Data and the stack

The data stack (S:) is directly accessible and has 32 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 holding numerical values. Functions get their arguments from the stack and leave their results there as well. Everything is interpreted as a word or a number.

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.

Notation

n, n1, n2, n3 Single-cell integers (16-bit).
un, u1, u2 Unsigned integers (16-bit).
x, x1, x2, x3 Single-cell item (16-bit).
c Character value (8-bit).
d Double-cell signed and unsigned (32-bit).
tu Triple-cell signed and unsigned (48-bit).
ququad Quad-cell signed and unsigned (64-bit).
f Boolean flag: 0 is false, 1 is true.
flt1flt3 Floating-point value (32-bit).

Numbers and values

2 Leave integer two onto the stack. ( -- 2 )
#255 Leave decimal 255 onto the stack. ( -- 255 )
%! Leave integer three onto the stack. ( -- 3 )
$10 Leave integer sixteen onto the stack. ( -- 16 )
23 Leave double number on the stack. ( -- 23 )
decimal Set number format to base 10. ( -- )
hex Set number format to hexadecimal. ( -- )
s+d Sign extend single to double number. ( n -- d )
sign Invert sign of number. ( n -- n )
s+b Set number format to binary. ( -- )

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 )
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. ( x u -- x u x u )

Arithmetic with single-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 )
a+ Add single cell to double number. ( d1 n -- d1+n )
a= Signed 16*16 to 32-bit multiply. ( n n -- d )
d2* Multiply by 2. ( d -- d )
d2/ Divide by 2. ( d -- d )
unmod Unsigned division. ( ud u1 -- u.ud u.quot )
udmod Unsigned division. ( ud u1 -- u.ud u.quot )
udmax Unsigned maximum. ( u1 u2 -- u.u )

Arithmetic with double-cell numbers

For PIC18, these words require core.txt, math.txt and qmath.txt.
q+ Add a quad to a quad. ( q1 q2 -- q3 )
qm+ Add double to a quad. ( q1 d -- q2 )
qu+ Add single cell to a quad. ( q1 n -- q2 )
m+ Add single cell to double number. ( d1 n -- d1+n )

qmath.h ( d -- q )
provided that the value is not too large to fit in a
drop 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.

For PIC18, these words require core.txt, math.txt and qmath.txt.
q+ Add a quad to a quad. ( q1 q2 -- q3 )
qm+ Add double to a quad. ( q1 d -- q2 )
qu+ Add single cell to a quad. ( q1 n -- q2 )
m+ Add single cell to double number. ( d1 n -- d1+n )

qmath.h ( d -- q )
provided that the value is not too large to fit in a
drop 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.

For PIC18, these words require core.txt, math.txt and qmath.txt.
q+ Add a quad to a quad. ( q1 q2 -- q3 )
qm+ Add double to a quad. ( q1 d -- q2 )
qu+ Add single cell to a quad. ( q1 n -- q2 )
m+ Add single cell to double number. ( d1 n -- d1+n )

qmath.h ( d -- q )
provided that the value is not too large to fit in a
drop 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.

For PIC18, these words require core.txt, math.txt and qmath.txt.
q+ Add a quad to a quad. ( q1 q2 -- q3 )
qm+ Add double to a quad. ( q1 d -- q2 )
qu+ Add single cell to a quad. ( q1 n -- q2 )
m+ Add single cell to double number. ( d1 n -- d1+n )

qmath.h ( d -- q )
provided that the value is not too large to fit in a
drop 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.

For PIC18, these words require core.txt, math.txt and qmath.txt.
q+ Add a quad to a quad. ( q1 q2 -- q3 )
qm+ Add double to a quad. ( q1 d -- q2 )
qu+ Add single cell to a quad. ( q1 n -- q2 )
m+ Add single cell to double number. ( d1 n -- d1+n )

qmath.h ( d -- q )
provided that the value is not too large to fit in a
drop 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.

For PIC18, these words require core.txt, math.txt and qmath.txt.
q+ Add a quad to a quad. ( q1 q2 -- q3 )
qm+ Add double to a quad. ( q1 d -- q2 )
qu+ Add single cell to a quad. ( q1 n -- q2 )
m+ Add single cell to double number. ( d1 n -- d1+n )

qmath.h ( d -- q )
provided that the value is not too large to fit in a
drop 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.

For PIC18, these words require core.txt, math.txt and qmath.txt.
q+ Add a quad to a quad. ( q1 q2 -- q3 )
qm+ Add double to a quad. ( q1 d -- q2 )
qu+ Add single cell to a quad. ( q1 n -- q2 )
m+ Add single cell to double number. ( d1 n -- d1+n )

qmath.h ( d -- q )
provided that the value is not too large to fit in a
drop 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.

For PIC18, these words require core.txt, math.txt and qmath.txt.
q+ Add a quad to a quad. ( q1 q2 -- q3 )
qm+ Add double to a quad. ( q1 d -- q2 )
qu+ Add single cell to a quad. ( q1 n -- q2 )
m+ Add single cell to double number. ( d1 n -- d1+n )

qmath.h ( d -- q )
provided that the value is not too large to fit in a
drop 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.

For PIC18, these words require core.txt, math.txt and qmath.txt.
q+ Add a quad to a quad. ( q1 q2 -- q3 )
qm+ Add double to a quad. ( q1 d -- q2 )
qu+ Add single cell to a quad. ( q1 n -- q2 )
m+ Add single cell to double number. ( d1 n -- d1+n )

qmath.h ( d -- q )
provided that the value is not too large to fit in a
drop in the cell above the least significant bits, you can
just drop the top cell to recover the single number,
Relational

* Leave true if \( x_1 \times x_2 \) are equal. ( \( x_1 \times x_2 \rightarrow f \) )

<> Leave true if \( x_1 \times x_2 \) are not equal. ( \( x_1 \times x_2 \rightarrow f \) )

< Leave true if \( n_1 \) less than \( n_2 \). ( \( n_1 \rightarrow n_2 \rightarrow f \) )

\( \geq \) Leave true if \( n_1 \) greater than \( n_2 \). ( \( n_1 \rightarrow n_2 \rightarrow f \) )

0 Leave true if \( n \) is zero. ( \( n \rightarrow f \) )

\( \geq \) Leave true if \( n \) is negative. ( \( n \rightarrow f \) )

within Leave true if \( x_1 \leq x < x_2 \). ( \( x_1 \rightarrow x \rightarrow f \) )

u< Leave true if \( u_1 < u_2 \). ( \( u_1 \rightarrow u_2 \rightarrow f \) )

u> Leave true if \( u_1 > u_2 \). ( \( u_1 \rightarrow u_2 \rightarrow f \) )

d< Leave true if \( d_1 < d_2 \). ( \( d_1 \rightarrow d_2 \rightarrow f \) )

d> Leave true if \( d_1 > d_2 \). ( \( d_1 \rightarrow d_2 \rightarrow f \) )

d2= Leave true if \( d_1 \rightarrow d_2 \) are equal. ( \( d_1 \rightarrow d_2 \rightarrow f \) )

Accessing Memory

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>Store ( x \rightarrow \text{addr} )</td>
</tr>
<tr>
<td>@</td>
<td>Fetch ( \text{addr} \rightarrow x )</td>
</tr>
<tr>
<td>&amp;</td>
<td>Fetch cell and increment address by cell size. ( \text{addr} \rightarrow \text{addr} + x )</td>
</tr>
</tbody>
</table>
| | Leave high limit of the current data space. ( \( -- u \) )

AVR8 Memory map

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

- **$0000 - (\text{RAMSIZE}-1)** SRAM
- **$0000 - (\text{RAMSIZE}+\text{EPPROMSIZE}-1)** EPPROM
- **$ffff-$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 EPPROM. ( \( -- \) )
- **flash** Set address context to Flash. ( \( -- \) )

Bitwise

- **invert** Ones complement. ( \( x \rightarrow x \) )
- **diverter** Invert double number. ( \( du \rightarrow du \) )
- **and** Bitwise and. ( \( x_1 \times x_2 \rightarrow x \) )
- **or** Bitwise or. ( \( x_1 \times x_2 \rightarrow x \) )
- **xor** Bitwise exclusive-or. ( \( x \rightarrow x \) )
- **lshift** Left shift by u bits. ( \( x_1 \rightarrow x_2 \) )
- **rshift** Right shift by u bits. ( \( x_1 \rightarrow x_2 \) )

Dictionary

**Dictionary management**

- **maker -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. ( \( \rightarrow u \) )
- **leaves** 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 words in dictionary. ( \( -- \) )

Defining constants and variables

- **constant name** Define new constant. ( \( n \rightarrow \) )
- **2constant name** Define double constant. ( \( x \times x \rightarrow \) )
- **name** Leave value on stack. ( \( -- n \) )
- **variable name** Define a variable in the current data section. ( \( -- \) )
- **ram** Set address context to SRAM. ( \( -- \) )
- **eeprom** Set address context to EPPROM. ( \( -- \) )
- **flash** Set address context to Flash. ( \( -- \) )

Examples

- **ram** Set SRAM context for variables and values. Be careful not to accidentally define variables in EPPROM or Flash memory. That memory wears quickly with multiple writes.
- **eeprom** Define variable in EPPROM.
- **value xx** Define value in SRAM.
- **variable yy** Define variable in SRAM.
- **6 yy !** Store 6 in variable yy.
- **eprom 5 value zz ram** Define value in EPPROM.
- **xx yy zz portb yy @** Leaves 3 $1725$ $f180$ $6$

Memory

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

PIC18 Memory map

The address ranges are:

- **$0000 - $0eff** Flash
- **$e000 - $efff** EPPROM
- **$f000 - $f5ff** SRAM, general use
- **$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 $\text{p18f-main}.\text{cfg}$ for the UAR1 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 EPPROM will have its 64kB address space divided into:

- **$0000 - $07ff** SRAM, special function registers
- **$0800 - ($0800+\text{RAMSIZE}-1)** SRAM, general use
- **($0800+\text{RAMSIZE}) - $7fff** Flash
- **$fc00 - $ffff** EPPROM

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.
Warming restart clears SRAM data.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx yy zz portb yy @</td>
<td>Leaves 0 f172 5 f8b1 0</td>
</tr>
<tr>
<td>4 to xx</td>
<td>Sets new value.</td>
</tr>
<tr>
<td>xx yy zz portb yy @</td>
<td>Prints the number of bytes free.</td>
</tr>
<tr>
<td>hi here - u.</td>
<td>PortB latch for the PIC18.</td>
</tr>
<tr>
<td>$ff8a constant latb</td>
<td>PortB direction-control register.</td>
</tr>
<tr>
<td>$ff93 constant trisb</td>
<td>Sets RB1 as output.</td>
</tr>
<tr>
<td>00000100 trisb mclr</td>
<td>Defines a word to set RB1 high.</td>
</tr>
<tr>
<td>latb 1 bitl: pbl-high</td>
<td>Sets RB1 high.</td>
</tr>
</tbody>
</table>

### Define compound data objects

#### mk-byte-array

- **Syntax**: `mk-byte-array [cell]`  
- **Description**: Creates an array object.
- **Example**: `create my-byte-array 10` creates an array of 10 bytes.

#### mk-cell-array

- **Syntax**: `mk-cell-array [cell]`  
- **Description**: Creates an array object.
- **Example**: `create my-cell-array 5` creates an array of 5 cells.

#### Memory operations

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

- **cmove**: Move u bytes from address-1 to address-2.
- **fill**: Fill u bytes with c starting at address.
- **erase**: Fill u bytes with 0 starting at address.
- **cells**: Convert cells to cell units.
- **chars**: Convert chars to address units.
- **aligned**: Align address to a cell boundary.

### Array examples

- **ram create my-array 20 allot**  
  "..of creating an array,"  
  "filling it with 8s,"  
  "displaying its content."  

- **create my-cell-array N=100, 340, 5**  
  Initialized cell array.

- **my-byte-array 18 c, 21 c, 255 c**  
  Should leave 5.  
  Initialized byte array.

- **my-byte-array 2 chars + c0**  
  Should leave 255.  
  Initialized byte array.

### Predefined constants

- **cell**: Size of one cell in characters.  
  Defined word (n -- )  
  ...to make byte array objects

- **true**: Boolean true value.  
  Defined word (n -- )  
  ...as shown in FF user's guide.

- **false**: Boolean false value.  
  Defined word (n -- )  
  ...and another.

- **ti#**: Size of the terminal input buffer.  
  Defined word (n -- )  
  ...to make cell array objects.

- **alpha**: Convert a real executable address to a Flash virtual address.  
  Defined word (n -- )  
  ...as shown in FF user's guide.

- **tp#**: Convert a Flash virtual address to a real executable address.  
  Defined word (n -- )  
  ...to make byte array objects.
The Compiler

Defining functions

: Begin colon definition. ( -- )
; End colon definition. ( -- )
[ Enter interpreter state. ( -- )
] Enter compilation state. ( -- )
state Compilation state. ( -- )
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]
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.
  [ $f8a 0 a, bsf, ] ;

The following functions call out to the Microchip math library.

Multiply by 2.
Leave maximum.

Arccosine of flt, radians. ( flt -- acos(flt) )
Arcinarctangent of flt, radians. ( flt -- atan(flt) )
Arccosine of flt, radians. ( flt -- atan(flt) )
Arccosine of flt/flt2, radians. ( flt1 flt2 -- atan(flt1/flt2) )
Square-root. ( flt -- sqrt(flt) )
Exponential. ( flt -- exp(flt) )
Natural logarithm. ( flt -- loge(flt) )
Natural logarithm, base 10. ( flt -- log10(flt) )
Hyperbolic cosine. ( flt -- cosh(flt) )
Hyperbolic sine. ( flt -- sinh(flt) )
Hyperbolic tangent. ( flt -- tanh(flt) )

Compilation state.

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 next Loop u times. ( u -- )
endif Sets loop counter to zero so that we leave a for loop when next is encountered. ( -- )

From do loop.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
...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 ;
  : sumfor 0 100 for i + next ;
  : print-two 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 r4
  : case 0 of " zero " endof
  : case 1 of " one " endof
  : case 2 of " two " endof
  : default " endof 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, type out last word executed, followed by text xxx. ( f -- )
quit Interpret from keyboard. ( -- )
**Vectorized 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.

- `@sex` Fetch vector from addr and execute. ( ** )

- `define vec-name` Define a deferred execution vector. ( ** )

- `is vec-name` Store execution token in `vec-name`. ( ** )

- `vec-name` Execute the word whose execution token
  is stored in `vec-name`’s data space.

- `int!` Store interrupt vector to table.
  ( at 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.

  PIC24/F: Alternate interrupt vector table in RAM.
  PIC24/FK: Alternate interrupt vector table in Flash.
  PIC24/E: 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, dsPIC30E.

- `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 0 1+ icnt1 !
  ] i
  )

  Set the user interrupt vector.

  : init
  ( Alternatively, compile a word
  [ i ] 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(int)er register. ( ** addr )

- `@p` Fetch the P register to the stack. ( ** )

- `!p>r` Push contents of P to return stack and
  store new address to P. ( ** ) ( R: -- addr )

- `rp` Pop from return stack to P register. ( ** ) ( R: addr -- )

- `p*` Increment P register by one. ( ** )

- `p+` Add 2 to P register. ( ** )

- `pc!` 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 )

- `>p` Convert a character to an ASCII value. ( c -- c )

  Non-numeric characters converted to a dot.

- `char` Char
  ( For example: char A ( -- 65 )

- `[char] char` Compile inline ASCII character. ( -- )

**Strings**

Some of these words come from core.txt.

- `" text"` Compile string into flash. ( ** )

  At run time, leaves address and length.

  ( ** addr u )

- `. text` Compile string to print into flash. ( ** )

**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. ( 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. <# # # ## rot sign #> type

  results in -34 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 `entit`, `key` and `key?`.
Serial communication ports

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tx0</td>
<td>Send a character via UART0 on ATmega. (c -- )</td>
</tr>
<tr>
<td>rx0?</td>
<td>Receive a character from UART0 on ATmega. (c -- )</td>
</tr>
<tr>
<td>u0?</td>
<td>Disable flow control for UART1 interface. (-- )</td>
</tr>
<tr>
<td>u0+</td>
<td>Enable flow control for UART1 interface, default. (--)</td>
</tr>
<tr>
<td>tx1</td>
<td>Send character to UART1. (c -- )</td>
</tr>
<tr>
<td>u1-</td>
<td>Disable flow control for UART2 interface. (--)</td>
</tr>
<tr>
<td>u1+</td>
<td>Enable flow control for UART2 interface, default. (--)</td>
</tr>
<tr>
<td>tx2</td>
<td>Send character to UART2. (c -- )</td>
</tr>
<tr>
<td>rx2?</td>
<td>Receive a character from UART2. (c -- )</td>
</tr>
</tbody>
</table>

Character queues on PIC24-30-33

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cq:</td>
<td>Create character queue. (u -- )</td>
</tr>
<tr>
<td>cq0</td>
<td>Initialize or reset queue. (queue-addr -- )</td>
</tr>
<tr>
<td>&gt;cq</td>
<td>Get character from queue. (queue-addr -- f)</td>
</tr>
<tr>
<td>cq?</td>
<td>Get character from queue. (queue-addr -- )</td>
</tr>
<tr>
<td>u1rxq</td>
<td>Leave UART1 RX queue address. (-- queue-addr)</td>
</tr>
<tr>
<td>u1txq</td>
<td>Leave UART1 TX queue address. (-- queue-addr)</td>
</tr>
<tr>
<td>u2rxq</td>
<td>Leave UART2 RX queue address. (-- queue-addr)</td>
</tr>
<tr>
<td>u2txq</td>
<td>Leave UART2 TX queue address. (-- queue-addr)</td>
</tr>
</tbody>
</table>

Multitasking

Load the words for multitasking from task.txt.

task: Define a new task in flash memory space

(t tasksize rstacksize addsize -- )

Use run xxx allot to leave space for the PAF of the previously defined task.
The OPERATOR task does not use PAF.

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 -- )

task: Makes a task run by inserting it after operator
task in the round-robin linked list. (task-addr -- )

May only be executed from the operator task.

task: Remove a task from the task list. (task-addr -- )

May only be executed from the operator task.
	
task: End all tasks except the operator task. (--) 

Removes all tasks from the task list.
	
tasks: List all running tasks. (--) 

May only be executed from the operator task.

task: 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. (--) 

Load is percentage of time that the CPU is busy.

load*: Enable the load LED on AVR8. (--) 

do*: Enable 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 -- ) 

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–ffl milliseconds. (u -- ) 

Multitasking

Load the words for multitasking from task.txt.

task: Define a new task in flash memory space

(t tasksize rstacksize addsize -- )

Use run xxx allot to leave space for the PAF of the previously defined task.
The OPERATOR task does not use PAF.

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
task in the round-robin linked list. (task-addr -- )

May only be executed from the operator task.

task: 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.

tasks: List all running tasks. (--) 

May only be executed from the operator task.

task: 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. (--) 

Load is percentage of time that the CPU is busy.

load*: Enable the load LED on AVR8. (--) 

do*: Enable 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 condition is true. (--) 

Assembler words for PIC18

In the stack-effect notation 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 -- ) 

cn, 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

u, Destination WREG (0 -- ) 

f, Destination file (1 -- ) 

a, Access bank (0 -- ) 

b, Use bank-select register (1 -- ) 

Byte-oriented file register operations

adduf, Add WREG and f. (f d a -- ) 

adducf, Add WREG and carry bit to f. (f d a -- ) 

anduf, AND WREG with f. (f d a -- ) 

clrf, Clear f. (f a -- ) 

conf, Complement f. (f d a -- ) 

cpfsseq, Compare f with WREG, skip if equal. (f a -- ) 

cpfsqg, Compare f with WREG, skip if greater than. (f a -- ) 

cpfslt, Compare f with WREG, skip if less than. (f a -- ) 

defcf, Decrement f. (f d a -- ) 

defcsfz, Decrement f, skip if zero. (f d a -- ) 

cfsnzd, Decrement f, skip if not zero. (f d a -- ) 

incf, Increment f. (f d a -- ) 

incsfz, Increment f, skip if zero. (f d a -- ) 

infzend, Increment f, skip if not zero. (f d a -- ) 

iorwf, Exclusive OR WREG with f. (f d a -- ) 

movf, Move f. (f d a -- ) 

movff, Move f. (f d 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 -- ) 

testf, Test f. (f d a -- ) 

subfub, Subtract f1 from WREG, with borrow. (f d a -- ) 

subfui, Subtract WREG from f1. (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 -- )
btsfc, Bit test f, skip if clear. ( f b a -- )
bttfs, Bit test f, skip if set. ( f b a -- )
bttg, 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. ( -- )
ioinw, Inclusive OR literal with WREG. ( k -- )
ifsr, Move literal to FSR. ( k f -- )
movlw, Move literal to BSR. ( k -- )
movlw, Move literal to WREG. ( k -- )
mullw, Multiply literal with WREG. ( k -- )
sublw, Subtract literal from WREG. ( 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. ( -- )
tbltw+, Table write. ( -- )
tbltw++, Table write with post-increment. ( -- )
tbltw+*, 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. ( -- )
callr, Relative call. ( rel-addr -- )
retief, Return from interrupt enable. ( -- )
retlw, Return with literal in WREG. ( -- )
retlw, Return from interrupt enable. ( -- )

Register constants

Z ( -- 0 )
Z+ ( -- 1 )
-Z ( -- 2 )
Y ( -- 8 )
Y+ ( -- 9 )
-Y ( -- 10 )
X ( -- 11 )
X* ( -- 12 )
-X ( -- 13 )
XH:XL ( -- 01 )
YH:YL ( -- 02 )
ZH:ZL ( -- 03 )

Conditions for structured flow control

z, test zero ( -- cc )
zx, test not zero ( -- cc )
not, invert condition ( cc -- not-cc )

Low-level flow control operations

bra, Branch unconditionally. ( rel-addr -- )
call, Call subroutine. ( rel-addr -- )
return, Return from subroutine. ( -- )
retief, Return from interrupt enable. ( -- )

Bit-oriented operations

bcir, Bit clear. ( bit ram-addr -- )
bset, Bit set. ( bit ram-addr -- )
btest, Bit test to z. ( bit ram-addr -- )
btest, Bit test, skip if clear. ( bit ram-addr -- )
btest, 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 )
ls, interrupt enabled ( -- cc )
lo, lower ( -- cc )
lit, less than ( -- cc )
mis, negative ( -- cc )

t, T flag set ( -- cc )
vs, no overflow ( -- cc )
not, invert condition ( cc -- not-cc )

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.

Arithmetic and logic instructions

add, Add without carry. ( Rd Rr -- )
adc, Add with carry. ( Rd Rr -- )
adiw, Add immediate to word. ( Rw K -- )
adiw, Add immediate to word. ( Rw K -- )
sub, Subtract without carry. ( Rd Rr -- )
subi, Subtract immediate. ( Rd K -- )
sbc, Subtract with carry. ( Rd Rr -- )
sbc, Subtract immediate with carry. ( Rd K -- )
sbiw, Subtract immediate from word. ( Rw K -- )
sub, Subtract without carry. ( Rd Rr -- )
subi, Subtract immediate. ( Rd K -- )
sbc, Subtract with carry. ( Rd Rr -- )
sbc, Subtract immediate with carry. ( Rd K -- )
sbiw, Subtract immediate from word. ( Rw K -- )
sub, Subtract without carry. ( Rd Rr -- )
subi, Subtract immediate. ( Rd K -- )
sbc, Subtract with carry. ( Rd Rr -- )
sbc, Subtract immediate with carry. ( Rd K -- )
sbiw, Subtract immediate from word. ( Rw K -- )
sub, Subtract without carry. ( Rd Rr -- )
subi, Subtract immediate. ( Rd K -- )
sbc, Subtract with carry. ( Rd Rr -- )
sbc, Subtract immediate with carry. ( Rd K -- )
sbiw, Subtract immediate from word. ( Rw K -- )
sub, Subtract without carry. ( Rd Rr -- )
subi, Subtract immediate. ( Rd K -- )
sbc, Subtract with carry. ( Rd Rr -- )
sbc, Subtract immediate with carry. ( Rd K -- )
sbiw, Subtract immediate from word. ( Rw K -- )
Branch instructions

rjmp, Relative jump. ( k -- )
ijmp, Indirect jump to (Z). ( -- )
eijmp, Extended indirect jump to (Z). ( -- )
jmp, Jump. ( x16 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. ( x16 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 -- )
sbrc, 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 -- )
mov, 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,Y,X+,-X,Y,Y+,-Y,Z,Z+,-Z)
ldd, Load indirect with displacement. ( Rd Rr q -- )
   Rr = (Y,Z)

st, Store direct to data space. ( k Rr -- )

std, Store indirect. ( Rr Rd -- )
   Rd = (X,Y,X+,-X,Y,Y+,-Y,Z,Z+,-Z)

set, Store indirect with displacement. ( Rr Rd q -- )

mcu.control instructions

break, Break. ( -- )
nop, No operation. ( -- )
sleep, Sleep. ( -- )
sleep, Sleep. ( -- )
wdt, 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 XXX is the specific microcontroller.

i2c.init Initializes I²C master mode, 100kHz clock. ( -- )
i2c.close Shuts down the peripheral module. ( -- )
i2c.ping? Leaves true if the addressed slave device acknowledges. ( 7-bit-addr --  f )
i2c.ack Address slave device for writing. ( -- )
leave true if the slave acknowledged. ( 7-bit-addr --  f )
ic1 Send byte and leave ack bit. ( c -- ack )
leave true if the ack bit will be high
   if the slave device did not acknowledge.
ic2 Send byte and leave slave device for reading. ( -- )
leave true if slave acknowledged. ( 7-bit-addr --  f )
ic2c.c0.ack Fetch a byte and ack for another. ( -- c )
ic2c.c0.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.ren 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, 100kHz clock. ( -- )
i2cws Wake slave. Bit 0 is R/W bit. ( slave-address -- )
i2c! Write one byte to I²C bus and wait for ACK. ( c -- )
i2c0ak Read one byte and continue. ( -- c )
i2c0nak Read one byte from the I²C bus. ( -- c )
i2c-addr1 Write 8-bit address to slave. ( addr slave-address -- )
i2c-addr2 Write 16-bit address to slave. ( addr slave-address -- )

Lower-level words:

ssen Assert start condition. ( -- )
sren Assert repeated start condition. ( -- )
sren Generate a stop condition. ( -- )
srcen Set receive enable. ( -- )
snack Send not-acknowledge. ( -- )
sack Send acknowledge bit. ( -- )
ssbuf! Write byte to SSBUF 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
spi-base-XXXX.txt where XXX 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
ing settings in spi.init to suit your particular device.

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

This guide assembled by Peter Jacobs, School of Mechanical Engineering,
The University of Queensland, February-2016 as Report 2016/02.
It is a remix of material from the following sources:
FlashForth v5.0 source code and word list by Mikael Nordin
http://flashforth.sourceforge.net/
EK Conklin and ED Rather Forth Programmer’s Handbook 3rd Ed.
2007 FORTH, Inc.
Robert B. Reese Microprocessors from Assembly Language to C Using the
PIC18F46x Cross Tool Engineers Press, 2005.
Microchip 16-bit MCU and DSC Programmers Reference Manual