

FlashForth 5 Quick Reference for PIC and AVR 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. Some words are compile-time use only and cannot be used interpretively. These are [coloured blue](#).

Data and the stack

The data stack (S:) is directly accessible and has 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).
u, u1, u2	Unsigned integers (16-bit).
x, x1, x2, x3	Single-cell item (16-bit).
c	Character value (8-bit).
d ud	Double-cell signed and unsigned (32-bit).
t ut	Triple-cell signed and unsigned (48-bit).
q uq	Quad-cell signed and unsigned (64-bit).
f	Boolean flag: 0 is false, -1 is true.
flt flt1 flt3	Floating-point value (32-bit).
	PIC24-30-33 only, with build option.
addr, addr1, addr2	16-bit addresses.
a-addr	cell-aligned address.
c-addr	character or byte address.
addr.d	32-bit address.

Numbers and values

2	Leave integer two onto the stack. (-- 2)
#255	Leave decimal 255 onto the stack. (-- 255)
%11	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 base 10. (--)
hex	Set number format to hexadecimal. (--)
bin	Set number format to binary. (--)
s>d	Sign extend single to double number. (n -- d) Since double numbers have the most significant bits in the cell above the least significant bits, you can just drop the top cell to recover the single number, provided that the value is not too large to fit in a single cell.
d>q	Extend double to quad-cell number. (d -- q) Requires qmath.h to be loaded. PIC18, PIC24-30-33.

Displaying data

.	Display a number. (n --)
u.	Display u unsigned. (u --)
u.r	Display u with field width n, 0 < n < 256. (u n --)
d.	Display double number. (d --)
ud.	Display unsigned double number. (ud --)
.s	Display stack content (nondestructively).
.st	Emit status string for base, current data section, and display the stack contents. (--)
?	Display content at address. (addr --) PIC24-30-33
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 second item to top. (x1 x2 -- x1 x2 x1)
drop	Discard top item. (x --)
nip	Remove x1 from the stack. (x1 x2 -- x2)
rot	Rotate top three items. (x1 x2 x3 -- x2 x3 x1)
tuck	Insert x2 below x1 in the stack. (x1 x2 -- x2 x1 x2)
pick	Duplicate the u-th item on top. (xu ... x0 u -- xu ... x0 xu)
2dup	Duplicate top double-cell item. (d -- d d)
2swap	Swap top two double-cell items. (d1 d2 -- d2 d1)
2over	Copy second double item to top. (d1 d2 -- d1 d2 d1)
2drop	Discard top double-cell item. (d --)
>r	Send to return stack. S:(n --) R:(-- n)
r>	Take from return stack. S:(-- n) R:(n --)
r@	Copy top item of return stack. S:(-- n) R:(n -- n)
rdrop	Discard top item of return stack. S:(--) R:(n --)
sp@	Leave data stack pointer. (-- addr)
sp!	Set the data stack pointer to address. (addr --)

Operators

Arithmetic with single-cell numbers

Some of these words require **core.txt** and **math.txt**.

+	Add. (n1 n2 -- n1+n2) sum
-	Subtract. (n1 n2 -- n1-n2) difference
*	Multiply. (n1 n2 -- n1*n2) product
/	Divide. (n1 n2 -- n1/n2) quotient
mod	Divide. (n1 n2 -- n.rem) remainder
/mod	Divide. (n1 n2 -- n.rem n.quot)
u/	Unsigned 16/16 to 16-bit division. (u1 u2 -- u2/u1)
u/mod	Unsigned division. (u1 u2 -- u.rem u.quot) 16-bit/16-bit to 16-bit
1	Leave one. (-- 1)
1+	Add one. (n -- n1)
1-	Subtract one. (n -- n1)
2+	Add two. (n -- n1)
2-	Subtract 2 from n. (n -- n1)
2*	Multiply by 2; Shift left by one bit. (u -- u1)
2/	Divide by 2; Shift right by one bit. (u -- u1)

*/	Scale. (n1 n2 n3 -- n1*n2/n3) Uses 32-bit intermediate result.
*/mod	Scale with remainder. (n1 n2 n3 -- n.rem n.quot) Uses 32-bit intermediate result.
u*/mod	Unsigned Scale u1*u2/u3 (u1 u2 u3 -- u.rem u.quot) Uses 32-bit intermediate result.
abs	Absolute value. (n -- u)
negate	Negate n. (n -- -n)
?negate	Negate n1 if n2 is negative. (n1 n2 -- n3)
min	Leave minimum. (n1 n2 -- n)
max	Leave maximum. (n1 n2 -- n)
umin	Unsigned minimum. (u1 u2 -- u)
umax	Unsigned maximum. (u1 u2 -- u)

Arithmetic with double-cell numbers

Some of these words require **core.txt**, **math.txt** and **qmath.txt**.

d+	Add double numbers. (d1 d2 -- d1+d2)
d-	Subtract double numbers. (d1 d2 -- d1-d2)
m+	Add single cell to double number. (d1 n -- d2)
m*	Signed 16*16 to 32-bit multiply. (n n -- d)
d2*	Multiply by 2. (d -- d)
d2/	Divide by 2. (d -- d)
um*	Unsigned 16x16 to 32 bit multiply. (u1 u2 -- ud)
ud*	Unsigned 32x16 to 32-bit multiply. (ud u -- ud)
um/mod	Unsigned division. (ud u1 -- u.rem u.quot) 32-bit/16-bit to 16-bit
ud/mod	Unsigned division. (ud u1 -- u.rem ud.quot) 32-bit/16-bit to 32-bit
fm/mod	Floored division. (d n -- n.rem n.quot)
sm/rem	Symmetric division. (d n -- n.rem n.quot) 32-bit/16-bit to 16-bit.
m*/	Scale with triple intermediate result. d2 = d1*n1/n2 (d1 n1 n2 -- d2)
um*/	Scale with triple intermediate result. ud2 = ud1*u1/u2 (ud1 u1 u2 -- ud2)
dabs	Absolute value. (d -- ud)
dnegate	Negate double number. (d -- -d)
?dnegate	Negate d if n is negative. (d n -- -d)

Arithmetic with triple- and quad-numbers

For PIC18, these words require **core.txt**, **math.txt** and **qmath.txt**.

q+	Add a quad to a quad. (q1 q2 -- q3) For PIC24-30-33.
qm+	Add double to a quad. (q1 d -- q2) For PIC18 and PIC24-30-33.
uq*	Unsigned 32x32 to 64-bit multiply. (ud ud -- uq) For PIC18 and PIC24-30-33.
ut*	Unsigned 32x16 to 48-bit multiply. (ud u -- ut)
ut/	Divide triple by single. (ut u -- ud)
uq/mod	Divide quad by double. (uq ud -- ud-rem ud-quot)

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)
Inverts logical value.
0< Leave true if n is negative. (n -- f)
within Leave true if xl <= x < xh. (x x1 xh -- f)
u< Leave true if u1 < u2. (u1 u2 -- f)
u> Leave true if u1 > u2. (u1 u2 -- f)
d= Leave true if d1 d2 are equal. (d1 d2 -- f)
d0= Leave true if d is zero. (d -- f)
d0< Leave true if d is negative. (d -- f)
d< Leave true if d1 < d2. (d1 d2 -- f)
d> Leave true if d1 > d2. (d1 d2 -- f)

Bitwise

invert Ones complement. (x -- x)
dinvert Invert double number. (du -- du)
and Bitwise and. (x1 x2 -- x)
or Bitwise or. (x1 x2 -- x)
xor Bitwise exclusive-or. (x -- x)
lshift Left shift by u bits. (x1 u -- x2)
rshift Right shift by u bits. (x1 u -- x2)

Memory

Typically, the microcontroller has three distinct memory contexts: Flash, EEPROM and SRAM. FlashForth unifies these memory spaces into a single 64kB address space.

PIC18 Memory map

The address ranges are:

\$0000 – \$ebff Flash
\$ec00 – \$efff EEPROM
\$f000 – \$ff5f SRAM, general use
\$ff60 – \$ffff SRAM, special function registers

The high memory mark for each context will depend on the particular device used. Using a PIC18F26K22 and the default values in p18f-main.cfg for the UART version of FF, a total of 423 bytes is dedicated to the FF system. The rest (3473 bytes) is free for application use. Also, the full 64kB of Flash memory is truncated to fit within the range specified above.

PIC24 Memory map

A device with EEPROM will have its 64kB address space divided into:

\$0000 – \$07ff SRAM, special function registers
\$0800 – (\$0800+RAMSIZE-1) SRAM, general use
(\$0800+RAMSIZE) – \$bfff Flash
\$fc00 – \$ffff EEPROM

The high memory mark for the Flash context will depend on the device. Also, the full Flash memory of the device may not be accessible.

AVR8 Memory map

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

\$0000 – (RAMSIZE-1) SRAM
RAMSIZE – (RAMSIZE+EEPROMSIZE-1) EEPROM
(\$ffff-FLASHSIZE+1) – \$ffff Flash

The SRAM space includes the IO-space and special function registers. The high memory mark for the Flash context is set by the combined size of the boot area and FF kernel.

Memory Context

ram Set address context to SRAM. (--)
eeprom Set address context to EEPROM. (--)
flash Set address context to Flash. (--)
fl- Disable writes to Flash, EEPROM. (--)
fl+ Enable writes to Flash, EEPROM, default. (--)
iflush Flush the flash write buffer. (--)
here Leave the current data section dictionary pointer. (-- addr)
align Align the current data section dictionary pointer to cell boundary. (--)
hi Leave the high limit of the current data space. (-- u)

Accessing Memory

! Store x to address. (x a-addr --)
@ Fetch from address. (a-addr -- x)
@+ Fetch cell and increment address by cell size.
(a-addr1 -- a-addr2 x)
2! Store 2 cells to address. (x1 x2 a-addr --)
2@ Fetch 2 cells from address. (a-addr -- x1 x2)
c! Store character to address. (c addr --)
c@ Fetch character from address. (addr -- c)
c@+ Fetch char, increment address.
(addr1 -- addr2 c)
+! Add n to cell at address. (n addr --)
-@ Fetch from addr and decrement addr by 2.
(addr1 -- addr2 x)
>a Write to the A register. (x --)
a> Read from the A register. (-- x)

Accessing Extended (Flash) Memory

x! Store u to real flash address. (u addr.d --)
PIC18, AVR8
x! Store u to real flash address. (ud addr.d --)
PIC24-30-33
x@ Fetch from real flash address. (addr.d -- u)
PIC18, AVR8
x@ Fetch from real flash address. (addr.d -- ud)
PIC24-30-33

Accessing bits in RAM

mset Set bits in file register with mask c. (c addr --)
For PIC24-30-33, the mask is 16 bits.
mclr Clear bits in file register with mask c. (c addr --)
mtst AND file register byte with mask c. (c addr -- x)

The following come from bit.txt

bit1: name Define a word to set a bit. (addr bit --)
bit0: name Define a word to clear a bit. (addr bit --)
bit?: name Define a word to test a bit. (addr bit --)
When executed, name leaves a flag. (-- f)

The Dictionary

Dictionary management

marker -my-mark Mark the dictionary and memory allocation state with -my-mark.
-my-mark Return to the dictionary and allotted-memory state that existed before -my-mark was created.
find name Find name in dictionary. (-- n)
Leave 1 immediate, -1 normal, 0 not found.
forget name Forget dictionary entries back to name.
empty Reset all dictionary and allotted-memory pointers. (--)
words List all words in dictionary. (--)
words xxx List words containing xxx. (--)

Defining constants and variables

constant name Define new constant. (n --)
2constant name Define double constant. (x x --)
name Leave value on stack. (-- n)
variable varname Define a variable in the current data section. (--)
Use ram, eeprom or flash to set data section.
2variable name Define double variable. (--)
varname Leave address on stack. (-- addr)
value valname Define value. (n --)
to valname Assign new value to valname. (n --)
valname Leave value on stack. (-- n)
user name Define a user variable at offset +n. (+n --)

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 constant portb Define constant in Flash.
3 value xx Define value in SRAM.
variable yy Define variable in SRAM.
6 yy ! Store 6 in variable yy.
eeprom 5 value zz ram Define value in EEPROM.
xx yy zz portb yy @ Leaves 3 f172 5 ff81 6

```
warm           Warm restart clears SRAM data.
xx yy zz portb yy @
4 to xx        Leaves 0 f172 5 ff81 0
               Sets new value.
xx yy zz portb yy @
hi here - u.    Prints the number of bytes free.
$ff8a constant latb
$ff93 constant trisb
%00000010 trisb mclr
latb 1 bit1: pb1-high
pb1-high       Defines a word to set RB1 high.
               Sets RB1 high.
```

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 -- )
,              Append x to the current data section. ( x -- )
c,             Append c to the current data section. ( c -- )
," xxx"        Append a string at HERE. ( -- )
i,             Append x to the flash data section. ( x -- )
ic,            Append c to the flash data section. ( c -- )
cf,            Compile xt into the flash dictionary. ( addr -- )
c>n            Convert code field addr to name field addr.
               ( addr1 -- addr2 )
n>c            Convert name field addr to code field addr.
               ( addr1 -- addr2 )
n>l            Convert nfa to lfa. ( nfa -- lfa )
               Not implemented; use 2- instead.
>body          Leave the parameter field address of the created
               word. ( xt -- a-addr )
:noname        Define headerless forth code. ( -- addr )
>xa            Convert a Flash virtual address to a real executable
               address. PIC24-30-33, ATmega ( a-addr1 -- a-addr2 )
xa>            Convert a real executable address to a Flash virtual
               address. PIC24-30-33, ATmega ( a-addr1 -- a-addr2 )
```

Array examples

```
ram            Example
create my-array 20 allot    ...of creating an array,
my-array 20 $ff fill        ...filling it with 1s, and
my-array 20 dump            ...displaying its content.
create my-cell-array
  100 , 340 , 5 ,           Initialised cell array.
my-cell-array 2 cells + @   Should leave 5. ( -- x )
create my-byte-array
  18 c, 21 c, 255 c,        Initialised byte array.
my-byte-array 2 chars + c@  Should leave 255. ( -- c )
```

```
: mk-byte-array      Defining word ( n -- )
  create allot        ...to make byte array objects
  does> + ;           ...as shown in FF user's guide.
10 mk-byte-array my-bytes
                       Creates an array object
                       my-bytes ( n -- addr ).
                       Sets an element
                       ...and another.
18 0 my-bytes c!
21 1 my-bytes c!
255 2 my-bytes c!
2 my-bytes c@         Should leave 255.
: mk-cell-array       Defining word ( n -- )
  create cells allot  ...to make cell array objects.
  does> swap cells + ;
5 mk-cell-array my-cells
                       Creates an array object
                       my-cells ( n -- addr ).
3000 0 my-cells !      Sets an element
45000 1 my-cells !     ...and another.
63000 2 my-cells !
1 my-cells @ .         Should print 45000
```

Memory operations

Some of these words come from `core.txt`.

```
cmove          Move u bytes from address-1 to address-2.
               ( addr1 addr2 u -- )
               Copy proceeds from low addr to high address.
wmove          Move u cells from address-1 to address-2.
               ( addr1 addr2 u -- ) PIC24-30-33 only
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 -- )
cells          Convert cells to address units. ( u -- u )
chars          Convert chars to address units. ( u -- u )
char+          Add one to address. ( addr1 -- addr2 )
cell+          Add size of cell to address. ( addr1 -- addr2 )
aligned        Align address to a cell boundary. ( addr -- a-addr )
```

Predefined constants

```
cell           Size of one cell in characters. ( -- n )
true           Boolean true value. ( -- -1 )
false          Boolean false value. ( -- 0 )
bl            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 )
               EEPROM value mirrored in SRAM.
prompt         Deferred execution vector for the info displayed
               by quit. Default value is .st ( -- a-addr )
'emit          EMIT vector. Default is tx1 ( -- a-addr )
'key           KEY vector. Default is rx1 ( -- a-addr )
'key?          KEY? vector. Default is rx1? ( -- a-addr )
'source        Current input source. ( -- a-addr )
s0             Variable for start of data stack. ( -- a-addr )
r0             Bottom of return stack. ( -- a-addr )
rcnt           Number of saved return stack cells. ( -- a-addr )
tib            Address 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 )
pad            Address of the temporary area for strings. ( -- addr )
: pad tib ti# + ;
               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.
dps            End address of dictionary pointers. ( -- d )
               Absolute address of start of free Flash.
               Library and C code can be linked,
               starting at this address. PIC24, dsPIC33
hp             Hold pointer for formatted numeric output.
               ( -- a-addr )
up            Variable holding a user pointer. ( -- addr )
latest         Variable holding the address of the latest
               defined word. ( -- a-addr )
float?         Interpreter defer for parsing floating-point values.
               ' >float is float?
               PIC24-30-33 only
```

Floating-point for PIC24-30-33

These words require that FlashForth has been built with the `.eq FLOATS, 1` option in the relevant processor config file.

```
>float         Convert a string into a float. ( c-addr u -- flt f )
               Note that it works for decimal base only.
               Examples: 1e10 -1e10 1.234e10 -1.234e10
f.             Print in decimal format. ( flt -- )
fe.            Print in engineering format. ( flt -- )
fs.            Print in scientific format. ( flt -- )
```

```

fdrop  Discard top float item. ( flt -- )
fdup   Duplicate top float item. ( flt -- flt flt )
fover  Copy second float item to top.
       ( flt1 flt2 -- flt1 flt2 flt1 )
fswap  Swap top two float items. ( flt1 flt2 -- flt2 flt1 )
frot   Rotate top three float items.
       ( flt1 flt2 flt3 -- flt2 flt3 flt1 )
fnip   Remove second top float. ( flt1 flt2 -- flt2 )
ftuck  Insert flt2 below flt1.
       ( flt1 flt2 -- flt2 flt1 flt2 )
nfswap Swap float and single. ( flt n -- n flt )
fnswap Swap float and single. ( n flt -- flt n )
nfover Copy float item over single. ( flt n -- flt n flt )
fnover Copy single over float item. ( n flt -- n flt n )
f@     Fetch float item to stack. ( addr -- flt )
f!     Store float item to address. ( flt addr -- )

fconstant name Define constant. ( flt -- )
fvariable name Define variable. ( -- )
fliteral      Compile in literal value. ( flt -- )
f0            Leave value 0.0 on stack. ( -- flt )
f1            Leave value 1.0 on stack. ( -- flt )
f10           Leave value 10.0 on stack. ( -- flt )
f0.5          Leave value 0.5 on stack. ( -- flt )

s>f  Convert single to float. ( n -- flt )
d>f  Convert double to float. ( d -- flt )
f>s  Convert float to single. ( flt -- n )
f>d  Convert float to double. ( flt -- d )
f0=  Leave true if flt equal to zero. ( flt -- f )
f0<  Leave true if flt less than zero. ( flt -- f )
f=   Leave true if floats are equal. ( flt1 flt2 -- f )
f<   Leave true if flt1 less than flt2. ( flt1 flt2 -- f )
f<=  eave true if flt1 less than or equal to flt2.
     ( flt1 flt2 -- f )
f>   Leave true if flt1 greater than flt2.
     ( flt1 flt2 -- f )
f>=  Leave true if flt1 greater than or equal to flt2.
     ( flt1 flt2 -- f )

fnegate  Negate float value. ( flt -- -flt )
fabs     Leave absolute value. ( flt1 -- flt2 )
fround   Round to nearest integral value. ( flt1 -- flt2 )
fmin     Leave minimum. ( flt1 flt2 -- flt )
fmax     Leave maximum. ( flt1 flt2 -- flt )
f2*      Multiple by 2. ( flt -- flt*2 )
f2/      Divide by 2. ( flt -- flt/2 )
The following functions call out to the Microchip math library.
f+  Addition ( flt1 flt2 -- flt1+flt2 )
f-  Subtraction ( flt1 flt2 -- flt1-flt2 )
f*  Multiplication ( flt1 flt2 -- flt1*flt2 )
f/  Division ( flt1 flt2 -- flt1/flt2 )
fpow  Power. ( flt1 flt2 -- flt1**flt2 )
fsin  Sine of flt in radians. ( flt -- sin(flt) )
fcos  Cosine of flt in radians. ( flt -- cos(flt) )
ftan  Tangent of flt in radians. ( flt -- tan(flt) )

```

```

fasin  Arcine of flt, radians. ( flt -- asin(flt) )
facos  Arccosine of flt, radians. ( flt -- acos(flt) )
fatan  Arctangent of flt, radians. ( flt -- atan(flt) )
fatan2 Arctangent of flt1/flt2, radians.
       ( flt1 flt2 -- atan(flt1/flt2) )
fsqrt  Square-root. ( flt -- sqrt(flt) )
fexp   Exponential. ( flt -- exp(flt) )
flog   Natural logarithm. ( flt -- loge(flt) )
flog10 Logarithm, base 10. ( flt -- log10(flt) )
fcosh  Hyperbolic cosine. ( flt -- cosh(flt) )
fsinh  Hyperbolic sine. ( flt -- sinh(flt) )
ftanh  Hyperbolic tangent. ( flt -- tanh(flt) )

```

The Compiler

Defining functions

```

:      Begin colon definition. ( -- )
;      End colon definition. ( -- )
[      Enter interpreter state. ( -- )
]      Enter compilation state. ( -- )
state  Compilation state. ( -- f )
       State can only be changed with [ and ].

[i]    Enter Forth interrupt context. ( -- )
       PIC18, PIC24-30-33
[i]    Enter compilation state. ( -- )
       PIC18, PIC24-30-33
;i     End an interrupt word. ( -- )

literal  Compile value on stack at compile time. ( x -- )
         At run time, leave value on stack. ( -- x )
2literal Compile double value on stack at compile time.
         ( x x -- )
         At run time, leave value on stack. ( -- x x )

inline name  Inline the following word. ( -- )
inlined     Mark the last compiled word as inlined. ( -- )
immediate   Mark latest definition as immediate. ( -- )
immed?      Leave a nonzero value if addr contains
             an immediate flag. ( addr -- f )
in?         Leave a nonzero flag if nfa has inline bit
             set. ( nfa -- f )
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 of colon definitions

```

: square ( n -- n**2 )  Example with stack comment.
  dup * ;               ...body of definition.
: poke0 ( -- )          Example of using PIC18 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. ( -- )
                    yyy repeat      yyy is not executed on the last iteration.
for xxx next          Loop u times. ( u -- )
                    r@ gets the loop counter u-1 ... 0
endit                Sets loop counter to zero so that we leave
                    a for loop when next is encountered.
                    ( -- )

```

From `doloop.txt`, we get the ANSI loop constructs which iterate from *initial* up to, but not including, *limit*:

```

limit initial do words-to-repeat loop
limit initial do words-to-repeat value +loop
i      Leave the current loop index. ( -- n )
       Innermost loop, for nested loops.
j      Leave the next-outer loop index. ( -- n )
leave  Leave the do loop immediately. ( -- )
?do    Starts a do loop which is not run if
       the arguments are equal. ( limit initial -- )

```

Loop examples

```

decimal
: sumdo 0 100 0 do i + loop ;          sumdo leaves 4950
: sumfor 0 100 for r@ + next ;          sumfor leaves 4950
: print-twos 10 0 do i u. 2 +loop ;

```

Case example

From `case.txt`, we get words `case`, `of`, `endof`, `default` and `endcase` to define case constructs.

```

: testcase
  4 for r@
    case
      0 of ." zero " endof
      1 of ." one " endof
      2 of ." two " endof
      default ." default " endof
    endcase
  next
;

```

Unstructured flow control

```

exit      Exit from a word. ( -- )
          If exiting from within a for loop,
          we must drop the loop count with rdrop.

abort     Reset stack pointer and execute quit. ( -- )
?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, type out last word executed,
          followed by text xxx. ( f -- )

quit     Interpret from keyboard. ( -- )

```

warm Make a warm start.
Reset reason will be displayed on restart.
S: Reset instruction
E: External reset pin
W: Watchdog reset
U: Return stack underflow
O: Return stack overflow
B: Brown out reset
P: Power on reset
M: Math error
A: Address error
Note that irq vector is cleared.

Vectored execution (Function pointers)

' name Search for *name* and leave its execution token (address). (-- **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.
@ex Fetch vector from *addr* and execute. (**addr** --)
defer vec-name Define a deferred execution vector. (--)
is vec-name Store execution token in *vec-name*. (**addr** --)
vec-name Execute the word whose execution token is stored in *vec-name*'s data space.
int! Store interrupt vector to table. (**xt vector-no** --)
PIC18: **vector-no** is dummy vector number (0) for high priority interrupts.
PIC30: Alternate interrupt vector table in Flash.
PIC33: Alternate interrupt vector table in RAM.
PIC24H: Alternate interrupt vector table in RAM.
PIC24F: Alternate interrupt vector table in RAM.
PIC24FK: Alternate interrupt vector table in Flash.
PIC24E: Main interrupt vector table in RAM.
ATmega: Interrupt vector table in RAM.
int/ Restore the original vector to the interrupt vector table in flash. PIC30 PIC24FK (**vector-no** --)
ivt Activate the normal interrupt vector table. (--)
Not PIC24E, dsPIC33E.
aivt Activate the alternate interrupt vector table. (--)

Autostart example

' my-app is turnkey Autostart my-app.
false is turnkey Disable turnkey application.

Interrupt example

```
ram variable icnt1 ...from FF source.  
: irq_forth          It's a Forth colon definition  
[i                  ...in the Forth interrupt context.  
    icnt1 @ 1+  
    icnt1 !  
  ]i  
;i  
' irq_forth 0 int!    Set the user interrupt vector.  
  
: init                Alternatively, compile a word  
['] irq_forth 0 int!  ...so that we can install the  
;                    ...interrupt service function  
' init is turnkey    ...at every warm start.
```

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(ointer) register. (**addr** --)
@p Fetch the P register to the stack. (-- **addr**)
!p>r Push contents of P to return stack and store new address to P. (**addr** --) (**R:** -- **addr**)
r>p Pop from return stack to P register. (**R:** **addr** --)
p+ Increment P register by one. (--)
p2+ Add 2 to P register. (--)
p++ Add *n* to the p register. (*n* --)
p! Store *x* to the location pointed to by the p register. (*x* --)
pc! Store *c* to the location pointed to by the p register. (*c* --)
p@ Fetch the cell pointed to by the p register. (-- *x*)
pc@ Fetch the char pointed to by the p register. (-- *c*)

In a definition, **!p>r** and **r>p** should always be used to allow proper nesting of words.

Characters

digit? Convert char to a digit according to base. (*c* -- *n f*)
>digit Convert *n* to ascii character value. (*n* -- *c*)
>pr Convert a character to an ASCII value. (*c* -- *c*)
Nongraphic characters converted to a dot.
char 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** --)
n= Compare strings in RAM(*addr*) and Flash(*nfa*). Leave true if strings match, *n* < 16. (**addr nfa u** -- *f*)
scan Scan string until *c* is found. *c-addr* must point to RAM and *u* < 255. (*c-addr u c* -- **caddr1 u1**)
skip Skip chars until *c* not found. *c-addr* must point to RAM and *u* < 255. (*c-addr u c* -- **caddr1 u1**)
/string Trim string. (**addr u n** -- **addr+n u-n**)
>number Convert string to a number. (0 0 **addr1 u1** -- **ud.l ud.h addr2 u2**)
number? Convert string to a number and flag. (**addr1** -- **addr2 0 | n 1 | d.l d.h 2**)
Prefix: # decimal, \$ hexadecimal, % binary.
sign? Get optional minus sign. (**addr1 n1** -- **addr2 n2 flag**)
type Type line to terminal, *u* < #256. (**addr u** --)
accept Get line from the terminal. (*c-addr +n1* -- *+n2*)
At most *n1* characters are accepted, until the line is terminated with a carriage return.
source Leave address of input buffer and number of characters. (-- *c-addr u*)
evaluate Interpret a string in SRAM. (**addr n** --)
interpret Interpret the buffer. (*c-addr u* --)
parse Parse a word in TIB. (*c* -- **addr length**)
word Parse a word in TIB and write length into TIB. Leave the address of length byte on the stack. (*c* -- *c-addr*)

Pictured numeric output

Formatted string representing an unigned double-precision integer is constructed in the end of **tib**.

<# Begin conversion to formatted string. (--)
Convert 1 digit to formatted string. (**ud1** -- **ud2**)
#s Convert remaining digits. (**ud1** -- **ud2**)
Note that **ud2** 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. (**ud1** -- *c-addr u*)

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

Interaction with the operator

Interaction with the user is via a serial communications port, typically UART1. Settings are 38400 baud, 8N1, using Xon/Xoff handshaking. Which particular serial port is selected is determined by the vectors **'emit**, **'key** and **'key?**.

emit	Emit c to the serial port FIFO. (c --) FIFO is 46 chars. Executes pause.
space	Emit one space character. (--)
spaces	Emit n space characters. (n --)
cr	Emit carriage-return, line-feed. (--)
key	Get a character from the serial port FIFO. Execute pause until a character is available. (-- c)
key?	Leave true if character is waiting in the serial port FIFO. (-- f)

Serial communication ports

tx0	Send a character via UART0 on ATmega. (c --)
rx0	Receive a character from UART0 on ATmega. (-- c)
rx0?	Leave true if the UART0 receive buffer is not empty. ATmega (-- f)
u0-	Disable flow control for UART1 interface. (--)
u0+	Enable flow control for UART1 interface, default. (--)
tx1	Send character to UART1. (c --) Buffered via an interrupt driven queue.
rx1	Receive a character from UART1. (-- c) Buffered by an interrupt-driven queue.
rx1?	Leave true if the UART1 receive buffer is not empty. (-- f)
u1-	Disable flow control for UART1 interface. (--)
u1+	Enable flow control for UART1 interface, default. (--)
tx2	Send character to UART2. (c --) PIC24-30-33
rx2	Receive a character from UART2. (-- c) PIC24-30-33
rx2?	Leave true if the UART1 receive buffer is not empty. PIC24-30-33 (-- f)
u2-	Disable flow control for UART2 interface. (--)
u2+	Enable flow control for UART2 interface, default. (--)
txu	Send a character via the USB UART. (c --) PIC18-USB
rxu	Receive a character from the USB UART. (-- c) PIC18-USB
rxu?	Leave true if the USB UART receive buffer is not empty. PIC18-USB (-- f)

Character queues on PIC24-30-33

cq: name	Create character queue. (u --)
cq0	Initialize or reset queue. (queue-addr --)
>cq?	Is there space available in queue. (queue-addr -- f)
>cq	Put character into queue. (c queue-addr --)
cq>?	Number of characters in queue. (queue-addr -- u)
cq>	Get character from queue. (queue-addr -- c)
u1rxq	Leave UART1 RX queue address. (-- queue-addr)
u1txq	Leave UART1 TX queue address. (-- queue-addr)
u2rxq	Leave UART2 RX queue address. (-- queue-addr)
u2txq	Leave UART2 TX queue address. (-- queue-addr)

Other Hardware

cwd	Clear the WatchDog counter. (--) PIC18, PIC24-30-33
ei	Enable interrupts. (--)
di	Disable interrupts. (--)
ms	Pause for +n milliseconds. (+n --)
ticks	System ticks, 0-ffff milliseconds. (-- u)

Multitasking

Load the words for multitasking from task.txt .	
task:	Define a new task in flash memory space (tibsize stacksize rstacksize addsize --) 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-addr task-addr --) Note that this may only be executed from the operator task.
task	Leave the address of the task definition table. (-- addr)
run	Makes a task run by inserting it after operator in the round-robin linked list. (task-addr --) May only be executed from the operator task.
end	Remove a task from the task list. (task-addr --) May only be executed from the operator task.
single	End all tasks except the operator task. (--) Removes all tasks from the task list. May only be executed from the operator task.
tasks	List all running tasks. (--)
pause	Switch to the next task in the round robin task list. Idle in the operator task if allowed by all tasks. (--)
his	Access user variables of other task. (task.addr vvar.addr -- addr)
load	Leave the CPU load on the stack. (-- n) Load is percentage of time that the CPU is busy. Updated every 256 milliseconds.
load+	Enable the load LED on AVR8. (--)
load-	Disable the load LED on AVR8. (--)
busy	CPU idle mode not allowed. (--)
idle	CPU idle is allowed. (--)
operator	Leave the address of the operator task. (-- addr)
ulink	Link to next task. (-- addr)

Structured Assembler

To use many of the words listed in the following sections, load the text file asm.txt . The assembler for each processor family provides the same set of structured flow control words, however, the conditionals that go with these words are somewhat processor-specific.	
if, xxx else, yyy then,	Conditional execution. (cc --)
begin, xxx again,	Loop indefinitely. (--)
begin, xxx cc until,	Loop until condion is true. (--)

Assembler words for PIC18

In the stack-effect notaion for the PIC18 family, f is a file register
address, d is the result destination, a is the access bank modifier,
and k is a literal value.

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)
nov,	test not overflow (-- cc)
not,	invert condition (cc -- not-cc)

Destination and access modifiers

w,	Destination WREG (-- 0)
f,	Destination file (-- 1)
a,	Access bank (-- 0)
b,	Use bank-select register (-- 1)

Byte-oriented file register operations

addwf,	Add WREG and f. (f d a --)
addwfc,	Add WREG and carry bit to f. (f d a --)
andwf,	AND WREG with f. (f d a --)
clrf,	Clear f. (f a --)
comf,	Complement f. (f d a --)
cpfseq,	Compare f with WREG, skip if equal. (f a --)
cpfsgt,	Compare f with WREG, skip if greater than. (f a --)
cpfslt,	Compare f with WREG, skip if less than. (f a --)
decf,	Decrement f. (f d a --)
decfsz,	Decrement f, skip if zero. (f d a --)
dcfsnz,	Decrement f, skip if not zero. (f d a --)
incf,	Increment f. (f d a --)
incfsz,	Increment f, skip if zero. (f d a --)
infsnz,	Increment f, skip if not zero. (f d a --)
iorwf,	Inclusive OR WREG with f. (f d a --)
movf,	Move f. (f d a --)
movff,	Move fs to fd. (fs fd --)
movwf,	Move WREG to f. (f a --)
mulwf,	Multiply WREG with f. (f a --)
negf,	Negate f. (f a --)
rlcf,	Rotate left f, through carry. (f d a --)
rlncf,	Rotate left f, no carry. (f d a --)
rrcf,	Rotate right f, through carry. (f d a --)
rrncf,	Rotate right f, no carry. (f d a --)
setf,	Set f. (f d a --)
subfwb,	Subtract f from WREG, with borrow. (f d a --)
subwf,	Subtract WREG from f. (f d a --)
subwfb,	Subtract WREG from f, with borrow. (f d a --)
swapf,	Swap nibbles in f. (f d a --)
tstfsz,	Test f, skip if zero. (f a --)
xorwf,	Exclusive OR WREG with f. (f d a --)

Bit-oriented file register operations

```
bcf,      Bit clear f. ( f b a -- )
bsf,      Bit set f. ( f b a -- )
btfsc,    Bit test f, skip if clear. ( f b a -- )
btfss,    Bit test f, skip if set. ( f b a -- )
btg,      Bit toggle f. ( f b a -- )
```

Literal operations

```
addlw,    Add literal and WREG. ( k -- )
andlw,    AND literal with WREG. ( k -- )
daw,      Decimal adjust packed BCD digits in WREG. ( -- )
iorlw,    Inclusive OR literal with WREG. ( k -- )
lfsr,     Move literal to FSRx. ( k f -- )
movlb,    Move literal to BSR. ( k -- )
movlw,    Move literal to WREG. ( k -- )
mullw,    Multiply literal with WREG. ( k -- )
sublw,    Subtract WREG from literal. ( k -- )
xorlw,    Exclusive OR literal with WREG. ( k -- )
```

Data memory – program memory operations

```
tblrd*,   Table read. ( -- )
tblrd**,  Table read with post-increment. ( -- )
tblrd*-,  Table read with post-decrement. ( -- )
tblrd+*,  Table read with pre-increment. ( -- )
tblwt*,   Table write. ( -- )
tblwt**,  Table write with post-increment. ( -- )
tblwt*-,  Table write with post-decrement. ( -- )
tblwt+*,  Table write with pre-increment. ( -- )
```

Low-level flow control operations

```
bra,      Branch unconditionally. ( rel-addr -- )
call,     Call subroutine. ( addr -- )
goto,     Go to address. ( addr -- )
pop,      Pop (discard) top of return stack. ( -- )
push,     Push address of next instruction to
top of return stack. ( -- )

rcall,    Relative call. ( rel-addr -- )
retfie,   Return from interrupt enable. ( -- )
retlw,    Return with literal in WREG. ( k -- )
return,   Return from subroutine. ( -- )
```

Other MCU control operations

```
clrwtdt,  Clear watchdog timer. ( -- )
nop,      No operation. ( -- )
reset,    Software device reset. ( -- )
sleep,    Go into standby mode. ( -- )
```

Assembler words for PIC24-30-33

As stated in the wordsAll.txt, there is only a partial set of words for these families of microcontrollers.

Conditions for structured flow control

```
z,        test zero ( -- cc )
nz,       test not zero ( -- cc )
not,      invert condition ( cc -- not-cc )
```

Low-level flow control operations

```
bra,      Branch unconditionally. ( rel-addr -- )
rcall,    Call subroutine. ( rel-addr -- )
return,   Return from subroutine. ( -- )
retfie,   Return from interrupt enable. ( -- )
```

Bit-oriented operations

```
bclr,     Bit clear. ( bit ram-addr -- )
bset,     Bit set. ( bit ram-addr -- )
btst,     Bit test to z. ( bit ram-addr -- )
btsc,     Bit test, skip if clear. ( bit ram-addr -- )
btss,     Bit test, skip if set. ( bit ram-addr -- )
```

Assembler words for AVR8

For the ATmega instructions, Rd denotes the destination (and source) register, Rr denotes the source register, Rw denotes a register-pair code, K denotes constant data, k is a constant address, b is a bit in the register, x,Y,Z are indirect address registers, A is an I/O location address, and q is a displacement (6-bit) for direct addressing.

Conditions for structured flow control

```
cs,       carry set ( -- cc )
eq,       zero ( -- cc )
hs,       half carry set ( -- cc )
ie,       interrupt enabled ( -- cc )
lo,       lower ( -- cc )
lt,       less than ( -- cc )
mi,       negative ( -- cc )
ts,       T flag set ( -- cc )
vs,       no overflow ( -- cc )
not,      invert condition ( cc -- not-cc )
```

Register constants

```
Z      ( -- 0 )
Z+     ( -- 1 )
-Z     ( -- 2 )
Y      ( -- 8 )
Y+     ( -- 9 )
-Y     ( -- 10 )
X      ( -- 12 )
X+     ( -- 13 )
-X     ( -- 14 )
XH:XL  ( -- 01 )
YH:YL  ( -- 02 )
ZH:ZL  ( -- 03 )
```

R0	(-- 0)	R16	(-- 16)
R1	(-- 1)	R17	(-- 17)
R2	(-- 2)	R18	(-- 18)
R3	(-- 3)	R19	(-- 19)
R4	(-- 4)	R20	(-- 20)
R5	(-- 5)	R21	(-- 21)
R6	(-- 6)	R22	(-- 22)
R7	(-- 7)	R23	(-- 23)
R8	(-- 8)	R24	(-- 24)
R9	(-- 9)	R25	(-- 25)
R10	(-- 10)	R26	(-- 26)
R11	(-- 11)	R27	(-- 27)
R12	(-- 12)	R28	(-- 28)
R13	(-- 13)	R29	(-- 29)
R14	(-- 14)	R30	(-- 30)
R15	(-- 15)	R31	(-- 31)

Arithmetic and logic instructions

```
add,      Add without carry. ( Rd Rr -- )
adc,      Add with carry. ( Rd Rr -- )
adiw,     Add immediate to word. ( Rw K -- )
Rw = {XH:XL,YH:YL,ZH:ZL}

sub,      Subtract without carry. ( Rd Rr -- )
subi,     Subtract immediate. ( Rd K -- )
sbc,      Subtract with carry. ( Rd Rr -- )
sbci,     Subtract immediate with carry. ( Rd K -- )
sbiw,     Subtract immediate from word. ( Rw K -- )
Rw = {XH:XL,YH:YL,ZH:ZL}

and,      Logical AND. ( Rd Rr -- )
andi,     Logical AND with immediate. ( Rd K -- )
or,       Logical OR. ( Rd Rr -- )
ori,      Logical OR with immediate. ( Rd K -- )
eor,      Exclusive OR. ( Rd Rr -- )
com,      One's complement. ( Rd -- )
neg,      Two's complement. ( Rd -- )
sbr,      Set bit(s) in register. ( Rd K -- )
cbr,      Clear bit(s) in register. ( Rd K -- )
inc,      Increment. ( Rd -- )
dec,      Decrement. ( Rd -- )
tst,      Test for zero or minus. ( Rd -- )
clr,      Clear register. ( Rd -- )
ser,      Set register. ( Rd -- )

mul,      Multiply unsigned. ( Rd Rr -- )
muls,     Multiply signed. ( Rd Rr -- )
mulsu     Multiply signed with unsigned. ( Rd Rr -- )
fmul,     Fractional multiply unsigned. ( Rd Rr -- )
fmuls,    Fractional multiply signed. ( Rd Rr -- )
fmulsu,   Fractional multiply signed with unsigned. ( Rd Rr -- )
```

Branch instructions

rjmp, Relative jump. (**k** --)
ijmp, Indirect jump to (Z). (--)
eijmp, Extended indirect jump to (Z). (--)
jmp, Jump. (**k16 k6** --)
 k6 is zero for a 16-bit address.
rcall, Relative call subroutine. (**k** --)
icall, Indirect call to (Z). (--)
eicall, Extended indirect call to (Z). (--)
call, Call subroutine. (**k16 k6** --)
 k6 is zero for a 16-bit address.
ret, Subroutine return. (--)
reti, Interrupt return. (--)
cpse, Compare, skip if equal. (**Rd Rr** --)
cp, Compare. (**Rd Rr** --)
cpc, Compare with carry. (**Rd Rr** --)
cpi, Compare with immediate. (**Rd K** --)
sbrc, Skip if bit in register cleared. (**Rr b** --)
sbrs, Skip if bit in register set. (**Rr b** --)
sbic, Skip if bit in I/O register cleared. (**A b** --)
sbis, Skip if bit in I/O register set. (**A b** --)

Data transfer instructions

mov, Copy register. (**Rd Rr** --)
movw, Copy register pair. (**Rd Rr** --)
ldi, Load immediate. (**Rd K** --)
lds, Load direct from data space. (**Rd K** --)
ld, Load indirect. (**Rd Rr** --)
 Rr = {**X,X+,-X,Y,Y+,-Y,Z,Z+,-Z**}
ldd, Load indirect with displacement. (**Rd Rr q** --)
 Rr = {**Y,Z**}
sts, Store direct to data space. (**k Rr** --)
st, Store indirect. (**Rr Rd** --)
 Rd = {**X,X+,-X,Y,Y+,-Y,Z,Z+,-Z**}
std, Store indirect with displacement. (**Rr Rd q** --)
 Rd=**{Y,Z}**
in, In from I/O location. (**Rd A** --)
out, Out to I/O location. (**Rr A** --)
push, Push register on stack. (**Rr** --)
pop, Pop register from stack. (**Rd** --)

Bit and bit-test instructions

lsl, Logical shift left. (**Rd** --)
lsr, Logical shift right. (**Rd** --)
rol, Rotate left through carry. (**Rd** --)
ror, Rotate right through carry. (**Rd** --)
asr, Arithmetic shift right. (**Rd** --)
swap, Swap nibbles. (**Rd** --)
bset, Flag set. (**s** --)
bclr, Flag clear. (**s** --)
sbi, Set bit in I/O register. (**A b** --)
cbi, Clear bit in I/O register. (**Rd b** --)
bst, Bit store from register to T. (**Rr b** --)
bld, Bit load from T to register. (**Rd b** --)

sec, Set carry. (--)
clc, Clear carry. (--)
sen, Set negative flag. (--)
cln, Clear negative flag. (--)
sez, Set zero flag. (--)
clz, Clear zero flag. (--)
sei, Global interrupt enable. (--)
cli, Global interrupt disable. (--)
ses, Set signed test flag. (--)
cls, Clear signed test flag. (--)
sev, Set two's complement overflow. (--)
clv, Clear two-s complement overflow. (--)
set, Set T in SREG. (--)
clt, Clear T in SREG. (--)
seh, Set half carry flag in SREG. (--)
clh, Clear half carry flag in SREG. (--)

MCU control instructions

break, Break. (--)
nop, No operation. (--)
sleep, Sleep. (--)
wdr, Watchdog reset. (--)

Synchronous serial communication

I²C communications as master

The following words are available as a common set of words for PIC18FXXK22, PIC24FV32KX30X and ATmega328P microcontrollers. Load them from a file with a name like **i2c-base-XXXX.txt** where **XXXX** is the specific microcontroller.

i2c.init Initializes I²C master mode, 100 kHz clock. (--)
i2c.close Shut down the peripheral module. (--)
i2c.ping? Leaves **true** if the addressed slave device acknowledges. (**7-bit-addr** -- **f**)
i2c.addr.write Address slave device for writing. Leave **true** if the slave acknowledged. (**7-bit-addr** -- **f**)
i2c.c! Send byte and leave **ack** bit. (**c** -- **ack**) Note that the **ack** bit will be high if the slave device did *not* acknowledge.
i2c-addr-read Address slave device for reading. Leave **true** if slave acknowledged. (**7-bit-addr** -- **f**)
i2c.c@.ack Fetch a byte and **ack** for another. (-- **c**)
i2c.c@.nack Fetch one last byte. (-- **c**) (--)

Low level words.
i2c.idle? Leave **true** if the I²C bus is idle. (-- **f**)
i2c.start Send start condition. (--)
i2c.rsen Send restart condition. (--)
i2c.stop Send stop condition. (--)
i2c.wait Poll the I²C hardware until the operation has finished. (--)
i2c.bus.reset Clock through bits so that slave devices are sure to release the bus. (--)

Alternate set I²C words for PIC18

Load these words from **i2c_base.txt** for a PIC18 microcontroller. They make use of the structured assembler for the PIC18.

i2cinit Initializes I²C master mode, 100 kHz 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@ak Read one byte and continue. (-- **c**)
i2c@nak Read one last byte from the I²C bus. (-- **c**)
i2c-addr1 Write 8-bit address to slave. (**addr slave-addr** --)
i2c-addr2 Write 16-bit address to slave (**addr slave-addr** --)
Lower-level words.
ssen Assert start condition. (--)
srssen Assert repeated start condition. (--)
spen Generate a stop condition. (--)
srcen Set receive enable. (--)
snoack Send not-acknowledge. (--)
sack Send acknowledge bit. (--)
sspbuf! Write byte to **SSPBUF** and wait for transmission. (**c** --)

SPI communications as master

The following words are available as a common set of words for PIC18FXXK22, PIC24FV32KX30X and ATmega328P microcontrollers. Load them from a file with a name like **spiN-base-XXXX.txt** where **XXXX** is the specific microcontroller and **N** identifies the particular SPI module. Because SPI devices are so varied in their specification, you likely have to adjust the register settings in **spi.init** to suit your particular device.

spi.init Initializes SPI master mode, 1 MHz clock. (--)
spi.close Shut down the peripheral module. (--)
spi.wait Poll the SPI peripheral until the operation has finished. (--)
spi.cexch Send byte **c1**, leave incoming byte **c2** on stack. (**c1** -- **c2**)
spi.csend Send byte **c**. (**c** --)
spi.select Select the external device. (--)
spi.deselect Deselect the external device. (--)

This guide assembled by Peter Jacobs, School of Mechanical Engineering, The University of Queensland, February-2016 as Report 2016/02. Update 2017-03-28.

It is a remix of material from the following sources:
FlashForth v5.0 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.
L Brodie *Starting Forth* 2nd Ed., 1987 Prentice-Hall Software Series.
Robert B. Reese *Microprocessors from Assembly Language to C Using the PIC18Fxx2* Da Vinci Engineering Press, 2005.
Microchip *16-bit MCU and DSC Programmers Reference Manual* Document DS70157F, 2011.
Atmel *8-bit AVR Insturction Set* Document 08561-AVR-07/10.