Hitchhiker’s Guide to FlashForth on 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.

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 address stack (R:) that can be used for temporary storage.

Notation

n, n1, n2, n3 Single-cell integers (16-bit).
U, u, u1, u2 Unsigned integers (16-bit).
x, x1, x2, x3 Single-cell integer (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.
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 one onto the stack. ( -- 1 )
23 Leave double number on the stack. ( -- 23 )

#2#3 Set number format to 64-bit/16-bit.
#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.

Displaying data

. Display a number. ( n -- )
um Display u unsigned. ( u -- )

u. Display u with field width n, 0 < n < 256. ( u n -- )
d. Display double number. ( d -- )
ud. Display unsigned double number. ( ud -- )

Stack manipulation

dup Duplicate top item. ( x -- x x )
?dup Duplicate top item if nonzero. ( x -- x x )
swap Swap top two items. ( x1 x2 -- x2 x1 )
over Copy second item to top. ( x1 x2 x3 -- x2 x1 x3 )
2swap Swap two double items. ( d1 d2 -- d2 d1 )
snip Remove x1 from the stack. ( x1 x2 x3 -- x2 x3 )
rot Insert x2 below x1 in the stack. ( x1 x2 x3 -- x2 x1 x3 )
pick Duplicate the u-th item on top.
( x u ... x0 x u ... x0 x )
dup2 Duplicate double top item. ( d -- d d )
2swap Swap two double items. ( d1 d2 -- d2 d1 )

Arithmetic with single-cell numbers

Some of these words require core.txt, math.txt and qmath.txt.
d+d Add double numbers. ( d1 d2 -- d1+d2 )
d-d Subtract double numbers. ( d1 d2 -- d1-d2 )
m+ Add single cell to double number. ( n1 n2 -- n )
m- Subtract single cell from double. ( n1 n2 -- n )
d* Multiply by 2. ( d -- d )
d/ Divide by 2. ( d -- d )

Arithmetic with double-cell numbers

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

Arithmetic with triple- and quad-numbers

These words require core.txt, math.txt and qmath.txt.

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 x1 less than x2. ( n1 n2 -- f )
> Leave true if x1 greater than n2. ( n1 n2 -- f )
0= Leave true if x is zero. ( n -- f )

Inverts logical value.

0c Leave true if n is negative. ( n -- f )
within Leave true if x <= x <= x. ( x x1 xh -- f )
uc Leave true if x < x2. ( x1 x2 -- f )
" Leave true if x >= x2. ( x1 x2 -- f )
d0= Leave true if x is zero or if x is negative. ( d -- f )
d< Leave true if x is negative. ( d -- f )
d< Leave true if x < x2. ( d1 d2 -- f )
d> Leave true if x1 > x2. ( d1 d2 -- f )

Arithmetic with single-cell numbers

--. Display stack content (nondestructively).
.st Emit status string for base, current data section, and display the stack contents. ( -- )
dump Display memory from address, for u bytes. ( addr u -- )

Operators

. Display stack content (nondestructively).
.st Emit status string for base, current data section, and display the stack contents. ( -- )
dump Display memory from address, for u bytes. ( addr u -- )

Arithmetic with single-cell numbers

Some of these words require core.txt, math.txt and qmath.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 -- n1%n2 ) remainder
/mod Divide. ( n1 n2 -- n1%n2 quotient
u/ Unsigned 16/16 to 16-bit division. ( u1 u2 -- u1/u2 )

Arithmetic with double-cell numbers

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

d+d Add double numbers. ( d1 d2 -- d1+d2 )
d-d Subtract double numbers. ( d1 d2 -- d1-d2 )
m+ Add single cell to double number. ( n1 n2 -- n )
m- Subtract single cell from double. ( n1 n2 -- n )
d* Multiply by 2. ( d -- d )
d/ Divide by 2. ( d -- d )

Arithmetic with triple- and quad-numbers

These words require core.txt, math.txt and qmath.txt.

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 x1 less than x2. ( n1 n2 -- f )
> Leave true if x1 greater than n2. ( n1 n2 -- f )
0= Leave true if x is zero. ( n -- f )

Inverts logical value.

0c Leave true if n is negative. ( n -- f )
within Leave true if x <= x <= x. ( x x1 xh -- f )
uc Leave true if x < x2. ( x1 x2 -- f )
" Leave true if x >= x2. ( x1 x2 -- f )
d0= Leave true if x is zero or if x is negative. ( d -- f )
d< Leave true if x is negative. ( d -- f )
d< Leave true if x < x2. ( d1 d2 -- f )
d> Leave true if x1 > x2. ( d1 d2 -- f )

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

Interaction with the operator
Interaction with the user is via the serial port, typically UART1.
Settings are 38400 baud, 8N1, using Xon/Xoff handshaking.

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) - $fff: Flash
- $fc00 - $ffff: EEPROM

The high memory mark for the Flash context will depend on the device. 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:

- $0000 - (RAMSIZE-1): RAM
- RAMSIZE - (RAMSIZE+EPPROMSIZE-1): EEPROM
- ($fff-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.
lock  Disable writes to Flash, EEPROM.
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 )

Other Hardware
clock  Clear the WatchDog counter. ( -- )
ei  Enable Interrupts. ( -- )
di  Disable Interrupts. ( -- )
ms  Pause for +n milliseconds. ( +n -- )
ticks  System ticks, 0–fff milliseconds. ( -- u )

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 - $07ff: Flash
- $8000 - $07ff: EEPROM
- $0800 - $fff: SRAM, general use
- $f000 - $7fff: 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 FS 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.

Converting between cells, chars
chars  Convert cells to address units. ( u -- u )
cells  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 )

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
3 value $x
variable $y
6 $y !
eeprom 5 value $z
xx $yy $zz $portb $yy $y
warm
xx $yy $zz $portb $yy $y
hi here - $u
xx $yy $zz $portb $yy $y
$xx $yy $zz $portb $yy $y
$zz
$ff81 constant portb
$z
$xx $yy $zz $portb $yy $y
print $number of bytes free.
print $number of bytes free.

PortB latch for the PIC18.
PortB direction-control register.
Sets RB1 as output.
Defines a word to set RB1 high.
Sets RB1 high.

For manipulating bits in the ATmega IO-space, the following come from bio.txt
bio1: name Define a word to set a bit. ( io-addr bit -- )
bio0: name Define a word to clear a bit. ( io-addr bit -- )
bio?: name Define a word to test a bit. ( io-addr bit -- )

When executed, name leaves a flag. ( -- f )

Constants and Variables
constant name  Define new constant. ( n -- )
2constant name  Define double constant. ( x n -- )
name  Define variable name. ( n -- )
variable varname  Define variable in address context. ( -- )
2variable varname  Define double variable. ( -- )
value varname  Define value. ( n -- )
to varname  Assign new value to varname. ( n -- )
varname  Leave value on stack. ( -- n )

Memory
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.
lock  Disable writes to Flash, EEPROM.
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 )
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 )

Accessing bits in RAM
mset  Set bits in file register with mask c. ( c addr -- )
mclr  Clear bits in file register with mask c. ( c addr -- )
mst  AND file register byte with mask c. ( c addr -- x )

The following come from bit.txt
bit: name  Define a word to set a bit. ( addr bit -- )
bit:0 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 )
Memory operations

Some of these words come from core.txt.
cmove Move u bytes from address1 to address-2. ( addr1 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 -- )

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(inter) register. ( addr -- )
?p Fetch the P register to the stack. ( -- addr )
?>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 Increment P register by one. ( -- )
p2+ Add 2 to P register. ( -- )
pn+ Add n to the p register. ( n -- )
!/p Store address to P(ointer) register. ( -- )

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

Predefined constants

cell Size of one cell in characters. ( -- n )
true Boolean true value. ( -- )
false Boolean false value. ( -- )
bl ASCII space. ( -- )
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 EEPROM value mirrored in SRAM.
prompt Deferred execution vector for the info displayed
by quit. ( -- a-addr )
'emit EMIT vector. Default is TXI. ( -- a-addr )
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 -- )
endwhile  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-out loop index. ( -- n )
leave Leave the do loop immediately. ( -- )

Loop examples

: sumfor 0 100 for r@ + next ; sumfor
leaves 4950
leaves 4950

decimal

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

Function pointers (vectors)

\' name  Search for name and leave its execution token (address). ( -- addr )
[\' ] name  Search for name and compile its execution token. ( -- )
execute  Execute word at address. ( addr -- )
The actual stack effect will depend on the word executed.
@exec  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 n -- )
PIC18: n is dummy vector number (0).
PIC19: n is dummy vector number (0).
PIC30: Alternate interrupt vector table in Flash.
PIC33: Alternate interrupt vector table in RAM.
PIC24: Alternate interrupt vector table in RAM.

ATmega: Interrupt vector table in RAM.

Autostart example

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

Interrupt example

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

Multitasking

Load the words for multitasking from task.txt.

\textbf{task}  Define a new task in flash memory space
\textbf{taskloop}  Use ram xxx allot to leave space for the PAD
\textbf{of the previously defined task.}
The OPERATOR task does not use PAD.

\textbf{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.

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. ( -- )
Remove all tasks from the task list.

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

busy  Updated every 256 milliseconds.

idle  CPU idle mode not allowed. ( -- )
operator  Leave the address of the operator task. ( -- )

uluink  Link to next task. ( -- addr )

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.

alloc  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 -- )
\cf,  Compile x 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 )
\", "xxx"  Append a string at HERE. ( -- )

Array examples

\textbf{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 + @
my-byte-array
\[18 c, 21 c, 255 c\]  Initialised byte array.

create my-byte-array
\[10 mk-byte-array my-bytes\]  Should leave 5. ( -- x )

create my-cell-array
\[100, 340, 5\]  Initialised cell array.

my-cell-array 2 cells + @
my-byte-array
\[18 c, 21 c, 255 c\]  Initialised byte array.

create my-byte-array
\[10 mk-byte-array my-bytes\]  Should leave 255. ( -- c )

Defining word ( n -- )
...to make byte array objects
...as shown in FF user's guide.

my-byte-array
\[18 0 my-bytes \]  Creates an array object.
\c  my-bytes ( n -- addr )
\c  Sets an element ..and another.

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, type output last word executed,
followed by text xxx. ( f -- )
quithelp  Interpreted from keyboard. ( -- )
cold  Make a cold start.
Reset all dictionary pointers.

warm  Make a warm start.
Note that irq vector is cleared.
Use bank-select register
Destination file
Destination WREG
not
invert condition
Test not zero
Test not overflow
Test overflow
Test zero
Test negative
Test not carry
Test carry
Conditions for structured flow control
Assembler words for PIC18
Assembler words for PIC24-30-33
Low-level flow control
Assembler words for AVR8

-Inclusive OR WREG with f. (f d a --)
mov, Move f. (f d a --)
movf, Move fs to fd. (fs fd --)
mulwf, Multiply WREG with f. (f a --)
negf, Negate f. (f a --)
rlcf, Rotate right f, no carry. (f d a --)
rrcf, Rotate right f, through carry. (f d a --)
rcf, Rotate left f, through carry. (f d a --)
rncf, Rotate left f, no carry. (f d a --)
sel, Set f. (f d a --)
sublw, Subtract WREG from literal. (f --)
subwf, Subtract WREG from f. (f d a --)
swapf, Swap cells + ;
subw, Subtract f from WREG, with borrow. (f d a --)
subwf, Subtract WREG from f. (f d a --)
swapf, Swap WREG with f. (f d a --)
tstfz, Test f, skip if zero. (f a --)
xorwf, Exclusive OR WREG with f. (f d a --)

<table>
<thead>
<tr>
<th>Bit-oriented file register operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcf, Bit clear f. (f b a --)</td>
</tr>
<tr>
<td>bcf, Bit clear f. (f b a --)</td>
</tr>
<tr>
<td>btfsc, Bit test f, skip if clear. (f b a --)</td>
</tr>
<tr>
<td>bftss, Bit test f, skip if set. (f b a --)</td>
</tr>
<tr>
<td>btg, Bit toggle f. (f b a --)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Literal operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>addiv, Add literal and WREG. (x k --)</td>
</tr>
<tr>
<td>andlw, AND literal with WREG. (k --)</td>
</tr>
<tr>
<td>iorlw, Inclusive OR literal with WREG. (k --)</td>
</tr>
<tr>
<td>movlw, Move literal to WREG. (k --)</td>
</tr>
</tbody>
</table>

- Bit-oriented operations
- Literal operations

- Assembler words for AVRs
  
  For the ATmega instructions, Rd denotes the destination (and source) register, Rr denotes the source register, 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

Add without carry. ( Rd Rs -- )
and, Logical AND. ( Rd Rs -- )
andi, Logical AND with immediate. ( Rd K -- )
or, Logical OR. ( Rd Rs -- )
orl, Logical OR with immediate. ( Rd K -- )
com, Exclusive OR. ( Rd Rs -- )
com, Two’s complement. ( Rd Rs -- )
neg, One’s complement. ( Rd Rs -- )
sbr, Set bit(s) in register. ( Rd K -- )
cbr, Clear bit(s) in register. ( Rd K -- )
inc, Increment. ( Rd Rs -- )
dec, Decrement. ( Rd Rs -- )
tst, Test for zero or minus. ( Rd Rs -- )
clr, Clear register. ( Rd Rs -- )
sr, Set register. ( Rd Rs -- )
mul, Multiply unsigned. ( Rd Rs -- )
muls, Multiply signed. ( Rd Rs -- )
mulsu, Multiply signed with unsigned. ( Rd Rs -- )
fmul, Fractional multiply unsigned. ( Rd Rs -- )
fmul, Fractional multiply signed. ( Rd Rs -- )

Branch instructions

rjmp, Relative jump. ( k Rs -- )
jmp, Jump. ( k Rs -- )
rcall, Relative call subroutine. ( k Rs -- )
icall, Extended indirect call to (Z). ( Z Rs -- )
jmp, Extended indirect jump to (Z). ( Z Rs -- )
call, Call subroutine. ( k Rs -- )
ret, Subroutine return. ( k Rs -- )
reti, Interrupt return. ( k Rs -- )
cpe, Compare, skip if equal. ( Rd Rs -- )
ep, Compare. ( Rd Rs -- )
cpe, Compare with carry. ( Rd Rs -- )
cpi, Compare with immediate. ( Rd K Rs -- )
sbr, Skip if bit in register cleared. ( Rd Rs -- )
sbr, Skip if bit in register set. ( Rd Rs -- )
sbc, Skip if bit in I/O register cleared. ( A Rs -- )
sbcs, Skip if bit in I/O register set. ( A Rs -- )

Data transfer instructions

mov, Copy register. ( Rd Rs -- )
mov, Copy register pair. ( Rd Rs -- )
ldi, Load immediate. ( Rd K Rs -- )
lds, Load direct from data space. ( Rd K Rs -- )
l, Load indirect. ( Rd Rs -- )
kl, Load indirect from displacement. ( Rd Rq Rs -- )
sh, Subtract without carry. ( Rd Rs -- )
subl, Subtract immediate. ( Rd K Rs -- )
sbc, Subtract with carry. ( Rd Rs -- )
sbc, Subtract immediate with carry. ( Rd K Rs -- )
sb, Subtract immediate from word. ( Rd K Rs -- )
sbw, Subtract immediate from word. ( Rd K Rs -- )

Add with carry.
adc, Add with carry. ( Rd Rs -- )
ad, Add without carry. ( Rd Rs -- )
ad, Add immediate to word. ( Rd K Rs -- )
sub, Subtract with carry. ( Rd Rs -- )
subl, Subtract immediate. ( Rd K Rs -- )
sbc, Subtract with carry. ( Rd Rs -- )
sb, Subtract immediate from word. ( Rd K Rs -- )

Arithmetic and logic instructions

Register constants

Z ( -- 0 )
Z+ ( -- 1 )
-Z ( -- 2 )
Y ( -- 3 )
Y+ ( -- 4 )
-Y ( -- 5 )
X ( -- 6 )
X+ ( -- 7 )
-X ( -- 8 )
XH:XL ( -- 01 )
YH:YL ( -- 02 )
ZH:ZL ( -- 03 )
R0 ( -- 0 )
R1 ( -- 1 )
R2 ( -- 2 )
R3 ( -- 3 )
R4 ( -- 4 )
R5 ( -- 5 )
R6 ( -- 6 )
R7 ( -- 7 )
R8 ( -- 8 )
R9 ( -- 9 )
R10 ( -- 10 )
R11 ( -- 11 )
R12 ( -- 12 )
R13 ( -- 13 )
R14 ( -- 14 )
R15 ( -- 15 )
R16 ( -- 16 )
R17 ( -- 17 )
R18 ( -- 18 )
R19 ( -- 19 )
R20 ( -- 20 )
R21 ( -- 21 )
R22 ( -- 22 )
R23 ( -- 23 )
R24 ( -- 24 )
R25 ( -- 25 )
R26 ( -- 26 )
R27 ( -- 27 )
R28 ( -- 28 )
R29 ( -- 29 )
R30 ( -- 30 )
R31 ( -- 31 )

Bit and bit-test instructions

lsl, Logical shift left. ( Rd Rs -- )
lsr, Logical shift right. ( Rd Rs -- )
rol, Rotate left through carry. ( Rd Rs -- )
ror, Rotate right through carry. ( Rd Rs -- )
ar, Arithmetic right shift. ( Rd Rs -- )
swap, Swap nibbles. ( Rd Rs -- )
bset, Flag set. ( Rd Rs -- )
bclr, Flag clear. ( Rs -- )
sbi, Set bit in I/O register. ( A Rs -- )
cb, Clear bit in I/O register. ( A Rs -- )
bst, Bit store from register to T. ( Rs T -- )
bld, Bit load from T to register. ( Rs Rs -- )
sec, Set carry. ( Rs Rs -- )
clc, Clear carry. ( Rs Rs -- )
sen, Set negative flag. ( Rs Rs -- )
cln, Clear negative flag. ( Rs Rs -- )
sez, Set zero flag. ( Rs Rs -- )
ciz, Clear zero flag. ( Rs Rs -- )
sei, Global interrupt enable. ( Rs Rs -- )
cli, Global interrupt disable. ( Rs Rs -- )
ses, Set signed test flag. ( Rs Rs -- )
cls, Clear signed test flag. ( Rs Rs -- )
sev, Set two’s complement overflow. ( Rs Rs -- )
clv, Clear two’s complement overflow. ( Rs Rs -- )
set, Set T in SREG. ( Rs Rs -- )
cit, Clear T in SREG. ( Rs Rs -- )
seh, Set half carry flag in SREG. ( Rs Rs -- )
chl, Clear half carry flag in SREG. ( Rs Rs -- )

 MCU control instructions

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

Extras

I²C communications as master

Load these words from i2c_base.txt for a PIC18 microcontroller.
i2cinit, Initializes I²C master mode, 100kHz clock. ( -- )
i2cw, Wake slave. Bit 0 is R/W bit. ( slave-addr -- )
i2cin, The 7-bit I²C address is in bits 7-0. ( -- )
i2can, Write one byte to I²C bus and wait for ACK. ( c -- )
i2can, Read one byte from the I²C bus. ( c -- )
i2can, Read one byte from the I²C bus. ( addr slave-addr -- )
i2can, Write 8-bit address to slave. ( addr slave-addr -- )
i2can, Write 16-bit address to slave ( addr slave-addr -- )

Lower-level words.
ssen Assert start condition. ( -- )
srsen 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 -- )

This guide assembled by Peter Jacobs, School of Mechanical Engineering, The University of Queensland, May 2014 as Report 2014/03.
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.
Microchip 16-bit MCU and DSC Programmers Reference Manual