -- n*4

Multiply n by four.

### 4/ n -- n/4

Divide n by four.

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

Skip leading space delimiters. Parse name delimited by a space. Create a definition for name, called a "colon definition". Enter compilation state and start the current definition.

Subsequent execution of **NOM** performs the execution sequence words compiled in his "colon" definition. After **: NOM**, the interpreter enters compile mode. All non-immediate words are compiled in the definition, the numbers are compiled in literal form. Only immediate words or placed in square brackets (words **[** and **]**) are executed during compilation to help control it. A "colon" definition remains invalid, ie not inscribed in the current vocabulary, as long as the interpreter did not execute **;** (semi-colon).

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

### :noname -- cfa-addr

Define headerless forth code. cfa-addr is the code execution of a definition.

```
:noname s" Saterday" ;
: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
```

### ; --

Immediate execution word usually ending the compilation of a "colon" definition.

```
: NAME
    nomex1 nomex2
    nomexn ;
```

### < n1 n2 -- fl

Leave fl true if n1 < n2

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

### <# n --

Marks the start of converting a integer number to a string of characters.

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

```
  ;

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

### <=  n1 n2 -- fl

Leave fl true if n1 <= n2

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

### <>  x1 x2 -- fl

flag is true if and only if x1 is different x2.

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

### =  n1 n2 -- fl

Leave fl true if n1 = n2

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

### >  x1 x2 -- fl

Test if x1 is greater than x2.

### >=  x1 x2 -- fl

flag is true if and only if x1 is equal x2.

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

### >**body**  cfa -- pfa

pfa is the data-field address corresponding to cfa.

### >**flags**  xt -- flags

Convert cfa address to flags address.

### >**in**  -- addr

Number of characters consumed from TIB

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

### >link cfa -- cfa2

Converts the cfa address of the current word into the cfa address of the word previously defined in the dictionary.

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

### >link& cfa -- lfa

Transforms the execution address of the current word into the link address of this word. This link address points to the cfa of the word defined before this word. Used by `>link`

### >name cfa -- nfa len

finds the name field address of a token from its code field address.

```
' 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

Transfers n to the return stack. This operation must always be balanced with `r>`

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

### ? addr -- c

Displays the content of any variable or address.

### ?do n1 n2 --

Executes a `do loop` or `do +loop` loop if n1 is strictly greater than 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

Duplicate n if n is not nul.

### @ addr -- n

Retrieves the integer value n stored at address addr.

```
TEMPERATURE @
```

### abort --

Raises an exception and interrupts the execution of the word and returns control to the interpreter.

### abs n -- n'

Return the absolute value of n.

```
-7 abs .    \ display 7
```

### accept addr n -- n

Accepts n characters from the keyboard (serial port) and stores them in the memory area pointed to by 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 for `analogRead`

### afliteral r:r --

Compiles a real number. Used by `fliteral`

### aft --

Jump to THEN in FOR-AFT-THEN-NEXT loop 1st time through.

```
: 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 --

Mark the end on an infinit loop of type `begin ... again`

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

### align --

Align the current data section dictionary pointer to cell boundary.

### aligned addr1 -- addr2

addr2 is the first aligned address greater than or equal to addr1.

### allot n --

Reserve n address units of data space.

### also --

Duplicate the vocabulary at the top of the vocabulary stack.

### analogRead pin -- n

Analog read from 0-4095. Use to read analog value. `analogRead` has only one argument which is a pin number of the analog channel you want to use.

```
\ 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

Execute logic AND.

The words `AND`, `OR`, and `XOR` perform operations binary **bitwise** logic on single-precision integers at the top of the data stack.

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

### ansi --

Selects the `ansi` vocabulary.

### ARSHIFT x1 u -- x2

Arithmetic right shift of u

### asm --

Select the `asm` vocabulary.

### assembler --

Alias for `asm`. Select the `asm` vocabulary.

### assert fl --

For tests and asserts.

### at-xy x y --

Positions the cursor at the x y coordinates.

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

### base -- addr

Single precision variable determining the current numerical base. The `BASE` variable contains the value 10 (decimal) when FORTH starts.

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

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

### begin --

Mark start of a structure `begin..until`, `begin..again` or `begin..while..repeat`

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

```
  ;
```

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

Selects the background display color. The color is in the range 0..255 in decimal.

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

### BIN mode -- mode'

Modify a file-access method to include BINARY.

### BINARY --

Select binary base.

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

### bl -- 32

Value 32 on stack.

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

### blank addr len --

If len is greater than zero, store byte $20 (space) in each of len consecutive characters of memory beginning at addr.

### block n -- addr

Get addr 1024 byte for block n.

### block-fid -- n

Flag indicating the state of a block file.

### block-id -- n

Pointer to a block file.

### bluetooth --

Select `bluetooth` vocabulary.

### bterm --

Select `bterm` vocabulary.

### buffer n - addr

Get a 1024 byte block without regard to old contents.

### bye --

Word defined by `defer`. Execute by default `esp32-bye` (in voc. Internals).

### c! c addr --

Stores an 8-bit c value at address addr.

```
36 constant DDRB  \ data direction register for PORT B on Arduino
32 DDRB c!        \ same as 35 32 c!
```

### c, c --

Append c to the current data section.

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

### c@ addr -- c

Retrieves the 8-bit c value stored at address addr.

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

### camera --

Select `camera` vocabulary.

### camera-server --

Select `camera-server` vocabulary.

### CASE --

```
: 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>

Display the file content.

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

### catch cfa -- fl

Initializes an action to perform in the event of an exception triggered by `throw`.

### cell -- 4

Return number of bytes in a 32-bit integer.

### cell+ n -- n'

Increment `CELL` content.

### cell/ n -- n'

Divide `CELL` content.

### cells n -- n'

Multiply `CELL` content.

Allows you to position yourself in an array of integers.

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

### char -- <string>

Word used in interpretation only. Leave the first character of the string following this word.

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

### chunk-size -- 2048

Constant. Value 2048.

### CLOSE-FILE fileid -- ior

Close an open file.

### cmove c-addr1 c-addr2 u --

If u is greater than zero, copy u consecutive characters from the data space starting at c-addr1 to that starting at c-addr2, proceeding character-by-character from lower addresses to higher addresses.

### code -- <:name>

Defines a word whose definition is written in assembly language.

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

### constant comp: n -- <name> | exec: -- n

Define a constant.

```
binary  \ masks are 32 bit length
00000000000000000000000000000100 constant ledGREEN      \  green LED on G2
00000000010000000000000000000000 constant ledYELLOW     \ yellow LED on G21
00000000000000100000000000000000 constant ledRED        \    red LED on G17
decimal
```

### context -- addr

Pointer to pointer to last word of context vocabulary

### copy from to --

Copy contents of block 'from' to block 'to'

### cp -- "src" "dst"

Copy "src" file to "dst".

### cr --

Show a new line return.

```
: .result ( ---)
```

```
    ." Port analys result" cr
    . "pool detectors" cr ;
```

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

The word **CREATE** can be used alone.

The word after **CREATE** is created in the dictionary, here **DATAS**. The execution of the word thus created deposits on the data stack the memory address of the parameter zone. In this example, we have compiled 4 8-bit values. To recover them, it will be necessary to increment the address stacked with the value shifting the data to be recovered.

```
\ 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

Create a file on disk, returning a 0 ior for success and a file id.

### current -- cfa

Pointer to pointer to last word of current vocabulary

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

### dacWrite n1 n0 --

Write to DAC, n1=pin, n0=value [0..255]

The ESP32 features two 8-bit digital-to-analog converters (DACs), which makes it possible to produce a true analog signal, i.e. a voltage that can take any value between 0 and 3.3V. Two pins can be used as analog output: GPIO 25 and GPIO 26. For example, to set GPIO pin 25 to a value of 1 volt, you write:

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

### DECIMAL --

Selects the decimal number base. It is the default digital base when FORTH starts.

```
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>

Define a deferred execution vector. `vec-name` execute the word whose execution token is stored in vec-name's data space.

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

### DEFINED? -- <word>

Returns a non-zero value if the word is defined. Otherwise returns 0.

```
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 --

Make the compilation word list the same as the first word list in the search order. Specifies that the names of subsequent definitions will be placed in the compilation word list. Subsequent changes in the search order will not affect the compilation word list.

```
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

Delete a named file from disk, and return ior=0 on success.

### depth -- n

n is the number of single-cell values contained in the data stack before n was placed on the stack.

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

### digitalRead n -- x

Read state of GPIO pin.

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

### digitalWrite pin value --

Set GPIO pin state.

```
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 --

Set up loop control parameters with index n2 and limit n1.

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

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

The word **CREATE** can be used in a new word creation word... Associated with **DOES>**, we can define words that say how a word is created then executed.

### drop n --

Removes the single-precision integer that was there from the top of the data stack.

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

### dump a n --

Dump a memory region

This version is not very interesting. Prefer this version: DUMP tool for ESP32Forth

### dump-file addr len addr2 len2 --

Transfers the contents of a text string addr len to a file pointed by addr2 len2

The content of the */spiffs/autoexec.fs* file is automatically interpreted and/or compiled when ESP32Forth starts. This feature can be leveraged to set up WiFi access when starting ESP32Forth by injecting the access parameters like this:

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

### dup n -- n n

Duplicates the single-precision integer at the top of the data stack.

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

### echo -- addr

Variable. Value is -1 by default. If 0, commands are not displayed.

```
: 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 --

Select `editor`.

- `l` lists the content of the current block
- `n` select the next block

- `p` select the previous block
- `wipe` empties the content of the current block
- `d` delete line n. The line number must be in the range 0..14. The following lines go up. Example: `3 D` erases the content of line 3 and brings up the content of lines 4 to 15.
- `e` erases the content of line n. The line number must be in the range 0..15. The other lines do not go up.
- `a` inserts a line n. The line number must be in the range 0..14. The lines located after the inserted line come down. Example: `3 A test` inserts **test** on line 3 and move the contents of lines 4 to 15.
- `r` replaces the content of line n. Example: `3 R test` replace the contents of line 3 with **test**

### else --

Word of immediate execution and used in compilation only. Mark a alternative in a control structure of the type `IF ... ELSE ... THEN`

At runtime, if the condition on the stack before `IF` is false, there is a break in sequence with a jump following `ELSE`, then resumed in sequence after `THEN`.

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

### emit x --

If x is a graphic character in the implementation-defined character set, display x. The effect of `EMIT` for all other values of x is implementation-defined.

When passed a character whose character-defining bits have a value between hex 20 and 7E inclusive, the corresponding standard character is displayed. Because different output devices can respond differently to control characters, programs that use control characters to perform specific functions have an environmental dependency. Each `EMIT` deals with only one character.

```
65 emit    \ display A
66 emit    \ display B
```

### empty-buffers --

Empty all buffers.

### ENDCASE --

Marks the end of a `CASE OF ENDOF ENDCASE` structure

```
: 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 --

Marks the end of a `OF .. ENDOF` choice in the control structure between `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 --

If len is greater than zero, store byte $00 in each of len consecutive characters of memory beginning at addr.

### ESP --

Select `ESP` vocabulary.

### ESP32-C3? -- -1|0

Stacks -1 if the card is ESP32-C3.

### ESP32-S2? -- -1|0

Stacks -1 if the card is ESP32-S2.

### ESP32-S3? -- -1|0

Stacks -1 if the card is ESP32-S3.

### ESP32? -- -1|0

Stacks -1 if the card is ESP32.

### evaluate addr len --

Evaluate the content of a string.

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

### EXECUTE addr --

Execute word at addr.

Take the execution address from the data stack and executes that token. This powerful word allows you to execute any token which is not a part of a token list.

### exit --

Aborts the execution of a word and gives back to the calling word.

Typical use: `: X ... test IF ... EXIT THEN ... ;` At run time, the word `EXIT` will have the same effect as the word `;`

### extract n base -- n c

Extract the least significant digit of n. Leave on the stack the quotient of n/base and the ASCII character of this digit.

### F* r1 r2 -- r3

Multiplication of two real numbers.

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

### F** r_val r_exp -- r

Raises a real r_val to the power 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 of two real numbers.

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

### F- r1 r2 -- r3

Subtraction of two real numbers.

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

### f. r --

Displays a real number. The real number must come from the real stack.

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

### f.s --

Display content of reals stack.

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

### F/ r1 r2 -- r3

Division of two real numbers.

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

### F0< r -- fl

Tests if a real number is less than zero.

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

### F0= r -- fl

Indicates true if the real is null.

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

### f< r1 r2 -- fl

fl is true if r1 < r2

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

### f<= r1 r2 -- fl

fl is true if r1 <= 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 is true if r1 <> r2.

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

### f= r1 r2 -- fl

fl is true if r1 = r2.

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

### f> r1 r2 -- fl

fl is true if r1> r2.

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

### f>= r1 r2 -- fl

fl is true if r1> = 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

Convert a real to an integer. Leaves the integer part on the data stack if the real has fractional parts.

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

### FABS r1 -- r1'

Returns the absolute value of a real number.

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

### FATAN2 r-tan -- r-rad

Calculates the angle in radians from the tangent.

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

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

Defines a constant of type real.

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

### FCOS r1 -- r2

Calculates the cosine of an angle expressed in radians.

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

### fdepth -- n

n is the number of reals values contained in the real stack.

### FDROP r1 --

Drop real r1 from real stack.

### FDUP r1 -- r1 r1

Duplicate real r1 from real stack.

### FEXP ln-r -- r

Calculate the real corresponding to e EXP r

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

### fg color[0..255] --

Selects the text display color. The color is in the range 0..255 in decimal.

```
: testFG ( -- )
  256 0 do
```

```
    i fg ." X"
  loop ;
```

### file-exists? addr len --

Tests if a file exists. The file is designated by a character string.

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

### FILE-POSITION fileid -- ud ior

Return file position, and return ior=0 on success.

### FILE-SIZE fileid -- ud ior

Get size in bytes of an open file as a double number, and return ior=0 on success.

### fill addr len c --

If len is greater than zero, store c in each of len consecutive characters of memory beginning at addr.

### FIND addr len -- xt | 0

Find a word in dictionnary.

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

### fliteral r:r --

Immediate execution word. Compiles a real number.

### FLN r -- ln-r

Calculates the natural logarithm of a real number.

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

### FLOOR r1 -- r2

Rounds a real down to the integer value.

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

### flush --

Save and empty all buffers.

After editing the contents of a block file, running `flush` ensures that changes to the contents of blocks are saved.

### FLUSH-FILE fileid –- ior

Attempt to force any buffered information written to the file referred to by fileid to be written to mass storage. If the operation is successful, ior is zero.

### FMAX r1 r2 -- r1|r2

Let the greatest real of r1 or r2.

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

### FMIN r1 r2 -- r1|r2

Let the smaller real of r1 or r2.

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

### FNEGATE r1 -- r1'

Reverses the sign of a real number.

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

### FNIP r1 r2 -- r2

Delete second element on reals stack.

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

### for n --

Marks the start of a loop `for .. next`

WARNING: the loop index will be processed in the interval [n..0], i.e. n+1 iterations, which is contrary to the other versions of the FORTH language implementing FOR..NEXT (FlashForth).

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

### forget -- <name>

Searches the dictionary for a name following it. If it is a valid word, trim dictionary below this word. Display an error message if it is not a valid word.

### forth --

Select the `FORTH` vocabulary in the word search order to execute or compile words.

### forth-builtins -- cfa

Entry point of `forth` vocabulary.

### FOVER r1 r2 -- r1 r2 r1

Duplicate second real on reals stack.

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

### fp0 -- addr

Points to the bottom of ESP32Forth's reals stack (data stack).

### FP@ -- addr

Retrieves the stack pointer address of the reals.

### free a -- f

free memory previously reserved by `allocate`

### freq chan freq --

sets frequency freq n to channel chan. Use`ledcWriteTone`

### FSIN r1 -- r2

Calculates the sine of an angle expressed in radians.

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

### FSINCOS r1 -- rcos rsin

Calculates the cosine eand sine of an angle expressed in 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

Square root of a real number.

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

### FSWAP r1 r2 -- r1 r2

Reverses the order of the two values on the ESP32Forth real stack.

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

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

Defines a floating point variable.

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

### handler -- addr

Ticket for interruptions.

### here -- addr

Leave the current data section dictionary pointer.

The dictionary pointer is incremented as the words are compiled and variables and data tables are defined.

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

### HEX --

Selects the hexadecimal digital base.

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

### HIGH -- 1

Constant. Defines the active state of a pin.

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

```
    ;
```

### hld -- addr

Pointer to text buffer for number output.

### hold c --

Inserts the ASCII code of an ASCII character into the character string initiated by <#.

### httpd --

Select `httpd` vocabulary-

### i -- n

n is a copy of the current loop index.

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

### if fl --

The word `IF` is executed immediately. `IF` marks the start of a control structure for type `IF..THEN` or `IF..ELSE..THEN`.

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

### immediate --

Make the most recent definition an immediate word.

Sets the compile-only lexicon bit in the name field of the new word just compiled. When the interpreter encounters a word with this bit set, it will not execute this word, but spit out an error message. This bit prevents structure words to be executed accidentally outside of a compound word.

### include -- <:name>

Loads the contents of a file designated by <name>.

The word **include** can only be used from the terminal. To load the contents of a file from another file, use the word **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 --

Loads the contents of a file from the SPIFFS filesystem, designated by a character string.

The word **included** can be used in a FORTH listing stored in the SPIFFS file system. For this reason, the filename to load should always be preceded by */spiffs/*

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

### included? addr len -- f

Tests whether the file named in the character string has already been compiled.

### INPUT -- 1

Constant. Value 1. Defines the direction of use of a GPIO register as an input.

### internals --

Select **internals** vocabulary.

### interrupts --

Select **interrupts** vocabulary.

### invert x1 -- x2

Complement to one of x1. Acts on 16 or 32 bits depending on the FORTH versions.

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

### is --

Affecte le code d'exécution d'un mot à un mot d'exécution vectorisée.

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

### j -- n

n is a copy of the next-outer loop index.

```
: 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 is a copy of the next-next-outer loop index.

```
: 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

Waits for a key to be pressed. Pressing a key returns its ASCII code.

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

### key? -- fl

Returns *true* if a key is pressed.

```
: keyLoop
```

```
    begin
    key? until
  ;
```

### L! n addr --

Store a value n.

```
hex
3ff44004 constant GPIO_OUT_REG

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

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

### latestxt -- xt

Stacks the execution code (cfa) address of the last compiled word.

```
: txtxtx  ;
latest
>name type  \ display txtxtx
```

### leave --

Prematurely terminates the action of a `do..loop` 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

Pin 2 value for LED on the board. Does not work with all cards.

### ledc --

Select `ledc` vocabulary.

### list n --

Displays the contents of block n.

### literal x --

Compiles the value x as a literal value.

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

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

### load n --

Evaluate a block.

`load` preceded by the number of the block you want to execute and/or compile the content. To compile the content of our block 0, we will execute `0 load`

### login z1 z2 --

Login to wifi only.

```
\ connection to local WiFi LAN
: myWiFiConnect
    z" Mariloo"
    z" 1925144D91DXXXXXXXXXX959F"
    login
    ;
myWiFiConnect
\ display:
\ 192.168.1.8
\ MDNS started
```

### loop --

Add one to the loop index. If the loop index is then equal to the loop limit, discard the loop parameters and continue execution immediately following the loop. Otherwise continue execution at the beginning of the loop.

### LOW -- 0

Constant. Defines the inactive state of a pin.

### ls -- "path"

Displays the contents of a file path.

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

### LSHIFT x1 u -- x2

Shift to the left of u bits by the value x1.

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

### max n1 n2 -- n1|n2

Leave the unsigned larger of u1 and u2.

### MDNS.begin name-z -- fl

Start multicast dns.

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

### min n1 n2 -- n1|n2

Leave min of n1 and n2

### mod n1 n2 -- n3

Divide n1 by n2, giving the single-cell remainder n3.

The modulo function can be used to determine the divisibility of one number by another.

```
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 --

Waiting in millisencondes.

For long waits, set a wait word in seconds.

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

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

```
12 seconds \ delay for 12 seconds
```

### MS-TICKS -- n

System ticks. One tick per millisecond. Useful for measuring the execution time of a definition.

### mv -- "src" "dest"

Rename "src" file to "dst".

### n. n --

Display anay value n in decimal format.

### negate n -- -n'

Two's complement of n.

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

### next --

Marks the end of a loop `for .. next`

### nip n1 n2 -- n2

Remove n1 from the stack.

### nl -- 10

Value 10 on stack.

### normal --

Disables selected colors for display.

### OCTAL --

Selects the octal digital base.

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

### OF n --

Marks a `OF .. ENDOF` choice in the control structure between `CASE ENDCASE` If the tested value is equal to the one preceding `OF`, the part of code located between `OF ENDOF` will be executed.

```
: 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 --

Displays the version of the FORTH ESP32forth language.

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

### oled --

Select `oled` vocabulary.

### only --

Reset context stack to one item, the FORTH dictionary Non-standard, as there's no distinct ONLY vocabulary

### open-blocks addr len --

Open a block file. The default blocks file is *blocks.fb*

### OPEN-FILE addr n opt -- n

Open a file. opt is one of the values `R/O` or `R/W` or `W/O`.

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

### OR n1 n2 -- n3

Execute logic OR.

The words `AND`, `OR`, and `XOR` perform operations binary **bitwise** logic on single-precision integers at the top of the data stack.

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

### order --

Print the vocabulary search order.

```
Serial
order   \ display Serial
```

### OUTPUT -- 2

Constant. Value 2. Defines the direction of use of a GPIO register as an output.

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

### over n1 n2 -- n1 n2 n1

Place a copy of n1 on top of the stack.

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

### page --

Erases the screen.

### PARSE c "string" -- addr count

Parse the next word in the input stream, terminating on character c. Leave the address and character count of word. If the parse area was empty then count=0.

### pause --

Yield to other tasks.

### pi -- r

PI constant.

```
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 of `digitalWrite`

### pinMode pin mode --

Set mode of GPIO. MODE = INPUT | OUTPUT

```
04 input pinmode            \ GO4 as an input
15 input pinmode            \ G15 as an input
```

### precision -- n

Pseudo constant determining the display precision of real numbers. Initial value 6.

If we reduce the display precision of real numbers below 6, the calculations will be when even performed with precision to 6 decimal places.

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

### prompt --

Displays an interpreter availability text. Default poster:
ok

### PSRAM? -- -1|0

Stacks -1 if PSRAM memory is available.

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

Creates a temporary counted string ended with "

### R/O -- 0

System constant. Stack 0.

### R/W -- 2

System constant. Stack 2.

### r> R: n -- S: n

Transfers n from the return stack. This operation must always be balanced with >r

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

### R@ -- n

Copies the contents of the top of the return stack onto the data stack.

### rdrop S: -- R: n --

Discard top item of return stack.

### READ-FILE a n fh -- n ior

Read data from a file. The number of character actually read is returned as u2, and ior is returned 0 for a successful read.

### recurse --

Append the execution semantics of the current definition to the current definition.

The usual example is the coding of the factorial function.

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

### registers --

Select `registers` vocabulary.

### remaining -- n

Indicates the remaining space for your definitions.

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

### remember --

Save a snapshot to the default file (./myforth or /spiffs/myforth on ESP32).

The word `REMEMBER` allows you to *freeze* the compiled code. If you compiled an application, run `REMEMBER`. Unplug the ESP32 board. Plug it back in. You should find your app. Use `STARTUP:` to set your application's password to run on startup.

### repeat --

End a indefinite loop `begin.. while.. repeat`

### REPOSITION-FILE ud fileid -- ior

Set file position, and return ior=0 on success

### required addr len --

Loads the contents of the file named in the character string if it has not already been loaded.

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

**rerun** t --

Rerun timer t triggering

**reset** --

Delete the default filename.

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

Set the size of the file to ud, an unsigned double number. After using **RESIZE-FILE**, the result returned by **FILE-POSITION** may be invalid

**restore** -- <:name>

Restore a snapshot from a file.

**revive** --

Restore the default filename.

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

Stacks -1 if the processor is RSIC-V.

**riscv-assembler** --

Loads and installs the **riscv** vocabulary. This word must be executed only once before the definition of words in RISC-V assembler.

**rm** -- "path"

Delete the file designed in file path.

**rmt** --

Select the **rmt** vocabulary.

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

Rotate three values on top of stack.

**rp0** -- addr

Points to the bottom of Forth's return stack (data stack).

**RSHIFT** x1 u -- x2

Right shift of the value x1 by u bits.

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

### rtos --

Select `rtos` vocabulary.

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

Creates a temporary counted string ended with |

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

In interpretation, leaves on the data stack the string delimited by " In compilation, compiles the string delimited by " When executing the compiled word, returns the address and length of the string...

```
\ 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

Converts an integer to a real number and transfers this real to the stack of reals.

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

### s>z a n -- z

Convert a counted string string to null terminated (copies string to heap)

### save -- <:name>

Saves a snapshot of the current dictionary to a file.

### save-buffers --

Save all buffers.

### SCR -- addr

Variable pointing to the block being edited.

### SD --

Select the `SD` vocabulary.

### SD_MMC --

Select `SD_MMC` vocabulary.

### see -- name>

Decompile a FORTH definition.

```
see include
: include  bl PARSE included ;

see space
: space  bl emit ;
```

### Serial --

Select serial vocabulary

### set-precision n --

Changes the display precision of Real numbers.

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

### SF! r addr --

Stores a real previously depoded on the real stack at the memory address addr.

### sf, r --

Compile a real number.

### SF@ addr -- r

Get the actual number stored at address addr, usually a variable defined by `fvariable`.

### sfloat -- 4

Constant. value 4.

### sfloat+ addr -- addr+4

Increments a memory address by the length of a real.

### sfloats n -- n*4

Calculate needed space for n reals.

### SMUDGE -- 2

Constant. Value 2.

### sockets --

Select `sockets` vocabulary.

### sp0 -- addr

Points to the bottom of Forth's parameter stack (data stack).

### SP@ -- addr

Push on stack the address of data stack.

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

### space --

Display one space.

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

### spaces n --

Displays the space character n times.

Defined since version 7.071

### SPI --

Select the `SPI` vocabulary.

List of `SPI` vocabulary words: `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 --

Select `SPIFFS` vocabulary.

### spi_flash --

Select the `spi_flash` vocabulary.

### start-task task --

Start a task.

### startup: -- <name>

Indicates the word that should run when ESP32forth starts after initialization of the general environment.

Here we have defined the word `myBoot` which displays a text on startup. To test the correct execution, you can type `bye` , which restart ESP32forth. You can also unplug the ESP32 board and plug it back in. This is this test that was carried out. Here is the result in the 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

Compilation state. State can only be changed by `[` and `]`.

-1 for compiling, 0 for interpreting

### str n -- addr len

Transforms any value n into an alphanumeric string, in the current numeric base.

### str= addr1 len1 addr2 len2 -- fl

Compare two strings. Leave true if they are identical.

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

### streams --

Select `streams` vocabulary.

### structures --

Select the `structures` vocabulary.

### swap n1 n2 -- n2 n1

Swaps values at the top of the stack.

```
2 5 SWAP
```

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

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

Create a new task with dsz size data stack and rsz size return stack running xt.

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

### tasks --

Select `tasks` vocabulary.

### telnetd --

Select `telnetd` vocabulary.

### then --

Immediate execution word used in compilation only. Mark the end a control structure of type `IF..THEN` or `IF..ELSE..THEN` .

### throw n --

Generates an error if n is not equal to zero.

If any bits of n are non-zero, pop the topmost exception frame from the exception stack, along with everything on the return stack above that frame. Then restore the input source specification in use before the corresponding CATCH and adjust the depths of all stacks defined by this standard so that they are the same as the depths saved in the exception frame (i is the same number as the i in the input arguments to the corresponding CATCH), put n on top of the data stack, and transfer control to a point just after the CATCH that pushed that exception frame.

```
: 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 --

Loads the contents of a block file, from block n1 to block n2.

### tib -- addr

returns the address of the the terminal input buffer where input text string is held.

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

### timers --

Select `timers` vocabulary.

### to n --- <valname>

`to` assign new value to *valname*

### tone chan freq --

sets frequency freq n to channel chan. Use `ledcWriteTone`

### touch -- "path"

Create "path" file if it doesn't exist.

### type addr c --

Display the string characters over c bytes.

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

### u. n --

Removes the value from the top of the stack and displays it as an unsigned single precision integer.

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

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

Unsigned int/int->int division.

### UL@ addr -- un

Retrieve a unsigned value.

**WARNING**: Previous versions of ESP32forth used the word `L@`.

### unloop --

Stop a do..loop action. Using `unloop` before `exit` only in a do..loop structure.

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

### until fl --

End of `begin.. until` structure.

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

### update --

Used for block editing. Forces the current block to the modified state.

### use -- <name>

Use "name" as the blockfile.

```
USE /spiffs/foo
```

### used -- n

Specifies the space taken up by user definitions. This includes already defined words from the FORTH dictionary.

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

Extracts the least significant 16 bits part of a memory zone pointed to by its unsigned 32-bit address.

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

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

Define value.

*valname* leave value on stack. A Value behaves like a Constant, but it can be changed.

```
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

Creation word. Defines a simple precision variable.

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

### visual --

Selects the `visual` vocabulary.

### vlist --

Display all words from a vocabulary.

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

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

Definition word for a new vocabulary. In 83-STANDARD, vocabularies are no longer declared to be executed immediately.

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

### W/O -- 1

System constant. Stack 1.

### web-interface --

Select `web-interface` vocabulary.

### webui z1 z2 --

Login and start webui.

```
z" Mariloo"                          \ SSID of your WiFi access
z" 1925144D91DXXXXXXXXXX959F"        \ 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 --

Mark the conditionnal part execution of a 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 --

Select `WiFi` vocabulary.

### Wire --

Select `Wire` vocabulary.

### words --

List the definition names in the first word list of the search order. The format of the display is implementation-dependent.

### WRITE-FILE a n fh -- ior

Write a block of memory to a file.

### XOR n1 n2 -- n3

Execute logic eXclusif OR.

The words **AND**, **OR**, and **XOR** perform operations binary **bitwise** logic on single-precision integers at the top of the data stack.

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

### xtensa-assembler --

Loads and installs the **xtensa** vocabulary. This word must be executed only once before the definition of words in XTENSA assembler.

```
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

Stacks -1 if the processor is XTENSA.

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

Compile zero terminated string into definition.

WARNING: these character strings marked with **z"** can only be used for specific functions, network for example.

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

### z>s z -- a n

Convert a null terminated string to a counted string.

### [ --

Enter interpretation state. **[** is an immediate word.

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

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

Use in compilation only. Immediate execution. Compile the cfa of <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

Place xchar, the value of the first xchar of name, on the stack.

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

### [ELSE] --

Mark a part of conditional sequence in **[IF] ... [ELSE] ... [THEN]**.

### [IF] fl --

Begins a conditional sequence of type **[IF] ... [ELSE]** or **[IF] ... [ELSE] ... [THEN]**.

If flag is 'TRUE' do nothing (and therefore execute subsequent words as normal). If flag is 'FALSE', parse and discard words from the parse area including nested instances of **[IF]**.. **[ELSE]**.. '[THEN]' and **[IF]**.. **[THEN]** until the balancing **[ELSE]** or **[THEN]** has been parsed and discarded.

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

### [THEN] --

Ends a conditional sequence of type **[IF] ... [ELSE]** or **[IF] ... [ELSE] ... [THEN]**.

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

### ] --

Return to compilation. ] is an immediate word.

With FlashForth, the words [ and ] allow you to use assembly code, subject to first compiling an assembler.

```
\ Load constant $1234 to top of stack
: a-number ( -- 1234 )
    dup                  \ Make space for new TOS value
    [ R24 $34 ldi, ]
    [ R25 $12 ldi, ]
;
```

### { -- <names..>

Marks the start of the definition of local variables. These local variables behave like pseudo-constants.

Local variables are an interesting alternative to the manipulation of stack data. They make the code more readable.

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

# ansi

Words defined in `ansi` vocabulary

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

### bel --

Equivalent to `7 emit`

### esc --

Equivalent to `27 emit`

# asm

Words defined in `asm` vocabulary

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

### >>1 n1 -- n2

Shift 1 bit to the right of n1.

### chere -- addr

Stacks the assembly pointer address.

### disasm addr --

Disassembles the XTENSA code.

```
code myL32R
    a1 32           ENTRY,
    a8 $fffe        L32R,
    a8          arPUSH,
                    RETW.N,
end-code

hex
' myL32R cell+ @ 5 disasm
```

### end-code --

Ends an assembly language definition.

```
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 --

Defines n words as constants.

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

### odd? n -- f

Returns flag true if n is odd.

**xtensa** --

Select the `xtensa` vocabulary.

# bluetooth

Words defined in `bluetooth` vocabulary.

```
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 --

Entry point to the `bluetooth` vocabulary.

### SerialBT.available bt -- n

Returns n #bytes available as input.

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

Starts a bluetooth instance.

addr: User assigned local name of the unit. `z" ESP32Master1"` master switch: 1 if in Master/controller mode 0 if in slave/receiver mode BT: Bluetooth object address assigned with 'New'. The unit will now be visible as 'user assigned name' Flag: 1 = success, 0 = failed

### SerialBT.connect addr bt --

Connect to a Receiving Unit - Used ONLY by a master\controlling unit

addr: name of the slave\receiving unit `z" ESP32-BT-Slave" BT SerialBT.connect ( add BT ? f )` Flag: 1 = success 0 = failed

### SerialBT.connectAddr addr bt -- f

An OPTIONAL connection mode.

Instead of using a device name the receiving device's actual MAC address can be used. This is a more secure connection mode than using a receivers name. Multiple devices can have the same name but not the same MAC address.

### SerialBT.connected n bt -- f

Checks for and waits for a connection for the time period specified in N (ms).

Flag: 1 true if connected else False 0

**SerialBT.delete** bt --

Free BT object.

**SerialBT.disconnect** bt -- f

Disconnects from client.

Returns True (1) or False (0)

**SerialBT.enableSSP** bt --

This function simply sets a flag for Bluetooth to use SSP (Simple Serial Paring)

**SerialBT.end** bt --

Terminates the bluetooth connection & object.

**SerialBT.flush** bt --

Use to make sure the transmit buffer has been emptied.

**SerialBT.hasClient** bt -- f

Checks to see if connected to a client.

Flag: True (1) if connected else False (0) not connected.

**SerialBT.isReady** master timeout -- f

Used ONLY by the Master\Controller.

Checks to see if Master is active Master = false 0 / timeout = 0 Flag = True \ False

**SerialBT.new** -- bt

Returns an address for a new bluetooth object.

This is the very first command to be executed when initializing bluetooth.

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

Returns n #bytes read into buffer.

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

Sets the pin used in pairing a device.

This is OPTIONAL and not necessary. Pin is in the form `z" 1234"`

**SerialBT.unpairDevice** addr -- f

Remote addr of receiving dving is required. see: esp_bt_dev_get_address

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

Returns n #bytes wrote to bluetooth from buffer.

0 returned if error occured. It is recommended outgoing strings of data be terminated with a crlf (0d0ah).

# editor

Words defined in `editor` vocabulary

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

### wipe --

Cleans the contents of the current block.

# ESP

Words defined in **ESP** vocabulary.

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

### deepSleep n --

Met the ESP32 board in deep sleep. The parameter n indicates the sleep delay. When waking up, ESP32Forth replaced.

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

### ESP-builtins -- addr

**ESP** vocabulary entry point.

### getChipCores -- n

Retrieves cores number of the processor.

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

### getChipModel -- addr-z

Retrieves the address of the 0-terminated string that identifies the chip model of the ESP32 board.

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

### getCpuFreqMHz -- n

Retrieves the frequency in Mhz of the processor.

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

### getFlashChipSize -- n

Retrieves the flash size.

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

### getSketchSize -- n

Stacks the size of the sketch used by ESP32Forth.

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

# httpd

Words defined in `httpd` vocabulary.

```
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

Request body.

### chunk -- addr

Data area defined by `create`

### chunk-filled -- n

Defined by `value`

### client -- addr

Defined by `sockaddr`

### client-emit c --

Send character c to network.

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

### client-len -- addr

Variable.

### goal -- addr

Defined by `variable`

### goal# -- addr

Defined by `variable`

### handleClient --

Get next request.

### hasHeader addr len -- fl

???

### header addr len -- addr len

Contents of header (or empty string).

### http-builtins -- addr

Entry point to the `http` vocabulary.

### httpd-port -- addr

Defined by `sockaddr`

### max-connections -- 1

Constant. Value 1

### method -- addr len

Request method, e.g. GET

### notfound-response --

Send error 404.

### ok-response mime$ --

Send 200.

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

### path -- addr len

Request path, e.g. /foo

### response mime$ result$ status --

Send a network response.

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

**send** addr len --

Request path, e.g. /foo

**sockfd** --- n

Store current socket.

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

# internalized

This vocabulary is an annex vocabulary of **internals**.

Words defined in **internalized** vocabulary.

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

# internals

Words defined in `internals` vocabulary.

```
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

Converts a real number into a character string. Used by `#fs`

### 'cold -- addr

Address of the word that will be run on startup. If contains 0 does not execute this word.

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

### 'notfound -- addr

Execution token of a handler to call on word not found

### 'sys -- addr

Base address for system variables.

### 'tib -- addr

Pointer to Terminal Input Buffer.

### (+to) xt --

`+to` part for local variables.

### (to) xt --

`to` part for local variables.

### block-data -- addr

Buffer area of 1024 bytes. Used by `editor`.

### block-dirty -- n

Serves as a flag to indicate if the current block has been modified.

### BRANCH --

Branch to the address following BRANCH. BRANCH is compiled by AFT, ELSE, REPEAT and AGAIN.

### common-default-use --

Opens the file *blocks.fb* by default

### default-use --

Runs `common-default-use` by default.

### digit n -- c

Convert a digit to ascii character.

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

### DOFLIT --

Puts a float from the next cell onto float stack.

### DOLIT -- n

Push the next token onto the data stack as an integer literal. It allows numbers to be compiled as in-line literals, supplying data to the data stack at run time.

### DONEXT --

Terminate a FOR-NEXT loop. The loop count was pushed on return stack, and is decremented by DONEXT. If the count is not negative, jump to the address following DONEXT; otherwise, pop the count off return stack and exit the loop. DONEXT is compiled by NEXT.

### esp32-bye --

Reboot ESP32Forth.

### esp32-stats --

Displays CPU-specific parameters of the ESP32 board and some parameters of 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 --

Expand the current file by n blocks.

### immediate? cfa -- fl

Test if a word is immediate.

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

### included-files -- n

Points to an included file.

### input-buffer -- addr

Memory zone defined by CREATE. Leave on stack address of input buffer. Size 200.

### input-limit -- 200

Constant value 200. Determines the size of the input buffer of the FORTH interpreter.

### last-vocabulary -- addr

Variable pointing to the last defined vocabulary.

### line-pos -- 0

value incremented with each word display by `words`.

### line-width -- 70

Sets the number of characters per line for running `words`

### locals-capacity -- 1024

Constant. Capacity of space dedicated to local variables.

### long-size -- 4

Pseudo-constant. Stack 4.

### MALLOC_CAP_32BIT -- 2

Constant. Value 2.

### MALLOC_CAP_8BIT -- 4

Constant. Value 4.

### MALLOC_CAP_DMA -- 8

Constant. Value 8.

### MALLOC_CAP_EXEC 1

Constant. Value 1

### remember-filename -- addr len

Deferred word specifying the platform specific default snapshot filename.

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

Evaluates the content of a character string and tries to transform the content into a number. Leave the value n and -1 if the evaluation is successful

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

### save-name a n --

Save a snapshot if the current vocabulary to a file.

### see-all --

Displays all the words in the dictionary. If the word is defined by: displays the decompilation of this word.

```
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 --

Displays the name of a FORTH word from its executable code.

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

### serial-key -- c

Get a pending character from the UART0 buffer.

### serial-key? -- c

Execute `Serial.available`. Tests if a character is available from serial port UART0.

### serial-type addr len --

Execute `Serial.write`.

### sourcefilename -- a n

Stacks the address and size of the filename pointed to by `sourcefilename&` and `sourcefilename#`.

### sourcefilename! a n --

Stores the address a and size n of the string pointing to a filename in `sourcefilename#` and `sourcefilename&`.

### sourcefilename# -- a

Store the address of the string pointing to a filename.

### sourcefilename& -- n

Store the size of the string pointing to a filename.

### voc. voc --

Used by `vocs.`

### voclist --

Displays the list of all available vocabularies.

```
\ on version v7.0.6.16
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 --

Used by `order`

### [SKIP] --

Defered word. Execute `[SKIP]'`

### [SKIP]' --

Loop that tests the words between `[IF][ELSE] {THEN]`.

# ledc

Words defined in `ledc` vocabulary.

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

### ledc-builtins -- addr

`ledc` vocabulary entry point

### ledcAttachPin pin channel --

Receives as input the GPIO and the channel.

The channels are numbered from 0 to 15. To produce a PWM signal on a pin, this pin must be associated with one of the 16 channels.

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

### ledcDetachPin pin --

Detaches the GPIO from the channel.

### ledcRead channel -- n

Gets the value of the PWM signal of the channel

### ledcReadFreq channel -- freq

Get frequency (x 1,000,000)

Returns the current frequency of the specified channel (this method returns 0 if the current duty cycle is 0).

### ledcSetup channel freq resolution -- freq

Init a PWM channel.

Set the frequency and count number (duty cycle resolution) corresponding to the LEDC channel. Returns the final frequency.

### ledcWrite channel duty --

Control PWM

### ledcWriteNote channel note octave -- freq

Play a note.

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

Write tone frequency (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

Words defined in `oled` vocabulary.

```
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

Constant, value 0

### HEIGHT -- 64

Constant, value 64

Indicates the height, in pixels, of an SSD1306 OLED display.

### oled-builtins -- addr

Entry point to the `oled` vocabulary.

### OledAddr -- addr

Variable.

### OledBegin 2 I2Caddr -- fl

Starts management of an SSD1306 OLED display.

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

### OledCirc x y radius color --

Draw a circle centered at x y, with radius radius and color color (0|1)

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

### OledCircF x y radius color --

Draw a full circle centered at x y, with radius radius and color color (0|1)

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

```
  ;
```

### OledCLS --

Clear OLED display.

```
oledCLS OledDisplay
```

### OledDelete --

Interrupts management of the SSD1306 OLED display.

### OledDisplay --

transmits the commands awaiting display to the OLED display.

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

### OledDrawL x0 y0 x1 y1 color --

### OledFastHLine x y length color --

Draws a horizontal line from x y of dimension length and color color.

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

### OledFastVLine x y length color --

Draws a vertical line from x y of dimension length and color color.

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

### OledHOME --

Set cursor at line 0, col 0 on OLED display

### OledInit --

Initializes communication with the SSD1306 OLED display.

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

### OledInvert --

Invert OLED display

### OledNew width height OledReset --

Instantiates a new SSD1306 OLED display with the parameters `WIDTH HEIGHT` and `OledReset`.

### OledNum n --

Displays the number n as a string on the OLED screen.

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

### olednumln n --

Displays an integer on the OLED display and moves to the next line.

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

### OledPixel x y color -- fl

Activates a pixel at position x y. The color parameter determines the color of the pixel.

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

### OledPrint z-string --

Displays z-string text on the OLED screen.

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

### OledPrintln z-string --

Transmits a z-string to the OLED display. The transmission ends with a return to the next line.

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

### OledRect x y width height color --

Draws an empty rectangle from position x y of size width height and color color.

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

### OledRectF x y width height color --

Draws an empty rectangle from position x y of size width height and color color.

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

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

Draw a rectangle with rounded corners, from the x y position, of dimension width heigh, in the color color, with a radius radius.

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

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

Draw a filled rectangle with rounded corners, from the x y position, of dimension width heigh, in the color color, with a radius radius.

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

### OledReset -- -1

Constant, value -1

### OledSetCursor x y --

Set cursor position.

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

### OledTextc color --

Sets the color of the text to display.

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

### OledTextsize n --

Sets the size of text to display on the OLED screen. The value of n must be in the interval [1..3]

For normal sized text, n=1. If you exceed the value 4, the text will be truncated on a 4-line display.

```
: 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

Constant, value 1

### SSD1306_SWITCHCAPVCC -- 2

Constant, value 2

### WHITE -- 1

Constant, value 1

Allows you to select the color of the pixels to display.

### WIDTH -- 128

Constant, value 128

Indicates the width, in pixels, of an SSD1306 OLED display.

# registers

Words defined in `registers` vocabulary.

```
m@ m!
```

### m! val shift mask addr --

Modifies the content of a register pointed to by addr, applies a logical mask with mask and shifts val by n bits according to 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

Reads the contents of a register pointed to by addr, applies a logical mask with mask and shifts by n bits according to 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

Words defined in `riscv` vocabulary.

Mots définis dans le vocabulaire

```
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 --

Load a 32-bit value from memory into register rd. It computes an effective address by adding the zero-extended offset, scaled by 4, to the stack pointer, x2.

### x1 -- 1

Push 1 on stack.

### x10 -- 10

Push 10 on stack.

### x11 -- 11

Push 11 on stack.

### x12 -- 12

Push 12 on stack.

### x13 -- 13

Push 13 on stack.

### x14 -- 14

Push 14 on stack.

### x15 -- 15

Push 15 on stack.

**x16** -- 16

Push 16 on stack.

**x17** -- 17

Push 17 on stack.

**x18** -- 18

Push 18 on stack.

**x19** -- 19

Push 19 on stack.

**x2** -- 2

Push 2 on stack.

**x20** -- 20

Push 20 on stack.

**x21** -- 21

Push 21 on stack.

**x22** -- 22

Push 22 on stack.

**x23** -- 23

Push 23 on stack.

**x24** -- 24

Push 24 on stack.

**x25** -- 25

Push 25 on stack.

**x26** -- 26

Push 26 on stack.

**x27** -- 27

Push 27 on stack.

**x28** -- 28

Push 28 on stack.

**x29** -- 29

Push 29 on stack.

**x3** -- 3

Push 3 on stack.

**x30** -- 30

Push 30 on stack.

**x31** -- 31

Push 31 on stack.

**x4** -- 4

Push 4 on stack.

**x5** -- 5

Push 5 on stack.

**x6** -- 6

Push 6 on stack.

**x7** -- 7

Push 7 on stack.

**x8** -- 8

Push 8 on stack.

**x9** -- 9

Push 9 on stack.

**zero** -- 0

Push 0 on stack.

# rmt

Words defined in `rmt` vocabulary.

```
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_uinstall 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

entry point of the `rmt` vocabulary.

### rmt_driver_uinstall channel -- err

Uninstall RMT driver.

### rmt_register_tx_end_callback --

NOT SUPPORTED

### rmt_set_clk_div channel div8 -- err

Set RMT clock divider, channel clock is divided from source clock.

### rmt_set_gpio channel mode gpio_num invert_signal -- err

Configure the GPIO used by RMT channel.

Parameters: - channel: RMT channel, in interval [0..7] - mode: RMT mode, either RMT_MODE_TX or RMT_MODE_RX - gpio_num: GPIO number, which is connected with certain RMT signal - invert_signal: Invert RMT signal physically by GPIO matrix

### rmt_set_mem_pd channel fl -- err

Set RMT memory in low power mode.

### rmt_set_pin --

DEPRECATED use rmt_set_gpio instead

# serial

Words defined in `Serial` vocabulary.

```
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

Fetch n|0 chars available in UART reception buffer.

```
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 --

Start serial port 0.

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

### Serial.end --

Disables serial communication, allowing the RX and TX pins to be used for general input and output. To re-enable serial communication, use `Serial.begin`.

### Serial.flush --

Waits for the transmission of outgoing serial data to complete.

### Serial.readBytes a n -- n

Read serial bytes on UART0, return characters count gotten.

### Serial.write addr len --

Send string addr len to UART

### Serial2.available -- n

Fetch n|0 chars available in UART 2 reception buffer.

```
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 --

Start serial port 2.

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

### Serial2.end --

Disables serial communication, allowing the RX and TX pins to be used for general input and output. To re-enable serial communication, use `Serial2.begin`.

### Serial2.flush --

Waits for the transmission of outgoing serial data to complete.

### Serial2.readBytes a n -- n

Read serial bytes on UART2, return characters count gotten.

### Serial2.write addr len --

Send string addr len to 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
```

# telnetd

Words defined in **telnetd** vocabulary.

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

### server port --

Start the telnet server on the specified port.

```
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 --

Emits a character on the active TELNET port.

### telnet-key -- n

Retrieves a character from the active TELNET port.

### telnet-type addr len --

Transmits a character string on the active TELNET port.

# web-interface

Words defined in `web-interface` vocabulary.

```
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

Marks the string address of the web interface.

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

### index-html# -- addr

Marks the size of string address of the web interface.

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

### ip# n -- n'

Displays part of the IP address. Used by `ip.`

### ip. --

Displays an IP address from its 32-bit address.

```
WIFI_MODE_STA  Wifi.mode
z" toSSID"                          \ type SSID of your netword=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

Words defined in `WiFi` vocabulary.

```
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

Entry point to the `WiFi` vocabulary

### Wifi.begin ssid-z password-z

Initializes the WiFi library's network settings and provides the current status.

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

### WiFi.config ip dns gateway subnet --

`WiFi.config` allows you to configure a static IP address as well as change the DNS, gateway, and subnet addresses on the WiFi shield.

Calling `WiFi.config` before `WiFi.begin` forces to configure the WiFi with the network addresses specified in `WiFi.config`. Parameters: • **ip**: the IP address of the device • **dns**: the address for a DNS server. • **gateway**: the IP address of the network gateway • **subnet**: the subnet mask of the network

### Wifi.disconnect --

Disconnects the WiFi shield from the current network.

### WiFi.getTxPower -- powerx4

Get power x4.

### WiFi.localIP -- ip

Get local IP.

### WiFi.macAddress -- a

Gets the MAC Address of your ESP32 WiFi port.

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

### WiFi.mode mode --

Set WiFi mode: WIFI_MODE_NULL WIFI_MODE_STA WIFI_MODE_AP WIFI_MODE_APSTA

### WiFi.setTxPower powerx4 --

Set power x4.

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

Set SSID and password for Acces Point mode.

### WiFi.status -- n

Return the connection status.

• **255** WL_NO_SHIELD
assigned when no WiFi shield is present • **0** WL_IDLE_STATUS
it is a temporary status assigned when `WiFi.begin` is called and remains active until the number of attempts expires (resulting in WL_CONNECT_FAILED) or a connection is established (resulting in WL_CONNECTED) • **1** WL_NO_SSID_AVAIL
assigned when no SSID are available • **2** WL_SCAN_COMPLETED
assigned when the scan networks is completed • **3** WL_CONNECTED
assigned when connected to a WiFi network • **4** WL_CONNECT_FAILED
assigned when the connection fails for all the attempts • **5** WL_CONNECTION_LOST
assigned when the connection is lost • **6** WL_DISCONNECTED
assigned when disconnected from a network

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

### WIFI_MODE_AP -- 2

Constant. Content 2

AP mode: in this mode, init the internal AP data, while the AP's interface is ready for RX/TX Wi-Fi data. Then, the Wi-Fi driver starts broad- casting beacons, and the AP is ready to get connected to other stations.

### WIFI_MODE_APSTA -- 3

Constant. Content 3

Station-AP coexistence mode: in this mode, will simultaneously init both the station and the AP.This is done in station mode and AP mode. Please note that the channel of the external AP, which the ESP Station is connected to, has higher priority over the ESP AP channel.

### WIFI_MODE_NULL -- 0

Constant. Content 0

In this mode, the internal data struct is not allocated to the station and the AP, while both the station and AP interfaces are not initialized for RX/TX Wi-Fi data. Generally, this mode is used for Sniffer, or when you only want to stop both the STA and the AP to unload the whole Wi-Fi driver.

### WIFI_MODE_STA -- 1

Constant. Content 1

Station mode: in this mode, will init the internal station data, while the station’s interface is ready for the RX and TX Wi-Fi data.

# wire

Words defined in `wire` vocabulary.

Mots définis dans le vocabulaire

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

### Wire-builtins -- addr

Entry point to the `Wire` vocabulary.

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

Returns the number of bytes available for retrieval with Wire.read word. This should be called on a master device after the use of the Wire.requestFrom word.

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

Initiate the Wire library and join the I2C bus as a master using the specified pins for sda and scl. The slave option is not available. This should normally be called only once.

sdapin# and sclpin# specifying the pin numbers used for sda and scl. error# 1 for success and 0 for failed. The 1 returned indicates that I2C bus was started properly.

```
\ 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.beginTransmission device-address --

Begin a transmission to the I2C slave device with the given address. Subsequently, queue bytes for transmission with the `Wire.write` function and transmit them by calling `Wire.endTransmission`.

device-address specifying the address of the slave device to transmit to.

```
Wire

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

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

### Wire.busy -- busy-indicator

Reads the state of the I2C bus. 1 for busy and 0 for free.

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

Ends a transmission to a slave device that was begun by beginTransmission and transmits the bytes that were queued by write.

sendstop-option: parameter changing its behavior for compatibility with certain I2C devices. If true or 1, endTransmission sends a stop message after transmission, releasing the I2C bus. If false or 0, endTransmission sends a restart message after transmission. The bus will not be released, which prevents another master device from transmitting between messages. This allows one master device to send multiple transmissions while in control. The default value is true. error: which indicates the status of the transmission: 0: success 1: data too long to fit in transmit buffer 2: received NACK on transmit of address 3: received NACK on transmit of data 4: other error

```
Wire

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

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

### Wire.flush --

Releases the I2C bus.

### Wire.getClock -- clockfrequency

Gets the clock frequency for I2C communication.

The following entries do the following: - activates the wire vocabulary with a 1 to indicate success, - starts the I2C interface using pin 21 and 22 on ESP32 DEVKIT V1 with 21 used as sda and 22 as scl, - shows the default clock speed of 100000 Khz after Wire.begin, - sets the clock speed at 400000 Khz, - confirms the clock speed at 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

Gets the address of null terminated text corresponding to the error specified.

### Wire.getTimeout -- timeout

Gets the timeout in ms for I2C communication.

### Wire.lastError -- lasterror

Gets the last error for I2C communication.

### Wire.peek -- read-data-byte

Reads a byte from a previously addressed slave device by using the requestFrom word. This is the same as a Wire.read except that the received-data-buffer-pointer is not incremented.

### Wire.read -- read-data-byte

Reads a byte from a previously addressed slave device by using the Wire.requestFrom word.

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

Used by the master to request a specified #of-bytes in a data buffer at address-of-data-buffer to the slave device with address-of-device and then end the transmission or not as specified by the sendstop-option.

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

Used by the master to request bytes from a slave device. The bytes may then be retrieved with the wire.available and wire.read words. The bytes are really received from the addressed slave device and stored in a data buffer then the connection is ended or not if required. The word wire.available indicates the numbers of bytes present in that data buffer and wire.read allows reading the bytes from the data buffer. Wire.requestFrom accepts a sendstop-option parameter changing its behavior for compatibility with certain I2C devices.

### Wire.setClock clockfrequency --

Modifies the clock frequency for I2C communication. I2C slave devices have no minimum working clock frequency, however 100KHz is usually the baseline.

clockFrequency: the value (in Hertz) of the communication clock. Accepted values are 100000 (standard mode) and 400000 (fast mode). Some processors also support 10000 (low speed mode), 1000000 (fast mode plus) and 3400000 (high speed mode). Please refer to the specific processor documentation to make sure the desired mode is supported.

### Wire.setTimeout timeout --

Modifies the timeout in ms for I2C communication. The timeout is expressed in ms. The default value is 50 ms.

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

Queues of-bytes to address-of-data-buffer for transmission from a master to previously selected slave device (in-between `Wire.beginTransmission` and `Wire.endTransmission` words).

```
Wire

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

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

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

Used by the master to send a specified #of-bytes from a data buffer at address-of-data-buffer to the slave device with address-of-device and then end the transmission. The sendstop-option is set true.

# xtensa

Words defined in `xtensa` vocabulary.

```
WUR, WSR, WITLB, WER, WDTLB, WAITI, SSXU, SSX, SSR, SSL, SSIU, SSI, SSAI,
SSA8L, SSA8B, SRLI, SRL, SRC, SRAI, SRA, SLLI, SLL, SICW, SICT, SEXT, SDCT,
RUR, RSR, RSIL, RFI, ROTW, RITLB1, RITLB0, RER, RDTLB1, RDTLB0, PITLB,
PDTLB, NSAU, NSA, MULA.DD.HH, MULA.DD.LH, MULA.DD.HL, MULA.DD.LL, MULS.DD
MULA.DA.HH, MULA.DA.LH, MULA.DA.HL, MULA.DA.LL, MULS.DA MULA.AD.HH, MULA.AD.LH,
MULA.AD.HL, MULA.AD.LL, MULS.AD MULA.AA.HH, MULA.AA.LH, MULA.AA.HL, MULA.AA.LL,
MULS.AA MULA.DD.HH.LDINC, MULA.DD.LH.LDINC, MULA.DD.HL.LDINC, MULA.DD.LL.LDINC,
MULA.DD.LDINC MULA.DD.HH.LDDEC, MULA.DD.LH.LDDEC, MULA.DD.HL.LDDEC,
MULA.DD.LL.LDDEC,
MULA.DD.LDDEC MULA.DD.HH, MULA.DD.LH, MULA.DD.HL, MULA.DD.LL, MULA.DD
MULA.DA.HH.LDINC,
MULA.DA.LH.LDINC, MULA.DA.HL.LDINC, MULA.DA.LL.LDINC, MULA.DA.LDINC
MULA.DA.HH.LDDEC,
MULA.DA.LH.LDDEC, MULA.DA.HL.LDDEC, MULA.DA.LL.LDDEC, MULA.DA.LDDEC MULA.DA.HH,
MULA.DA.LH, MULA.DA.HL, MULA.DA.LL, MULA.DA MULA.AD.HH, MULA.AD.LH, MULA.AD.HL,
MULA.AD.LL, MULA.AD MULA.AA.HH, MULA.AA.LH, MULA.AA.HL, MULA.AA.LL, MULA.AA
MUL16U, MUL16S, MUL.DD.HH, MUL.DD.LH, MUL.DD.HL, MUL.DD.LL, MUL.DD MUL.DA.HH,
MUL.DA.LH, MUL.DA.HL, MUL.DA.LL, MUL.DA MUL.AD.HH, MUL.AD.LH, MUL.AD.HL,
MUL.AD.LL, MUL.AD MUL.AA.HH, MUL.AA.LH, MUL.AA.HL, MUL.AA.LL, MUL.AA MOVT,
MOVSP, MOVT.S, MOVF.S, MOVGEZ.S, MOVLTZ.S, MOVNEZ.S, MOVEQZ.S, ULE.S, OLE.S,
ULT.S, OLT.S, UEQ.S, OEQ.S, UN.S, CMPSOP NEG.S, WFR, RFR, ABS.S, MOV.S,
ALU2.S UTRUNC.S, UFLOAT.S, FLOAT.S, CEIL.S, FLOOR.S, TRUNC.S, ROUND.S,
MSUB.S, MADD.S, MUL.S, SUB.S, ADD.S, ALU.S MOVF, MOVGEZ, MOVLTZ, MOVNEZ,
MOVEQZ, MAXU, MINU, MAX, MIN, CONDOP MOV, LSXU, LSX, L32E, LICW, LICT,
LDCT, JX, IITLB, IDTLB, LSIU, LSI, LDINC, LDDEC, L32R, EXTUI, S32E, S32RI,
S32C1I, ADDMI, ADDI, L32AI, L16SI, S32I, S16I, S8I, L32I, L16UI, L8UI,
LDSTORE MOVI, IIU, IHU, IPFL, DIWBI, DIWB, DIU, DHU, DPFL, CACHING2 III,
IHI, IPF, DII, DHI, DHWBI, DHWB, DPFWO, DPFRO, DPFW, DPFR, CACHING1 CLAMPS,
BREAK, CALLX12, CALLX8, CALLX4, CALLX0, CALLXOP CALL12, CALL8, CALL4, CALL0,
CALLOP LOOPGTZ, LOOPNEZ, LOOP, BT, BF, BRANCH2b J, BGEUI, BGEI, BGEZ, BLTUI,
BLTI, BLTZ, BNEI, BNEZ, ENTRY, BEQI, BEQZ, BRANCH2e BRANCH2a BRANCH2 BBSI,
BBS, BNALL, BGEU, BGE, BNE, BANY, BBCI, BBC, BALL, BLTU, BLT, BEQ, BNONE,
BRANCH1 REMS, REMU, QUOS, QUOU, MULSH, MULUH, MULL, XORB, ORBC, ORB, ANDBC,
ANDB, ALU2 ALL8, ANY8, ALL4, ANY4, ANYALL SUBX8, SUBX4, SUBX2, SUB, ADDX8,
ADDX4, ADDX2, ADD, XOR, OR, AND, ALU XSR, ABS, NEG, RFDO, RFDD, SIMCALL,
SYSCALL, RFWU, RFWO, RFDE, RFUE, RFME, RFE, NOP, EXTW, MEMW, EXCW, DSYNC,
ESYNC, RSYNC, ISYNC, RETW, RET, ILL, ILL.N, NOP.N, RETW.N, RET.N, BREAK.N,
MOV.N, MOVI.N, BNEZ.N, BEQZ.N, ADDI.N, ADD.N, S32I.N, L32I.N, tttt t ssss
s rrrr r bbbb b y w iiii i xxxx x sa sa. >sa entry12 entry12' entry12.
>entry12 coffset18 cofs cofs. >cofs offset18 offset12 offset8 ofs18 ofs12
ofs8 ofs18. ofs12. ofs8. >ofs sr imm16 imm8 imm4 im numeric register reg.
nop a15 a14 a13 a12 a11 a10 a9 a8 a7 a6 a5 a4 a3 a2 a1 a0
```

### a0 -- 0

Push 0 on stack.

### a1 -- 1

Push 1 on stack.

**a10** -- 10

Push 10 on stack.

**a11** -- 11

Push 11 on stack.

**a12** -- 12

Push 12 on stack.

**a13** -- 13

Push 13 on stack.

**a14** -- 14

Push 14 on stack.

**a15** -- 15

Push 15 on stack.

**a2** -- 2

Push 2 on stack.

**a3** -- 3

Push 3 on stack.

**a4** -- 4

Push 4 on stack.

**a5** -- 5

Push 5 on stack.

**a6** -- 6

Push 6 on stack.

**a7** -- 7

Push 7 on stack.

**a8** -- 8

Push 8 on stack.

### a9 -- 0

Push 9 on stack.

### ABS, at ar --

Absolute value. Format RRR

**ABS,** calculates the absolute value of the contents of address register at and writes it to address register ar. Arithmetic overflow is not detected.

```
\ for macros, see:
\
https://github.com/MPETREMANN11/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 --

Absolute Value Single. Instruction Word (RRR).

**ABS.S,** computes the single-precision absolute value of the contents of floating-point register fs and writes the result to floating-point register fr.

### ADD, at as ar --

Addition. Instruction Word (RRR).

**ADD,** calculates the two's complement 32-bit sum of address registers as and at. The low 32 bits of the sum are written to address register ar. Arithmetic overflow is not detected.

```
\ for marcos, see:
\
https://github.com/MPETREMANN11/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 --

Narrow Add. Instruction Word (RRRN). AR[r] ← AR[s] + AR[t]

This performs the same operation as the **ADD,** instruction in a 16-bit encoding.

```
\ for macros, see:
\
https://github.com/MPETREMANN11/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 --

Add Single. Instruction Word (RRR).

**ADD.S,** computes the IEEE754 single-precision sum of the contents of floating-point registers fs and ft, and writes the result to floating-point register fr.

## ADDI, at as -128..127

Add Immediate. Instruction Word (RRI8). AR[t] ← AR[s] + (imm8724||imm8)

**ADDI,** calculates the two's complement 32-bit sum of address register as and a constant encoded in the imm8 field. The low 32 bits of the sum are written to address register at. Arithmetic overflow is not detected.

```
\ for macros, see:
\
https://github.com/MPETREMANN11/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 --

Narrow Add Immediate. Instruction Word (RRRN).

**ADDI.N,** is similar to **ADDI,** but has a 16-bit encoding and supports a smaller range of immediate operand values encoded in the instruction word. **ADDI.N,** calculates the two's complement 32-bit sum of address register as and an operand encoded in the t field. The low 32 bits of the sum are written to address register ar. Arithmetic overflow is not detected. The operand encoded in the instruction can be -1 or one to 15. If t is zero, then a value of -1 is used, otherwise the value is the zero-extension of t.

## ADDMI, at as -32768..32512

Add Immediate with Shift by 8. Instruction Word (RRI8).

**ADDMI,** extends the range of constant addition. It is often used in conjunction with load and store instructions to extend the range of the base, plus offset the calculation. ADDMI calculates the two's complement 32-bit sum of address register as and an operand encoded in the imm8 field. The low 32 bits of the sum are written to address register at. Arithmetic overflow is not detected. The operand encoded in the instruction can have values that are multiples of 256 ranging from -32768 to 32512. It is decoded by sign-extending imm8 and shifting the result left by eight bits.

## ADDX2, ar as at --

Add with Shift by 1. Instruction Word (RRR).

**ADDX2,** calculates the two's complement 32-bit sum of address register as shifted left by one bit and address register at. The low 32 bits of the sum are written to address register ar. Arithmetic overflow is not detected. **ADDX2,** is frequently used for address calculation and as part of sequences to multiply by small constants.

## ADDX4, ar as at --

Add with Shift by 1. Instruction Word (RRR).

**ADDX4,** calculates the two's complement 32-bit sum of address register as shifted left by two bit and address register at. The low 32 bits of the sum are written to address register ar. Arithmetic overflow is not detected. **ADDX4,** is frequently used for address calculation and as part of sequences to multiply by small constants.

## ADDX8, ar as at --

Add with Shift by 1. Instruction Word (RRR).

**ADDX8,** calculates the two's complement 32-bit sum of address register as shifted left by three bit and address register at. The low 32 bits of the sum are written to address register ar. Arithmetic overflow is not detected. **ADDX8,** is frequently used for address calculation and as part of sequences to multiply by small constants.

### ALL4, bt bs --

All 4 Booleans True. Instruction Word (RRR).

**ALL4,** sets Boolean register bt to the logical and of the four Boolean registers bs+0, bs+1, bs+2, and bs+3. bs must be a multiple of four (b0, b4, b8, or b12); otherwise the operation of this instruction is not defined. **ALL4,** reduces four test results such that the result is true if all four tests are true. When the sense of the bs Booleans is inverted (0 → true, 1 → false), use **ANY4,** and an inverted test of the result.

### ALL8, bt bs --

All 8 Booleans True. Instruction Word (RRR).

**ALL8,** sets Boolean register bt to the logical and of the eight Boolean registers bs+0, bs+1, … bs+6, and bs+7. bs must be a multiple of eight (b0 or b8); otherwise the operation of this instruction is not defined. ALL8 reduces eight test results such that the result is true if all eight tests are true. When the sense of the bs Booleans is inverted (0 → true, 1 → false), use **ANY8,** and an inverted test of the result.

### AND, at as ar --

Bitwise Logical And. Instruction Word (RRR). AR[r] ← AR[s] and AR[t]

**AND,** calculates the bitwise logical and of address registers as and at. The result is written to address register 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 --

Boolean And. Instruction Word (RRR).

**ANDB,** performs the logical and of Boolean registers bs and bt and writes the result to Boolean register br.

### ANDBC, br bs bt

Boolean And with Complement. Instruction Word (RRR).

**ANDBC,** performs the logical and of Boolean register bs with the logical complement of Boolean register bt, and writes the result to Boolean register br.

### ANY4, bt bs --

Any 4 Booleans True. Instruction Word (RRR).

**ANY4,** sets Boolean register bt to the logical or of the four Boolean registers bs+0, bs+1, bs+2, and bs+3. bs must be a multiple of four (b0, b4, b8, or b12); otherwise the operation of this instruction is not defined. **ANY4,** reduces four test results such that the result is true if any of the four tests are true. When the sense of the bs Booleans is inverted (0 → true, 1 → false), use **ALL4,** and an inverted test of the result.

### ANY8, bt bs --

Any 8 Booleans True. Instruction Word (RRR).

**ANY8,** sets Boolean register bt to the logical or of the eight Boolean registers bs+0, bs+1, … bs+6, and bs+7. bs must be a multiple of eight (b0 or b8); otherwise the operation of this instruction is not defined. **ANY8,** reduces eight test results such that the result is true if any of the eight tests are true. When the sense of the bs Booleans is inverted (0 → true, 1 → false), use **ALL8,** and an inverted test of the result.

### BALL, as at label --

Branch if All Bits Set. Instruction Word (RRI8).

**BALL,** branches if all the bits specified by the mask in address register at are set in address register as. The test is performed by taking the bitwise logical and of at and the complement of as, and testing if the result is zero. The target instruction address of the branch is given by the address of the **BALL,** instruction, plus the sign-extended 8-bit imm8 field of the instruction plus four. If any of the masked bits are clear, execution continues with the next sequential instruction. The inverse of **BALL,** is **NBALL,**.

### BEQ, as at --

Branch if Equal.

**BEQ,** branches if address registers as and at are equal. It is advisable to use this connection through the macro instruction **<>,**.

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

code my<> ( n1 n2 -- fl )    \ fl=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 --

Branch if Equal to Zero. Instruction Word BRI12.

**BEQZ,** branches if address register as is equal to zero. **BEQZ,** provides 12 bits of target range instead of the eight bits available in most conditional branches. The target instruction address of the branch is given by the address of the **BEQZ,** instruction, plus the sign-extended 12-bit imm12 field of the instruction plus four. If register **as** is not equal to zero, execution continues with the next sequential instruction. The inverse of **BEQZ,** is **BNEZ,**.

### CALL0, addr --

Makes a call to a subroutine pointed to by 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 --

Subroutine Entry. Instruction Word (BRI12).

`ENTRY,` is intended to be the first instruction of all subroutines called with `CALL4, CALL8, CALL12, CALLX4, CALLX8,` or `CALLX12,`. This instruction is not intended to be used by a routine called by `CALL0` or `CALLX0,`. `ENTRY,` serves two purposes:

1. First, it increments the register window pointer (WindowBase) by the amount requested by the caller (as recorded in the PS.CALLINC field).
2. Second, it copies the stack pointer from caller to callee and allocates the callee's stack frame. The as operand specifies the stack pointer register; it must specify one of a0..a3 or the operation of `ENTRY,` is undefined. It is read before the window is moved, the stack frame size is subtracted, and then the as register in the moved window is written.

The stack frame size is specified as the 12-bit unsigned imm12 field in units of eight bytes. The size is zero-extended, shifted left by 3, and subtracted from the caller's stack pointer to get the callee's stack pointer. Therefore, stack frames up to 32760 bytes can be specified. The initial stack frame size must be a constant, but subsequently the MOVSP instruction can be used to allocate dynamically-sized objects on the stack, or to further extend a constant stack frame larger than 32760 bytes.

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

### EXTW, --

External Wait. Instruction Word (RRR).

### IDTLB, as --

Invalidate Data TLB Entry. Instruction Word (RRR).

### J, label --

Unconditional Jump. Instruction Word (CALL).

`J,` performs an unconditional branch to the target address. It uses a signed, 18-bit PC-relative offset to specify the target address. The target address is given by the address of the `J,` instruction plus the sign-extended 18-bit offset field of the instruction plus four, giving a range of -131068 to +131075 bytes.

### JX, as --

Unconditional Jump Register. Instruction Word (CALLX).

`JX,` performs an unconditional jump to the address in register as.

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

Narrow Load 32-bit. Instruction Word (RRRN).

`L32I.N,` is a 32-bit load from memory. It forms a virtual address by adding the contents of address register as and a 4-bit zero-extended constant value encoded in the instruction word shifted left by two. Therefore, the offset can specify multiples of four from zero to 60. Thirty-two bits (four bytes) are read from the physical address. This data is then written to address register 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 --

Load 32-bit PC-Relative.

```
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, as label --

Manage loop.

Instruction not usable directly. See the `For,` and `Next,` macros.

### MAX, ar as at --

Maximum Value. Instruction Word (RRR).

`MAX,` computes the maximum of the twos complement contents of address registers as and at and writes the result to address register ar.

### MAXU, ar as at --

Maximum Value. Instruction Word (RRR).

`MAXU,` computes the maximum of the unsigned contents of address registers as and at and writes the result to address register ar.

### MIN, ar as at --

Minimum Value. Instruction Word (RRR).

`MIN,` computes the minimum of the twos complement contents of address registers as and at and writes the result to address register ar.

### MINU, ar as at --

Minimum Value. Instruction Word (RRR).

`MINU,` computes the minimum of the unsigned contents of address registers as and at, and writes the result to address register ar.

### MOV, ar as --

Move. Instruction Word (RRR).

`MOV,` is an assembler macro that uses the OR, instruction to move the contents of address register as to address register ar. The assembler input `ar as MOV,` expands into `ar as as OR,` ar and as should not specify the same register due to the `MOV.N,` restriction.

### MOVF, bt as ar --

Move if False.

`MOVF,` moves the contents of address register as to address register ar if Boolean register bt is false. Address register ar is left unchanged if Boolean register bt is true. The inverse of `MOVF,` is `MOVT,`.

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

Move Immediate.

`MOVI,` sets address register **at** to a constant in the range -2048..2047 encoded in the instruction word.

```
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 --

Move if True.

`MOVT,` moves the contents of address register as to address register ar if Boolean register bt is true. Address register ar is left unchanged if Boolean register bt is false. The inverse of `MOVT,` is `MOVF,`.

## MULL, ar as at --

Multiply Low. Instruction Word (RRR).

`MULL,` performs a 32-bit multiplication of the contents of address registers as and at, and writes the least significant 32 bits of the product to address register ar. Because the least significant product bits are unaffected by the multiplicand and multiplier sign, `MULL,` is useful for both signed and unsigned multiplication.

```
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 --

Negate. AR[r] ← 0 − AR[t]

`NEG,` calculates the two's complement negation of the contents of address register at and writes it to address register ar. Arithmetic overflow is not detected.

```
\ 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, --

No-Operation Instruction Word (RRR)

This instruction performs no operation. It is typically used for instruction alignment. `NOP,` is a 24-bit instruction. For a 16-bit version, see `NOP.N,`.

### NOP.N, --

No-Operation Instruction Word (RRRN)

This instruction performs no operation. It is typically used for instruction alignment. `NOP.N,` is a 16-bit instruction. For a 24-bit version, see `NOP,`.

### OR, ar as at --

Bitwise Logical Or. Instruction Word (RRR).

`OR,` calculates the bitwise logical or of address registers as and at. The result is written to address register ar.

### QUOS, at as ar --

Quotient Signed. Instruction Word (RRR).

`QUOS,` performs a 32-bit two's complement division of the contents of address register as by the contents of address register at and writes the quotient to address register ar. The ambiguity which exists when either address register as or address register at is negative is resolved by requiring the product of the quotient and address register at to be smaller in absolute value than the address register as. If the contents of address register at are zero, QUOS raises an Integer Divide by Zero exception instead of writing a result. Overflow (-2147483648 divided by -1) is not detected.

```
\ 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 --

Remainder Signed. Instruction Word (RRR).

**REMS,** performs a 32-bit two's complement division of the contents of address register as by the contents of address register at and writes the remainder to address register ar. The ambiguity which exists when either address register as or address register at is negative is resolved by requiring the remainder to have the same sign as address register as. If the contents of address register at are zero, REMS raises an Integer Divide by Zero exception instead of writing a result.

```
\ 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, --

Non-Windowed Return. Instruction Word (CALLX).

**RET,** returns from a routine called by **CALL0,** or **CALLX0,**. It is equivalent to the instruction **JX**. **RET,** exists as a separate instruction because some Xtensa ISA implementations may realize performance advantages from treating this operation as a special case.

### RET.N, --

**RET.N,** is the same as **RET,** in a 16-bit encoding. **RET** returns from a routine called by **CALL0,** or **CALLX0**.

### RETW.N, --

Narrow Windowed Return. Instruction Word (RRRN).

**RETW.N,** is the same as **RETW,** in a 16-bit encoding. **RETW.N** returns from a subroutine called by **CALL4, CALL8, CALL12, CALLX4, CALLX8,** or **CALLX12,** and that had **ENTRY,** as its first 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, ...

Round Single to Fixed.

ROUND.S converts the contents of floating-point register fs from single-precision to signed integer format, rounding toward the nearest. The single-precision value is first scaled by a power of two constant value encoded in the t field, with 0..15 representing 1.0, 2.0, 4.0, …, 32768.0. The scaling allows for a fixed point notation where the binary point is at the right end of the integer for t=0 and moves to the left as t increases until for t=15 there are 15 fractional bits represented in the fixed point number.

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

Read Special Register.

The contents of the Special Register designated by the 8-bit sr field of the instruction word are written to address register at.

```
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
```

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

Narrow Store 32-bit. Instruction Word (RRRN).

`S32I.N,` is a 32-bit store to memory. It forms a virtual address by adding the contents of address register as and an 4-bit zero-extended constant value encoded in the instruction word shifted left by two. Therefore, the offset can specify multiples of four from zero to 60. The data to be stored is taken from the contents of address register at and written to memory at the physical address.

```
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 --

Sign Extend. Instruction Word (RRR).

`SEXT,` takes the contents of address register as and replicates the bit specified by its immediate operand (in the range 7 to 22) to the high bits and writes the result to address register ar. The input can be thought of as an imm+1 bit value with the high bits irrelevant and this instruction produces the 32-bit sign-extension of this value.

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

Shift Left Logical Immediate. Instruction Word (RRR).

Shifts the contents of address register as left by a constant amount in the range 1..31 encoded in the 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 --

Shift Right Arithmetic

`SRA,` arithmetically shifts the contents of address register **at** right, inserting the sign of `at` on the left, by the number of bit positions specified by SAR (shift amount register) and writes the result to address register `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 --

Shift Right Arithmetic Immediate.

`SRAI,` arithmetically shifts the contents of address register **at** right, inserting the sign of **at** on the left, by a constant amount encoded in the instruction word in the range 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 --

Shift Right Logical Immediate. Instruction Word (RRR).

`SRLI,` shifts the contents of address register at right, inserting zeros on the left, by a constant amount encoded in the instruction word in the range 0..15. There is no `SRLI,` for shifts ≥ 16.

### SSA8B, as --

Set Shift Amount for BE Byte Shift. Instruction Word (RRR).

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

Write a special register.

The content of the at address register is written to the special register designated by the SR field defined in the range 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 --

Boolean Exclusive Or

`XOR,` calculates the bitwise logical exclusive or of address registers **as** and **at**. The result is written to address register **ar**.