Hitchiker’s Guide to FlashForth on PIC18 Microcontrollers

Interpreter
The outer interpreter looks for words and numbers delimited by whitespace. Everything is interpreted as a word or a number. Numbers are pushed onto the stack. Words are looked up and acted upon. Names of words are limited to 15 characters.

Data and the stack
The data stack is directly accessible and has 48 16-bit cells for holding numerical values. Functions get their arguments from the stack and leave their results there as well. There is also a return address stack (R) that can be used for temporary storage.

Notation
n, n1, n2, n3 Single-cell integers (16-bit).
un Signed integer (16-bit).
x, x1, x2, x3 Single-cell integer (16-bit).
c Character value (8-bit).
d Double-cell value (32-bit).
u Integer (16-bit).
ud Unsigned integer (16-bit).
udc Unsigned double-cell value (32-bit).
f Boolean flag: 0 is false, -1 is true.
addr, addr1, addr2 16-bit addresses.
a=addr cell-aligned address.
c=addr character or byte address.

Numbers and values
2 Leave integer two onto the stack. ( -- 2 )
255 Leave decimal 255 onto the stack. ( -- 255 )
%1 Leave integer three onto the stack. ( -- 3 )
$10 Leave integer sixteen onto the stack. ( -- 16 )
23 Leave double number on the stack. ( -- 23 0 )
decimal Set number format to decimal. ( -- )
hex Set number format to hexadecimal. ( -- )
bin Set number format to binary. ( -- )
s+d Sign extend single to double number. ( n -- d )

Displaying data

.dump Display memory from address, for u bytes. ( addr u -- )

Stack manipulation
dup Duplicate top item. ( x -- x x )
?dup Duplicate top item if nonzero. ( x -- (0 | x x) )
swap Swap top two items. ( x1 x2 -- x2 x1 )
over Copy copy second item to top. ( x1 x2 x3 -- x1 x2 x3 x1 )
drop Discard top item. ( x -- )
nip Remove x1 from the stack. ( x1 x2 x3 -- x2 x3 )
rot Rotate top three items. ( x1 x2 x3 -- x2 x3 x1 )
tuck Insert x2 below x1 in the stack. ( x1 x2 x3 -- x2 x1 x3 )
pick Duplicate the u-th item on top.
( xu ... x0 u -- xu ... x0 xu )
2dup Duplicate top item. ( d -- d d )
2swap Swap top two double-cell items. ( d1 d2 -- d2 d1 )
2over Copy second item to top. ( d1 d2 -- d1 d2 d1 )
2drop Discard top item. ( d -- )
>rdrop Send to return stack. ( n -- ( R: n n ) )
r> Take from return stack. ( n -- ( R: n n ) )
r> Copy top item of return stack. ( -- ( R: n n ) )
rdrop Discard top item from return stack. ( -- ( R: n n ) )
sp Leave data stack pointer. ( -- addr )
sp! Set data stack pointer to address. ( addr -- )

Operators

Arithmetic
Some of these words come from core.txt.
+ Add. ( n1 n2 -- n1+n2 )
* Subtract. ( n1 n2 -- n1-n2 )
* Multiply. ( n1 n2 -- n1*n2 )
/ Divide. ( n1 n2 -- n1/n2 )
mod Divide. ( n1 n2 -- n1%n2 )
mod Remainder. ( n1 n2 -- n1%n2 )
rem Remainder. ( n1 n2 -- n1%n2 )
*/ Scale. ( n1 n2 n3 -- n1*n2*n3 )
uses 32-bit intermediate result.
+/mod Scale with remainder. ( n1 n2 n3 -- n1*n2%n3 )
uses 32-bit intermediate result.
+= Unsign. Scale (1<<32) ( u1 u2/u3 ( u1 u2 u3 -- u1*u2/u3 )
uses 32-bit intermediate result.
+- Unsign. Scale (1<<64) ( u1 u2/u3 ( u1 u2 u3 -- u1*u2/u3 )
uses 32-bit intermediate result.
-= Unsign. Scale (1<<32) ( u1 u2/u3 ( u1 u2 u3 -- u1*u2/u3 )
uses 32-bit intermediate result.
-= Unsign. Scale (1<<64) ( u1 u2/u3 ( u1 u2 u3 -- u1*u2/u3 )
uses 32-bit intermediate result.
1 << n Left shift by u bits. ( x1 u -- x1 u )

Relational
= Leave true if x1 x2 are equal. ( x1 x2 -- f )
<> Leave true if x1 x2 are not equal. ( x1 x2 -- f )
< Leave true if n1 less than n2. ( n1 n2 -- f )
> Leave true if n1 greater than n2. ( n1 n2 -- f )
0= Leave true if n is zero. ( n -- f )

Bitwise
invert Inverts logical value. ( n -- ~n )
and AND file register byte with mask c. ( c addr -- )
mclr Clear bits in file register with mask c. ( c addr -- )
set AND file register byte with mask c. ( c addr -- )

Interaction with the operator
Interaction with the user is via the serial port, typically UART1. Settings are 38400 baud, 8N1, using Xon/Xoff handshaking.
x0 Send a character via the USB UART. ( c -- )
The P register

The P register can be used as a variable or as a pointer. It can be used in conjunction with for..next or at any other time.

| p  | Store address to P(point) register. (addr -- ) |
| q  | Fetch the P register to the stack. (addr -- ) |
| pr | Push contents of P to return stack and store new address to P. (addr -- ) (R: addr -- ) |
| rpp| Pop from return stack to P register. (R: addr -- ) |
| p+ | Increment P register by one. (addr -- ) |
| p2+| Add 2 to P register. (addr -- ) |
| p++| Add n to the p register. (n addr -- ) |
| p! | Store x to the location pointed to by the p register. (x addr -- ) |
| pc1| Store c to the location pointed to by the p register. (c addr -- ) |
| pc0| Fetch the char pointed to by the p register. (c addr -- ) |

In a definition |p| and |rpp| should always be used to allow proper nesting of words.

Predefined constants

cell Size of one cell in characters. (-- n)
true Boolean true value. (-- -1)
false Boolean false value. (-- 0)
b1 ASCII space. (-- c)
Fcy Leave the cpu instruction-cycle frequency in kHz. (-- u)
ti# Size of the terminal input buffer. (-- u)

Predefined variables

base Variable containing number base. (-- a addr)
irq Interrupt vector (SRAM variable). (-- a addr)
Always disable user interrupts and clear related interrupt enable bits before zeroing interrupt vector.
di false to irq ei

turnkey Vector for user start-up word. (-- a addr)
EEEPROM value mirrored in SRAM.
prompt Deferred execution vector for the info displayed by quit. (-- a addr)
*emit EMIT vector. Default is TX1. (-- a addr)
*key KEY vector. Default is RX1. (-- a addr)
*k? KEY? vector. Default is RX1. (-- a addr)
*source Current input source. (-- a addr)
s0 Variable for start of data stack. (-- a addr)
rcnt Number of saved return stack cells. (-- a addr)
tib Addr of the terminal input buffer. (-- a addr)
tiu Terminal input buffer pointer. (-- a addr)
>in Variable containing the offset, in characters, from the start of tib to the current parse area. (-- a addr)

Context

| ram | Set address context to SRAM. |
| eeprom | Set address context to EEPROM. |
| flash | Set address context to Flash. |
| f1 | Disable writes to Flash, EEPROM, default. |
| f1+ | Enable writes to Flash, EEPROM, default. |

Definitions

| con name | Define new constant, inline code. (a n -- ) |
| constant name | Define new constant, doctrate. (n -- ) |
| variable vname | Define variable in address context. (n v -- ) |
| value valname | Define value. (n valname -- ) |
| to valname | Assign new value to valname. (n valname -- ) |
| 2con name | Define double constant. (x x -- ) |
| 2vvariable name | Define double variable. (n -- ) |

Accessing

| varname | Leave address of variable on stack. (addr -- ) |
| valname | Leave value on stack. (n -- ) |
| ! | Store x to address. (x addr -- ) |
| @ | Fetch from address. (addr x -- ) |
| & | Store character to address. (c addr -- ) |
| @ | Fetch character from address. (addr c -- ) |
| @+ | Increment array address. (addr2 c -- addr2) |
| @- | Fetch from addr and decrement addr by 2. (addr1 -- addr2 x) |

Examples

| ram | Set SRAM context for variables and values. Be careful not to accidentally define variables in EEPROM or Flash memory. That memory wears quickly with multiple writes. |
| $ff81 | Define constant in Flash. |
| 3 | Define value in SRAM. |
| 6 | Store 6 in variable. |
| eeeprom | Define value in EEPROM. |
| 5 | Add n to cell address. (n addr -- ) |
| 0 | Fetch from addr and decrement addr by 2. (addr1 -- addr2 x) |

Converting between cells, chars

| cells | Convert cells to address units. (u -- u) |
| chars | Convert chars to address units. (u -- u) |
| char* | Add one to address. (addr1 -- addr1) |
| cell* | Add size of cell to address. (addr1 -- addr2) |
| aligned | Align address to a cell boundary. (addr addr -- ) |

Memory operations

| some_of_these_words | from core.txt | movc | Move x bytes from address-1 to address-2. (addr2 u -- ) |
| copy | Proceeds from low addr to high address. |
| fill | Fill u bytes with c starting at address. (addr u c -- ) |
| erase | Fill u bytes with 0 starting at address. (addr u -- ) |
| blanks | Fill u bytes with spaces starting at address. (addr u -- ) |

Other Hardware

| cwd | Clear the WatchDog counter. (addr -- ) |
| ei | Enable interrupts. (addr -- ) |
| di | Disable interrupts. (addr -- ) |
| ms | Pause for n milliseconds. (n -- ) |
| ticks | System ticks, 0-fif milliseconds. (n -- ) |

Constants, variables and memory

Memory map

All operations are restricted to 64kB address space that is divided into three spaces:

- $0000 - $00ff Flash
- $0e00 - $0fff EEPROM
- $f000 - $ffff SRAM

Using the default values in p18f-main.cfg for the UART version of FF, SRAM is further subdivided as:

- $f000 - $f03f 64-byte flash write buffer
- $f040 - $f05f 32-byte used internally by FF
- $f060 - $f06f 16-byte interrupt parameter stack
- $f070 - $f093 36-byte RX buffer (32) and TX buffer (4)
- $f094 - $f09d 10-byte mirror of turnkey, dp, latest
- $f09e - $f0ff 2-byte interrupt vector
- $f0a0 - $f0a1 2-byte user pointer
- $f0a2 - $f1b1 272-byte user area for operator task

A total of 434 bytes is dedicated to the FF system.

- $f1b2 - $f1ff Free for application use, up to RAM_HI in p18fxxxx.cfg
- $f6f0 - $ffff Special function registers

For the PIC18F2520 MCU, RAM_HI is $f5e0, leaving 1070 bytes for application use.
pad Address of the temporary area for strings. ( -- addr )
    : pad tib t# + ;
    Each task has its own pad but has zero default size.
    If needed the user must allocate it separately
    with allot for each task.

dp Leave the address of the current data section
dictionary pointer. ( -- addr )
    EEPROM variable mirrored in RAM.

hp Hold pointer for formatted numeric output. ( -- a-addr )

Characters

digit? Convert char to a digit according to base.
    ( c n -- )
>digit Convert n to ascii character value. ( n -- c )
char Parse a character and leave ASCII value. ( -- n )
    For example: char A ( -- 65 )
[char] char Compile inline ASCII character. ( -- )

Strings

Some of these words come from core.txt.
s" text" Compile string into flash. ( -- )
    At run time, leaves address and length.
    ( -- addr u )
." text" Compile string to print into flash.
    ( -- )
place Place string from a1 to a2 as a counted string. ( addr1 u addr2 -- )
count Leave the address and length of text portion
    of a counted string ( addr1 -- addr2 n )
    Compare strings in RAM(a) and flash(nfa).
    Leave true if strings match, n < 16.
    ( addr nfa u -- f )
/string Trim string. ( addr u n -- addr+n u-n )
/number Convert string to a number. ( 0 0 addr1 u1 -- 1 ud.l ud.h addr2 u2 )
/number? Convert string to a number and flag.
    ( addr1 -- addr2 0 | n 1 | d.d d h 2 )
    Prefix: # decimal, % hexadecimal, \% binary.
type Type line to terminal, u < $100. ( addr u -- )
accept Get line from the terminal. ( c addr+1 n1 -- n2 )
    At most n1 characters are accepted, until the line
    is terminated with a carriage return.
source Leave address and length of input buffer.
    ( -- addr u u addr + )
evaluate Interpret a string in SRAM. ( addr n -- )

Pictured numeric output

Formatted string representing an unsigned double-precision integer is
constructed in the end of tib.
<# Begin conversion to formatted string. ( -- )
# Convert 1 digit to formatted string. ( u1 -- u2 )
#s Convert remaining digits. ( u1 -- u2 )
    Note that u2 will be zero.
hold Append char to formatted string. ( c -- )
sign Add minus sign to formatted string, if n<0. ( n -- )
#> End conversion, leave address and count
    of formatted string. ( u1 -- c addr u )

For example, the following:
    -1 34. <# # # # rot sign #> type
    results in -034 ok

Defining functions

Colon definitions

define : Begin colon definition. ( -- )
    ; End colon definition. ( -- )
    [ ] Enter interpreter state. ( -- )
    ] Enter compilation state. ( -- )
    [i Enter Forth interrupt context. ( -- )
    i] Enter compilation state. ( -- )
    ;i End an interrupt word. ( -- )
literal Compile value on stack at compile time.
    At run time, leave value on stack. ( -- x )
inline name Compile the following word. ( -- )
    Mark the last compiled word as inline. ( -- )
    immediate Mark latest definition as immediate. ( -- )
    postpone name Postpone action of immediate word. ( -- )
    see name Show definition. Load see.txt.

Comments

( comment text ) Inline comment.
    ( comment text ) Skip rest of line.

Examples

: square ( n -- n**2 ) Example with stack comment.
    ...body of definition.
    dup * ;
    poke0 ( -- ) Example of using assembler.
    [ $f8a 0 a, bsf, ]

Flow control

Structured flow control

if xxx else yyy then Conditional execution. ( f -- )
    begin xxx again Infinite loop. ( -- )
    begin xxx cond until Loop until cond is true. ( -- )
    begin xxx cond while Loop while cond is true. ( -- )
    for xxx repeat yyy is not executed on the last iteration.
    for xxx next Loop u times. ( u -- )
    leave r0 gets the loop counter u-1 ... 0
    sets r0 to zero so that we leave
    a for loop when next is encountered.
    ( -- )

Unstructured flow control

exit Exit from a word. ( -- )
    If exiting from within a for loop,
    we must drop the loop count with rdrop.
?abort If flag is false, print message
    and abort. ( f addr u -- )
?abort? If flag is false, output '? and abort. ( f -- )
    abort" xxx" If flag is false, output 'xxx' and abort.
    ( -- )
quit Interpret from keyboard. ( -- )
cold Make a cold start.
    Reset all dictionary pointers.
warn Make a warn start.
    Note that irq vector is cleared.

Function pointers (vectors)

' name Search for name and leave its
    execution token. ( -- addr )
['] name Search for name and compile
    it's execution token. ( -- )
execute Execute word at address. ( addr -- )
    The actual stack effect will depend on
    the word executed.
@sex Fetch vector from addr and execute.
    ( addr -- )
defer vec-name Define a deferred execution vector. ( -- )
    Store execution token in vec-name.
    ( addr -- )
vec-name Execute the word whose execution token
    is stored in vec-name's data space.
    ...in the Forth interrupt context.
' my-app is turnkey Disable turnkey application.
    ...at every warm start.

Interrupt example

ram variable icnt1 ...from FF source.
    ...so that we can install the
    irq_forth is irq
    ...in the Forth interrupt context.
    Autostart my-app
    Set the user interrupt vector.
    Alternatively, compile a word
    and compile
    ...so that we can install the
    ...interrupt service function
    ...at every warm start.
Multitasking

Load the words for multitasking from task.txt.

task
Define a new task in flash memory space
(tsize stacksize rsize adsize --)
Use ram xxx allot to leave space for the PAD
of the previously defined task.
The OPERATOR task does not use PAD.
tinit
Initialise a user area and link it
to the task loop. (taskloop-push task-name --)
Note that this may only be executed from
the operator task.
run
Makes a task run by inserting it after operator
in the round-robin linked list. (task-address --)
May only be executed from the operator task.
end
Remove a task from the task list. (task-address --)
May only be executed from the operator task.
single
End all tasks except the operator task. ( --)
Removes all tasks from the task list.

tasks
List all running tasks. ( -- )
pause
Switch to the next task in the
round robin task list. ( -- )

his
Check access user variables of other task.
(task-address varname -- address)
load
Leave the CPU load on the stack. ( -- n)
load is percentage of time that the CPU is busy.
Updated every 256 milliseconds.
busy
CPU idle mode not allowed. ( -- )
idle
CPU idle is allowed. ( -- )
operator
Leave the address of the operator task. ( -- )
unlink
Link to next task. ( -- address )

Defining compound data objects

create name
Create a word definition and store
the current data section pointer.
does>
Define the runtime action of a created word.
allot
Advance the current data section dictionary
pointer by u bytes. (u --)
.x
Append x to the current data section. (x --)
cf
Append c to the file register dictionary. (c --)
cf
Compile xt into the flash dictionary. (addr --)
cfn
Convert code field addr to name field addr.
(adr1 -- adr2)

Examples

Examples

ram
create my-array 20 allot
my-array 20 $ff fill
my-array 20 dump
create my-cell-array
100 300 5
my-cell-array 2 cells +
create my-byte-array
18 c 21 c 255 c
my-byte-array 2 chars +
create my-byte-array
18 0 my-bytes c!
21 1 my-bytes c!
255 2 my-bytes c!
2 my-bytes C0
create my-cell-array
does> +
10 my-byte-array my-bytes
create allocate
...to make byte array objects
...as shown in FF user's guide.
does> ;
10 my-byte-array my-bytes
create allocate
...to make byte array objects.
does> +
3000 0 my-cells !
65000 1 my-cells !
63000 2 my-cells !
1 my-cells @

Dictionary manipulation

marker -my-mark
Mark the dictionary state with -my-mark.

Examples

Examples

my-byte-array 2 chars + c@
my-byte-array 20 dump
my-byte-array 20 $ff fill
my-byte-array 20 dump
create my-cell-array
100 300 5
my-cell-array 2 cells +
create my-byte-array
18 c 21 c 255 c
my-byte-array 2 chars + c0:

Dictionary manipulation

marker -my-mark
Mark the dictionary state with -my-mark.

Examples

Examples

my-byte-array 2 chars + c@
my-byte-array 20 dump
my-byte-array 20 $ff fill
my-byte-array 20 dump
create my-cell-array
100 300 5
my-cell-array 2 cells +
create my-byte-array
18 c 21 c 255 c
my-byte-array 2 chars + c0:

Dictionary manipulation

marker -my-mark
Mark the dictionary state with -my-mark.

Examples

Examples

my-byte-array 2 chars + c@
my-byte-array 20 dump
my-byte-array 20 $ff fill
my-byte-array 20 dump
create my-cell-array
100 300 5
my-cell-array 2 cells +
create my-byte-array
18 c 21 c 255 c
my-byte-array 2 chars + c0:

Dictionary manipulation

marker -my-mark
Mark the dictionary state with -my-mark.

Examples

Examples

my-byte-array 2 chars + c@
my-byte-array 20 dump
my-byte-array 20 $ff fill
my-byte-array 20 dump
create my-cell-array
100 300 5
my-cell-array 2 cells +
create my-byte-array
18 c 21 c 255 c
my-byte-array 2 chars + c0:

Dictionary manipulation

marker -my-mark
Mark the dictionary state with -my-mark.

Examples

Examples

my-byte-array 2 chars + c@
my-byte-array 20 dump
my-byte-array 20 $ff fill
my-byte-array 20 dump
create my-cell-array
100 300 5
my-cell-array 2 cells +
create my-byte-array
18 c 21 c 255 c
my-byte-array 2 chars + c0:

Dictionary manipulation

marker -my-mark
Mark the dictionary state with -my-mark.

Examples

Examples

my-byte-array 2 chars + c@
my-byte-array 20 dump
my-byte-array 20 $ff fill
my-byte-array 20 dump
create my-cell-array
100 300 5
my-cell-array 2 cells +
create my-byte-array
18 c 21 c 255 c
my-byte-array 2 chars + c0:

Dictionary manipulation

marker -my-mark
Mark the dictionary state with -my-mark.

Examples

Examples

my-byte-array 2 chars + c@
my-byte-array 20 dump
my-byte-array 20 $ff fill
my-byte-array 20 dump
create my-cell-array
100 300 5
my-cell-array 2 cells +
create my-byte-array
18 c 21 c 255 c
my-byte-array 2 chars + c0:

Dictionary manipulation

marker -my-mark
Mark the dictionary state with -my-mark.

Examples

Examples

my-byte-array 2 chars + c@
my-byte-array 20 dump
my-byte-array 20 $ff fill
my-byte-array 20 dump
create my-cell-array
100 300 5
my-cell-array 2 cells +
create my-byte-array
18 c 21 c 255 c
my-byte-array 2 chars + c0:

Dictionary manipulation

marker -my-mark
Mark the dictionary state with -my-mark.

Examples

Examples

my-byte-array 2 chars + c@
my-byte-array 20 dump
my-byte-array 20 $ff fill
my-byte-array 20 dump
create my-cell-array
100 300 5
my-cell-array 2 cells +
create my-byte-array
18 c 21 c 255 c
my-byte-array 2 chars + c0:

Dictionary manipulation

marker -my-mark
Mark the dictionary state with -my-mark.

Examples

Examples

my-byte-array 2 chars + c@
my-byte-array 20 dump
my-byte-array 20 $ff fill
my-byte-array 20 dump
create my-cell-array
100 300 5
my-cell-array 2 cells +
create my-byte-array
18 c 21 c 255 c
my-byte-array 2 chars + c0:

Dictionary manipulation

marker -my-mark
Mark the dictionary state with -my-mark.

Examples

Examples

my-byte-array 2 chars + c@
my-byte-array 20 dump
my-byte-array 20 $ff fill
my-byte-array 20 dump
create my-cell-array
100 300 5
my-cell-array 2 cells +
create my-byte-array
18 c 21 c 255 c
my-byte-array 2 chars + c0:

Dictionary manipulation

marker -my-mark
Mark the dictionary state with -my-mark.

Examples

Examples

my-byte-array 2 chars + c@
my-byte-array 20 dump
my-byte-array 20 $ff fill
my-byte-array 20 dump
create my-cell-array
100 300 5
my-cell-array 2 cells +
create my-byte-array
18 c 21 c 255 c
my-byte-array 2 chars + c0:
Low-level flow control

bra, Branch unconditionally. ( rel-addr -- )
call, Call subroutine. ( addr -- )
cirwdt, Clear watchdog timer. ( -- )
daw, Decimal adjust WREG. ( -- )
goto, Go to address. ( addr -- )
nop, No operation. ( -- )
pop, Pop top of return stack. ( -- )
push, Push top of return stack. ( -- )
rcall, Relative call. ( rel-addr -- )
reset, Software device reset. ( -- )
retfie, Return from interrupt enable. ( -- )
return, Return from subroutine. ( -- )
sleep, Go into standby mode. ( -- )

Structured flow control

if, xxx else, yyy then, Conditional execution. ( cc -- )
begin, xxx again, Loop indefinitely. ( -- )
begin, xxx cc until, Loop until condition is true. ( -- )

Conditions for structured flow control

cc, test carry ( -- cc )
nc, test not carry ( -- cc )
mi, test negative ( -- cc )
pl, test not negative ( -- cc )
z, test zero ( -- cc )
nz, test not zero ( -- cc )
ov, test overflow ( -- cc )
ov, test not overflow ( -- cc )
not, invert condition ( cc -- not-cc )

Extras

I2C communications as master

Load these words from i2c_base.txt.
i2cinit Initialize I2C master mode, 100kHz clock. ( -- )
i2cws Wake slave. Bit 0 is R/W bit. ( slave-addr -- )
   The 7-bit I2C address is in bits 7-1.
i2c? Write one byte to I2C bus and wait for ACK. ( c -- )
i2c@ Read one byte and continue. ( -- c )
i2c@r Write 8-bit address to slave. ( addr slave-addr -- )
i2c@r Write 16-bit address to slave ( addr slave-addr -- )

Lower-level words.
sseen Assert start condition. ( -- )
srsen Assert repeated start condition. ( -- )
spen Generate a stop condition. ( -- )
srce Set receive enable. ( -- )
snoack Send not-acknowledge. ( -- )
sack Send acknowledge bit. ( -- )
sspbuf! Write byte to SSPBUF and wait for transmission. ( c -- )

This guide assembled by Peter Jacobs, School of Mechanical Engineering, The University of Queensland, May – 07-Jan-2013 as Report 2012/06.
It is a remix of material from the following sources:
FlashForth v3.8 source code and word list by Mikael Nordman
http://flashforth.sourceforge.net/
EK Conklin and ED Rather Forth Programmer’s Handbook 3rd Ed. 2007 FORTH, Inc.