# f8 manual

Philipp Klaus Krause

2024-07-30

# Chapter 1

# Architecture

## 1.1 Introduction

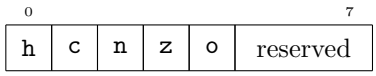
Little-endian. Stack grows downward. 16-bit flat address space.

`pc` after reset: 0x4000. Other registers (including `sp`) after reset: unspecified. (P)ROM/Flash from 0x4000. RAM up to 0x3fff. I/O from 0x0000. Empty PROM/Flash is logically 0x00 (to trigger trap). All instructions execute atomically.

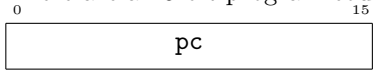
Safety features: trap on opcode 0x00. Trap on write to address 0x0000.

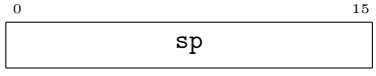
## 1.2 Registers

There is an 8-bit flag register `f`, which contains the half-carry flag `h`, the carry flag `c`, the negative flag `n`, the zero flag `z`, the overflow / parity flag `o`, and three reserved bits. Unless otherwise noted, instructions leave the reserved flags in an undefined state. The reserved bits should not be written by the user except via the `xch f, (n, sp)` instruction.

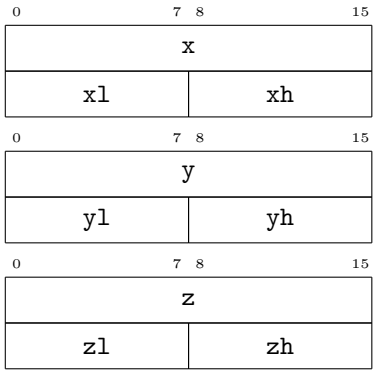



There are a 16-bit program counter `pc` and a 16-bit stack pointer `sp`.




There are three 16-bit general-purpose registers, each consisting of two 8.bit registers.




## 1.3 Instructions

There is the f8 lightweight instruction subset f8l.

Instructions have up to 3 source and up to 2 destination operands. At most one source and one destination operand are in memory.

Each instruction is encoded by 1 to 3 bytes: an optional prefix byte is followed by the opcode byte and 0 to 2 operand bytes.

There are 8 prefix bytes:

| Prefix | semantics | group |
|---|---|---|
| `swapop` | swap operands | 0 |
| `altacc1` | alternative acumulator `xh` instead of `xl` | 1 |
| `altacc2` | alternative acumulator `yl` instead of `xl`, `z` instead of `y` | 2 |
| `altacc4` | alternative acumulator `yh` instead of `xl`, `z` instead of `y` | 2 |
| `altacc3` | alternative acumulator `zl` instead of `xl`, `x` instead of `y` | 2 |
| `altacc5` | alternative acumulator `zh` instead of `xl` | 2 |

## 1.4 Addressing Modes

| | |
|---|---|
| `xl`, `xh`, `yl`, `yh`, `zl`, `zh`, `f` | 8-bit register |
| `x`, `y`, `z`, `sp` | 16-bit register |
| `#i` | 8-bit immediate |
| `#ii` | 16-bit immediate |
| `#d` | 8-bit immediate sign-extended to 16 bit |
| `mm` | direct |
| `(n, sp)`, `(n, y)` | indexed with 8-bit offset |
| `(nn, z)` | indexed with 16-bit offset |
| `(x)`, `(y)`, `(z)` | indirect |

# Chapter 2

# Instructions

| | |
|---|---|
| op8_2 | Any of xh, yl, yh, zl, #i, mm, (n, sp), (nn, z). |
| op8_2ni | Any of xh, yl, yh, zl, mm, (n, sp), (nn, z). |
| altacc8 | Any of xl, xh, yl, yh, zl, zh. |
| op16_2 | Any of x, #ii, mm, (n, sp). |
| op16_2ni | Any of x, mm, (n, sp). |
| altacc16 | Any of x, z. |
| op8_1 | Any of xl, mm, (n, sp), (n, y). |
| op16_1 | Any of y, mm, (n, sp), (nn, z). |

## 2.1 8-bit two-operand instructions

Instructions where the location is used for altacc8 and op8 are not valid.

### 2.1.1 adc: 8-bit addition with carry

| Assembler code | Operation | f8l |
|---|---|---|
| adc xl, op8_2 | xl = xl + op8_2 + c | Yes |
| adc altacc8, op8_2 | altacc8 = altacc8 + op8_2 + c | Yes |
| adc op8_2ni, xl | op8_2ni = op8_2ni + xl + c | Yes |

#### Affected Flags

hcnzo

### 2.1.2 add: 8-bit addition

| Assembler code | Operation | f8l |
|---|---|---|
| add xl, op8_2 | xl = xl + op8_2 | Yes |
| add altacc8, op8_2 | altacc8 = altacc8 + op8_2 | Yes |
| add op8_2ni, xl | op8_2ni = op8_2ni + xl | Yes |

**Affected Flags**

`hcnzo`

### 2.1.3 and: 8-bit bitwise and

| Assembler code | Operation | f8l |
|---|---|---|
| `and xl, op8_2` | `xl = xl & op8_2` | Yes |
| `and altacc8, op8_2` | `altacc8 = altacc8 & op8_2` | Yes |
| `and op8_2ni, xl` | `op8_2ni = op8_2ni & xl` | Yes |

**Affected Flags**

`nz`

### 2.1.4 cp: 8-bit comparison

Subtraction where the result is used to update the flags only.

| Assembler code | Operation | f8l |
|---|---|---|
| `cp xl, op8_2` | `xl + ~op8_2 + 1` | Yes |
| `cp altacc8, op8_2` | `altacc8 + ~op8_2 + 1` | Yes |
| `cp op8_2, xl` | `op8_2 + ~xl + 1` | No |

**Affected Flags**

`hcnzo`

### 2.1.5 or: 8-bit bitwise or

| Assembler code | Operation | f8l |
|---|---|---|
| `or xl, op8_2` | `xl = xl \| op8_2` | Yes |
| `or altacc8, op8_2` | `altacc8 = altacc8 \| op8_2` | Yes |
| `or op8_2ni, xl` | `op8_2ni = op8_2ni \| xl` | Yes |

**Affected Flags**

`nz`

### 2.1.6 sbc: 8-bit subtraction with carry

| Assembler code | Operation | f8l |
|---|---|---|
| `sbc xl, op8_2ni` | `xl = xl + ~op8_2ni + c` | Yes |
| `sbc altacc8, op8_2ni` | `altacc8 = altacc8 + ~op8_2ni + c` | Yes |
| `sbc op8_2ni, xl` | `op8_2ni = op8_2ni + ~xl + c` | No |

**Affected Flags**

`hcnzo`

### 2.1.7 sub: 8-bit subtraction

| Assembler code | Operation | f8l |
|---|---|---|
| `sub xl, op8_2ni` | `xl = xl + ~op8_2ni + 1` | Yes |
| `sub altacc8, op8_2ni` | `altacc8 = altacc8 + ~op8_2ni + 1` | Yes |
| `sub op8_2ni, xl` | `op8_2ni = op8_2ni + ~xl + 1` | No |

#### Affected Flags

`hcnzo`

### 2.1.8 xor: 8-bit bitwise exclusive or

| Assembler code | Operation | f8l |
|---|---|---|
| `xor xl, op8_2` | `xl = xl ^ op8_2` | Yes |
| `xor altacc8, op8_2` | `altacc8 = altacc8 ^ op8_2` | Yes |
| `xor op8_2ni, xl` | `op8_2ni = op8_2ni ^ xl` | Yes |

#### Affected Flags

`nz`

## 2.2 16-bit 2-operand-instructions

Todo: Document possible altacc prefixes.

### 2.2.1 adcw: 16 bit addition with carry

| Assembler code | Operation | f8l |
|---|---|---|
| `adcw xl, op16_2` | `y = y + op16_2 + c` | No |
| `adcw op16_2ni, xl` | `op16_2ni = op16_2ni + y + c` | No |

#### Affected Flags

`cnzo`

### 2.2.2 addw: 16 bit addition

| Assembler code | Operation | f8l |
|---|---|---|
| `adcw xl, op16_2` | `y = y + op16_2` | No |
| `adcw op16_2ni, xl` | `op16_2ni = op16_2ni + y` | No |

#### Affected Flags

`cnzo`

### 2.2.3 orw: 16 bit bitwise or

todo: do we really want the effect on o here? If yes, why not on the 8-bit logic ops?

| Assembler code | Operation | f8l |
|---|---|---|
| `orw xl, op16_2` | `y = y \| op16_2` | No |
| `orw op16_2ni, xl` | `op16_2ni = op16_2ni \| y` | No |

#### Affected Flags

`nzo`

### 2.2.4 sbcw: 16 bit subtraction with carry

| Assembler code | Operation | f8l |
|---|---|---|
| `sbcw xl, op16_2ni` | `y = y + ~op16_2ni + c` | No |
| `sbcw op16_2ni, xl` | `op16_2 = ~op16_2ni + y + c` | No |

#### Affected Flags

`cnzo`

### 2.2.5 subw: 16 bit subtraction

| Assembler code | Operation | f8l |
|---|---|---|
| `subw xl, op16_2ni` | `y = y + ~op16_2ni + 1` | No |
| `subw op16_2ni, xl` | `op16_2 = ~op16_2ni + y + 1` | No |

#### Affected Flags

`cnzo`

### 2.2.6 xorw: 16 bit bitwise exclusive or

todo: do we really want the effect on o here? If yes, why not on the 8-bit logic ops?

| Assembler code | Operation | f8l |
|---|---|---|
| `xorw xl, op16_2` | `y = y ^ op16_2` | No |
| `xorw op16_2ni, xl` | `op16_ni2 = op16_2ni ^ y` | No |

#### Affected Flags

`nzo`

## 2.3 8-bit 1-operand-instructions

### 2.3.1 clr: 8-bit clear

| Assembler code | Operation | f8l |
|---|---|---|
| `clr op8_1` | `op8 = 0x00` | Yes, except (n, y) |
| `clr altacc8` | `altacc8 = 0x00` | Yes |

#### Affected Flags

none

### 2.3.2 dec: 8-bit decrement

| Assembler code | Operation | f8l |
|---|---|---|
| `dec op8_1` | `op8 = op8 + -1` | Yes, except (n, y) |
| `dec altacc8` | `altacc8 = altacc8 + -1` | Yes |

#### Affected Flags

`hcnzo`

### 2.3.3 inc: 8-bit increment

| Assembler code | Operation | f8l |
|---|---|---|
| `inc op8_1` | `op8 = op8 + 1` | Yes, except (n, y) |
| `inc altacc8` | `altacc8 = altacc8 + 1` | Yes |

#### Affected Flags

`hcnzo`

### 2.3.4 push: 8-bit push onto stack

| Assembler code | Operation | f8l |
|---|---|---|
| `push op8_1` | `(--sp) = op8` | Yes, except (n, y) |
| `push altacc8` | `(--sp) = altacc8` | Yes |

#### Affected Flags

none

### 2.3.5 sll: 8-bit shift left logical

| Assembler code | Operation | f8l |
|---|---|---|
| `sll op8_1` | `c = (op8 & 0x80) >> 7`<br>`op8 = op8 << 1` | Yes, except (n, y) |
| `sll altacc8` | `c = (op8 & 0x80) >> 7`<br>`altacc8 = altacc8 << 1` | Yes |

**Affected Flags**

cz

### 2.3.6 srl: 8-bit shift right logical

| Assembler code | Operation | f8l |
|---|---|---|
| `srl op8_1` | `c = op8 & 0x01`<br>`op8 = op8 >> 1` | Yes (except (n, y) |
| `srl altacc8` | `c = op8 & 0x01`<br>`altacc8 = altacc8 >> 1` | Yes |

**Affected Flags**

cz

### 2.3.7 rlc: 8-bit rotate left through carry

| Assembler code | Operation | f8l |
|---|---|---|
| `rlc op8_1` | `tc = (op8 & 0x80) >> 7`<br>`op8 = (op8 << 1) \| c`<br>`c = tc` | Yes (except (n, y) |
| `rlc altacc8` | `tc = (altacc8 & 0x80) >> 7`<br>`altacc8 = (altacc8 << 1) \| c`<br>`c = tc` | Yes |

**Affected Flags**

cz

### 2.3.8 rrc: 8-bit rotate right through carry

| Assembler code | Operation | f8l |
|---|---|---|
| `rrc op8_1` | `tc = op8 & 0x01`<br>`op8 = (op8 >> 1) \| (c << 7)`<br>`c = tc` | Yes (except (n, y) |
| `rrc altacc8` | `tc = altacc8 & 0x01`<br>`altacc8 = (altacc8 >> 1) \| (c << 7)`<br>`c = tc` | Yes |

**Affected Flags**

cz

### 2.3.9 tst: 8-bit test

Set n and z flags according to value of operand, o flag by parity, reset c.

| Assembler code | Operation | f8l |
|---|---|---|
| `tst op8_1` | `op8` | Yes (except (n, y) |
| `tst altacc8` | `altacc8` | Yes |

#### Affected Flags

`cnzo`

## 2.4 16-bit 1-operand-instructions

### 2.4.1 adcw: 16 bit addition with carry

| Assembler code | Operation | f8l |
|---|---|---|
| `adcw op16_1` | `op16 = op16 + c` | No |
| `adcw altacc16` | `altacc16 = altacc16 + c` | No |

#### Affected Flags

`cnzo`

### 2.4.2 clrw: 16-bit clear

| Assembler code | Operation | f8l |
|---|---|---|
| `clrw op16_1` | `op16 = 0x0000` | Yes |
| `clrw altacc15` | `altacc16 = 0x0000` | Yes |

#### Affected Flags

none

### 2.4.3 incw: 16-bit increment

| Assembler code | Operation | f8l |
|---|---|---|
| `incw op16_1` | `op16 = op16 + 1` | Yes |
| `incw altacc16` | `altacc16 = altacc16 + 1` | Yes |

#### Affected Flags

`cnzo`

### 2.4.4 pushw: 16-bit push onto stack

| Assembler code | Operation | f8l |
|---|---|---|
| `pushw op16_1` | `sp -= 2; (sp) = op16` | Yes |
| `pushw altacc16` | `sp -= 2; (sp) = altacc16` | Yes |

**Affected Flags**

none

### 2.4.5 sbcw: 16-bit subtraction with carry

| Assembler code | Operation | f8l |
|---|---|---|
| `sbcw op16_1` | `op16 = op16 + 0xffff + c` | No |
| `sbcw altacc16` | `altacc16 = altacc16 + 0xffff + c` | No |

**Affected Flags**

`cnzo`

### 2.4.6 tstw: 16-bit test

Set n and z flags according to value of operand, o flag by parity, set c.

| Assembler code | Operation | f8l |
|---|---|---|
| `tstw op16_1` | `op16` | Yes |
| `tstw altacc16` | `altacc16` | Yes |

**Affected Flags**

`cnzo`

## 2.5 8-bit loads

### 2.5.1 ld: 8-bit load from memory

| Assembler code | Operation | f8l |
|---|---|---|
| `ld xl, #i` | `xl = #i` | Yes |
| `ld altacc8, #i` | `altacc8 = #i` | Yes |
| `ld xl, mm` | `xl = mm` | Yes |
| `ld altacc8, mm` | `altacc8 = mm` | Yes |
| `ld xl, (n, sp)` | `xl = (n, sp)` | Yes |
| `ld altacc8, (n, sp)` | `altacc8 = (n, sp)` | Yes |
| `ld xl, (nn, z)` | `xl = (nn, z)` | Yes |
| `ld altacc8, (nn, z)` | `altacc8 = (nn, z)` | Yes |
| `ld xl, (y)` | `xl = xh` | Yes |
| `ld altacc8, (altacc16)` | `altacc8 = (altacc16)` | Yes |
| `ld xl, (n, y)` | `xl = (n, y)` | No |
| `ld altacc8, (n, y)` | `altacc8 = (n, y)` | No |

**Affected Flags**

`nz`

### 2.5.2 ld: 8-bit load from register

| Assembler code | Operation | f8l |
|---|---|---|
| `ld xl, xh` | `xl = xh` | Yes |
| `ld xh, xl` | `xh = xl` | Yes |
| `ld altacc8, xh` | `altacc8 = xh` | Yes |
| `ld xl, yl` | `xl = yl` | Yes |
| `ld yl, xl` | `yl = xl` | Yes |
| `ld altacc8, yl` | `altacc8 = yl` | Yes |
| `ld xl, yh` | `xl = yh` | Yes |
| `ld yh, xl` | `yh = xl` | Yes |
| `ld altacc8, yh` | `altacc8 = yh` | Yes |
| `ld xl, zl` | `xl = zl` | Yes |
| `ld zl, xl` | `zl = xl` | Yes |
| `ld altacc8, zl` | `altacc8 = zl` | Yes |
| `ld xl, zh` | `xl = zh` | Yes |
| `ld zh, xl` | `zh = xl` | Yes |
| `ld altacc8, zh` | `altacc8 = zh` | Yes |
| `ld mm, xl` | `mm = xl` | Yes |
| `ld mm, altacc8` | `mm = altacc8` | Yes |
| `ld (n, sp), xl` | `(n, sp) = xl` | Yes |
| `ld (n, sp), altacc8` | `(n, sp) = altacc8` | Yes |
| `ld (nn, z), xl` | `(nn, z) = altacc8` | Yes |
| `ld (nn, z), altacc8` | `(nn, z) = altacc8` | Yes |
| `ld (y), xl` | `(y) = xl` | Yes |
| `ld (altacc16), altacc8` | `(altacc16) = altacc8` | Yes |
| `ld (n, y), xl` | `(n, y) = xl` | No |
| `ld (n, y), altacc8` | `(n, y) = altacc8` | No |

#### Affected Flags

none

### 2.5.3 ldi: 8-bit load with increment

Flags according to old (y).

| Assembler code | Operation | f8l |
|---|---|---|
| `ldi (z), (y)` | `(z) = (y); z += 1;` | No |
| `ldi (z), (x)` | `(z) = (x); z += 1;` | No |

#### Affected Flags

`nz`

## 2.6 16-bit loads

### 2.6.1 ldw: 16-bit load from memory

| Assembler code | Operation | f8l |
|---|---|---|
| `ldw y, #ii` | `y = #ii` | Yes |
| `ldw altacc16, #ii` | `altacc16 = #ii` | Yes |
| `ldw y, mm` | `y = mm` | Yes |
| `ldw altacc16, mm` | `altacc16 = mm` | Yes |
| `ldw y, (n, sp)` | `y = (n, sp)` | Yes |
| `ldw altacc16, (n, sp)` | `altacc16 = (n, sp)` | Yes |
| `ldw y, (nn, z)` | `y = (nn, z)` | Yes |
| `ldw altacc16, (nn, z)` | `altacc16 = (nn, z)` | Yes |
| `ldw y, (n, y)` | `y = (n, y)` | No |
| `ldw altacc16, (n, y)` | `altacc16 = (n, y)` | No |
| `ldw y, (y)` | `y = (y)` | Yes |
| `ldw altacc16, (altacc16)` | `altacc16 = (altacc16)` | Yes |

**Affected Flags**

`nz`

### 2.6.2 ldw 16-bit load from register

| Assembler code | Operation | f8l |
|---|---|---|
| `ldw y, x` | `y = x` | Yes |
| `ldw x, z` | `x = z` | Yes |
| `ldw y, #d` | `y = #d` | Yes |
| `ldw altacc16, #d` | `altacc16 = #d` | Yes |
| `ldw mm, y` | `mm = y` | Yes |
| `ldw mm, altacc16` | `mm = altacc16` | Yes |
| `ldw (n, sp), y` | `(n, sp) = y` | Yes |
| `ldw (n, sp), altacc16` | `(n, sp) = altacc16` | Yes |
| `ldw (nn, z), y` | `(nn, z) = y` | Yes |
| `ldw (nn, z), altacc16` | `(nn, z) = altacc16` | Yes |
| `ldw x, y` | `x = y` | Yes |
| `ldw z, y` | `z = y` | Yes |
| `ldw y, z` | `y = z` | Yes |
| `ldw z, x` | `z = x` | Yes |
| `ldw (y), x` | `(y) = x` | Yes |
| `ldw (z), y` | `(z) = y` | Yes |
| `ldw (x), z` | `(x) = z` | Yes |
| `ldw (y), z` | `(y) = z` | Yes |
| `ldw (n, y), x` | `(n, y) = x` | No |
| `ldw y, sp` | `y = sp` | Yes |
| `ldw sp, y` | `sp = y` | Yes |
| `ldw altacc16, sp` | `altacc16 = sp` | Yes |
| `ldw ((d, sp)), y` | `(d, sp) = y` | No |
| `ldw ((d, sp)), altacc16` | `(d, sp) = altacc16` | No |

#### Affected Flags

none

### 2.6.3 ldwi: 16-bit load with increment

Flags according to old (y).

| Assembler code | Operation | f8l |
|---|---|---|
| `ldwi (z), (y)` | `(z) = (y); z += 2;` | No |
| `ldwi (z), (x)` | `(z) = (x); z += 2;` | No |

#### Affected Flags

`nz`

## 2.7 Other 8-bit instructions

### 2.7.1 bool: 8-bit cast to bool

Todo: Remove from f8l subset?

| Assembler code | Operation | f8l |
|---|---|---|
| `bool xl` | `xl = (bool)xl` | Yes |
| `bool altacc8` | `altacc8 = (bool)altacc8` | Yes |

#### Affected Flags

`z`

### 2.7.2 cax: 8-bit compare and exchange

z is set according to the old value of (y) - zl.

| Assembler code | Operation | f8l |
|---|---|---|
| `cax (y), zl, xl` | `if ((y) == zl) (y) = xl; else zl = (y);` | Yes |
| `cax (y), zl, xh` | `if ((y) == zl) (y) = xh; else zl = (y);` | Yes |
| `cax (y), zl, zh` | `if ((y) == zl) (y) = zh; else zl = (y);` | Yes |

#### Affected Flags

`z`

### 2.7.3 da: decimal adjust

Decimal adjust for addition / subtraction - binary coded decimal semantics.

todo: describe details!

| Assembler code | Operation | f8l |
|---|---|---|
| `da xl` | | Yes |
| `da altacc8` | | Yes |

#### Affected Flags

`hcnzo`

### 2.7.4 mad: multiply and add

| Assembler code | Operation | f8l |
|---|---|---|
| `mad x, mm, yl` | `x = mm * yl + xh + c` | No |
| `mad x, (n, sp), yl` | `x = (n, sp) * yl + xh + c` | No |
| `mad x, (nn, z), yl` | `x = (nn, z) * yl + xh + c` | No |
| `mad x, (z), yl` | `x = (z) * yl + xh + c` | No |

#### Affected Flags

`nz`

### 2.7.5 msk: mask

z flag set according to old value of (y) & ~#i.

| Assembler code | Operation | f8l |
|---|---|---|
| `msk (y), xl, #i` | `(y) = xl & #i \| (y) & ~#i` | Yes |
| `msk (altacc16), altacc8, #i` | `(altacc16) = altacc8 & #i \| (altacc16) & ~#i` | Yes |

#### Affected Flags

`z`

### 2.7.6 pop: 8-bit pop from stack

| Assembler code | Operation | f8l |
|---|---|---|
| `pop xl` | `xl = (sp++)` | Yes |
| `pop altacc8` | `altacc8 = (sp++)` | Yes |

#### Affected Flags

none

### 2.7.7 push: 8-bit push onto stack

Ignores all flags, changes no flags, not even the reserved ones.

| Assembler code | Operation | f8l |
|---|---|---|
| `push #i` | `(--sp) = #i` | Yes |

#### Affected Flags

none

### 2.7.8 rot: 8-bit rotate

| Assembler code | Operation | f8l |
|---|---|---|
| `rot xl, #i` | `xl = (xl << #i) \| (xl >> (8 - #i))` | No |
| `rot altacc8, #i` | `altacc8 = (altacc8 << #i) \| (altacc8 >> (8 - #i))` | No |

#### Affected Flags

none todo: do we want some flags to be affected?

### 2.7.9 sra: 8-bit shift right arithmetic

| Assembler code | Operation | f8l |
|---|---|---|
| `sra xl` | `c = op8 & 0x01`<br>`xl = (xl >> 1) \| xl & 0x80` | Yes |
| `sra altacc8` | `c = op8 & 0x01`<br>`altacc8 = (altacc8 >> 1) \| altacc & 0x80` | Yes |

**Affected Flags**

`cz`

### 2.7.10 thrd

Get current hardware thread number.

| Assembler code | Operation | f8l |
|---|---|---|
| `thrd xl` | `xl` = current hardware thread number | Yes |
| `thrd altacc8` | `altacc8` = current hardware thread number | Yes |

**Affected Flags**

`z`

### 2.7.11 xch: 8-bit exchange

| Assembler code | Operation | f8l |
|---|---|---|
| `xch yl, yh` | `t = yl; yl = yh; yh = t` | Yes |
| `xch xl, xh` | `t = xl; xl = xh; xh = t` | Yes |
| `xch zl, zh` | `t = zl; zl = zh; zh = t` | Yes |
| `xch xl, (n, sp)` | `t = (n, sp); (n, sp) = xl; xl = t` | No |
| `xch altacc8, (n, sp)` | `t = (n, sp); (n, sp) = altacc8; altacc8 = t` | No |
| `xch xl, (y)` | `t = (y); (y) = xl; xl = t` | Yes |
| `xch altacc8, (altacc16)` | `t = (altacc16); (altacc16) = altacc8; altacc8 = t` | Yes |
| `xch f, (n, sp)` | `t = (n, sp); (n, sp) = f; f = t` | Yes |

**Affected Flags**

All, including reserved ones (`xch f, (n, sp)`) or none (all others).

## 2.8 Other 16-bit instructions

### 2.8.1 addw: 16-bit addition

`addw sp, #d` ignores all flags, changes no flags, not even the reserved ones.

| Assembler code | Operation | f8l |
|---|---|---|
| `addw sp, #d` | `sp = sp + #d` | Yes |
| `addw y, #d` | `y = y + #d` | Yes |
| `addw altacc16, #d` | `altacc16 = altacc16 + #d` | Yes |

**Affected Flags**

none (`addw sp, #d`) or `cnzo` (all others).

### 2.8.2 boolw: 16-bit cast to bool

| Assembler code | Operation | f8l |
|---|---|---|
| `boolw y` | `y = (bool)y` | No |
| `boolw altacc16` | `altacc16 = (bool)altacc16` | No |

**Affected Flags**

`z`

### 2.8.3 caxw: 16-bit compare and exchange

z is set according to the old value of (y) - z.

| Assembler code | Operation | f8l |
|---|---|---|
| `caxw (y), z, x` | `if ((y) == z) (y) = x; else z = (y);` | Yes |

**Affected Flags**

`z`

### 2.8.4 cpw: 16-bit comparison

Subtraction where the result is used to update the flags only.

| Assembler code | Operation | f8l |
|---|---|---|
| `cpw y, #ii` | `y + ~#ii + 1` | No |
| `cpw #ii, y` | `#ii + ~y + 1` | No |
| `cpw altacc16, #ii` | `altacc16 + ~#ii + 1` | No |

**Affected Flags**

`cnzo`

### 2.8.5 decw: 16-bit decrement

| Assembler code | Operation | f8l |
|---|---|---|
| `decw (n, sp)` | `(n, sp) = (n, sp) + -1` | No |

**Affected Flags**

`cnzo`

### 2.8.6 incnw: 16-bit increment without carry update

Ignores all flags, changes no flags (except possibly the reserved ones).

| Assembler code | Operation | f8l |
|---|---|---|
| `incnw y` | `y = y + 1` | No |
| `incnw altacc16` | `altacc16 = altacc16 + 1` | No |

**Affected Flags**

none

### 2.8.7 negw: 16-bit negation

| Assembler code | Operation | f8l |
|---|---|---|
| `negw y` | `y = ~y + 1` | No |
| `negw altacc16` | `altacc16 = ~altacc16 + 1` | No |

**Affected Flags**

`cnzo`

### 2.8.8 mul: multiplication

Clears carry.

| Assembler code | Operation | f8l |
|---|---|---|
| `mul y` | `y = yl * yh` | No |
| `mul x` | `x = xl * xh` | No |
| `mul z` | `z = zl * zh` | No |

**Affected Flags**

`cnz`

### 2.8.9 popw: 16-bit pop from stack

| Assembler code | Operation | f8l |
|---|---|---|
| `popw y` | `y = (sp); sp += 2` | Yes |
| `popw altacc16` | `altacc16 = (sp); sp += 2` | Yes |

**Affected Flags**

none

### 2.8.10 pushw: 16-bit push onto stack

| Assembler code | Operation | f8l |
|---|---|---|
| `pushw #ii` | `sp -= 2; (sp) = #ii` | Yes |

**Affected Flags**

none

### 2.8.11 rlcw: 16-bit rotate left through carry

| Assembler code | Operation | f8l |
|---|---|---|
| `rlcw y` | `tc = (y & 0x8000) >> 15`<br>`y = (y >> 1) \| (c << 15)`<br>`c = tc` | No |
| `rlcw (n, sp)` | `tc = ((n, sp) & 0x8000) >> 15`<br>`(n, sp) = ((n, sp) >> 1) \| (c << 15)`<br>`c = tc` | No |
| `rlcw altacc16` | `tc = (altacc16 & 0x8000) >> 15`<br>`altacc16 = (altacc16 >> 1) \| (c << 15)`<br>`c = tc` | No |

**Affected Flags**

`cnz`

### 2.8.12 rrcw: 16-bit rotate right through carry

| Assembler code | Operation | f8l |
|---|---|---|
| `rrcw y` | `tc = y & 0x0001`<br>`y = (y >> 1) \| c`<br>`c = tc` | No |
| `rrcw (n, sp)` | `tc = (n, sp) & 0x0001`<br>`(n, sp) = ((n, sp) << 1) \| c`<br>`c = tc` | No |
| `rrcw altacc16` | `tc = altacc16 & 0x0001`<br>`altacc16 = (altacc16 << 1) \| c`<br>`c = tc` | No |

**Affected Flags**

`cnz`

### 2.8.13 sex: sign-extend

| Assembler code | Operation | f8l |
|---|---|---|
| `sex y, xl` | `y = (int8_t)xl` | No |
| `sex altacc16, altacc8` | `altacc16 = (int8_t)altacc8` | No |

**Affected Flags**

`nz`

### 2.8.14 sllw: 16-bit shift left logical

| Assembler code | Operation | f8l |
|---|---|---|
| `sllw y` | `c = y & (0x8000 >> 15); y = y << 1` | No |
| `sllw altacc16` | `altacc16 = altacc16 << 1` | No |
| `sllw y, xl` | `c = y & (0x8000 >> 15); y = y << xl` | No |
| `sllw altacc16, altacc8` | `altacc16 = altacc16 << altacc8` | No |

#### Affected Flags

`cnz` (`sllw y` and `sllw altacc16`) or `nz` (others).

### 2.8.15 sraw: 16-bit shift right arithmetic

| Assembler code | Operation | f8l |
|---|---|---|
| `sraw y` | `c = y & 0x0001; y = y >> 1 \| y & 0x8000` | No |
| `sraw altacc16` | `c = y & 0x0001; altacc16 = altacc16 >> 1 \| altacc16 & 0x8000` | No |

#### Affected Flags

`cnz`

### 2.8.16 srlw: 16-bit shift right logical

| Assembler code | Operation | f8l |
|---|---|---|
| `srlw y` | `c = y & 0x0001; y = y >> 1` | No |
| `srlw altacc16` | `c = y & 0x0001; altacc16 = altacc16 >> 1` | No |

#### Affected Flags

`cnz`

### 2.8.17 xchw: 16-bit exchange

| Assembler code | Operation | f8l |
|---|---|---|
| `xchw x, (y)` | `t = x; x = (y); (y) = t` | Yes |
| `xchw y, (z)` | `t = y; y = (z); (z) = t` | Yes |
| `xchw z, (x)` | `t = z; z = (x); (x) = t` | Yes |
| `xchw z, (y)` | `t = z; z = (y); (y) = t` | Yes |
| `xchw y, (n, sp)` | `t = y; y = (n, sp); (n, sp) = t` | No |
| `xchw altacc16, (n, sp)` | `t = altacc16; altacc16 = (n, sp); (n, sp) = t` | No |

#### Affected Flags

none

### 2.8.18 zex: zero-extend

| Assembler code | Operation | f8l |
|---|---|---|
| `zex y, xl` | `y = xl` | No |
| `zex altacc16, altacc8` | `altacc16 = altacc8` | No |

#### Affected Flags

`z`

## 2.9 Bit Instructions

### 2.9.1 xchb: exchange bit

Exchange xl with bit b at mm. z flag according to new value of xl.

todo: is it really worth having this?

| Assembler code | Operation | f8l |
|---|---|---|
| `xchb xl, mm, #b` | `t = mm >> #b`<br>`mm = mm & ~(1 << #b) \| (xl << #b)`<br>`xl = t` | No |
| `xchb altacc8, mm, #b` | `t = mm >> #b`<br>`mm = mm & ~(1 << #b) \| (altacc8 << #b)`<br>`altacc8 = t` | No |

#### Affected Flags

`z`

## 2.10 Relative Jumps

### 2.10.1 dnjnz

dnjnz yh, #d (2) ; decrement yh, without updating carry, jump if result is not zero.

#### Affected Flags

`nz`

### 2.10.2 jr

`jr #d` ignores all flags, changes no flags, not even reserved ones.

| Assembler code | Operation | f8l |
|---|---|---|
| `jr #d` | `pc += #d` | Yes |

**Affected Flags**

none

### 2.10.3 jrc

| Assembler code | Operation | f8l |
|---|---|---|
| `jr #d` | `if (c) pc += #d;` | Yes |

**Affected Flags**

none

### 2.10.4 jrgt

| Assembler code | Operation | f8l |
|---|---|---|
| `jrgt #d` | `if (c && !z) pc += #d;` | Yes |

**Affected Flags**

none

### 2.10.5 jrle

| Assembler code | Operation | f8l |
|---|---|---|
| `jrle #d` | `if (!c \|\| z) pc += #d;` | Yes |

**Affected Flags**

none

### 2.10.6 jrn

| Assembler code | Operation | f8l |
|---|---|---|
| `jrn #d` | `if (n) pc += #d;` | Yes |

**Affected Flags**

none

### 2.10.7 jrnc

| Assembler code | Operation | f8l |
|---|---|---|
| `jrnc #d` | `if (!c) pc += #d;` | Yes |

**Affected Flags**

none

### 2.10.8 jrnn

| Assembler code | Operation | f8l |
|---|---|---|
| `jrnn #d` | `if (!n) pc += #d;` | Yes |

**Affected Flags**

none

### 2.10.9 jrno

| Assembler code | Operation | f8l |
|---|---|---|
| `jrno #d` | `if (!o) pc += #d;` | Yes |

**Affected Flags**

none

### 2.10.10 jrnz

| Assembler code | Operation | f8l |
|---|---|---|
| `jrnz #d` | `if (!n) pc += #d;` | Yes |

**Affected Flags**

none

### 2.10.11 jro

| Assembler code | Operation | f8l |
|---|---|---|
| `jro #d` | `if (o) pc += #d;` | Yes |

**Affected Flags**

none

### 2.10.12 jrsge

| Assembler code | Operation | f8l |
|---|---|---|
| `jrsge #d` | `if (!(n ^ o)) pc += #d;` | Yes |

**Affected Flags**

none

### 2.10.13 jrsgt

| Assembler code | Operation | f8l |
|---|---|---|
| `jrsgt #d` | `if (!z && !(n ^ o)) pc += #d;` | Yes |

**Affected Flags**

none

### 2.10.14 jrsle

| Assembler code | Operation | f8l |
|---|---|---|
| `jrsle #d` | `if (z \|\| (n ^ o)) pc += #d;` | Yes |

**Affected Flags**

none

### 2.10.15 jrslt

| Assembler code | Operation | f8l |
|---|---|---|
| `jrslt #d` | `if (n ^ o) pc += #d;` | Yes |

**Affected Flags**

none

### 2.10.16 jrz

| Assembler code | Operation | f8l |
|---|---|---|
| `jrz #d` | `if (z) pc += #d;` | Yes |

**Affected Flags**

none

## 2.11 Other Instructions

### 2.11.1 call

| Assembler code | Operation | f8l |
|---|---|---|
| `call #ii` | `sp -= 2; (sp) = pc; pc = #ii` | Yes |
| `call y` | `sp -= 2; (sp) = pc; pc = y` | Yes |
| `call altacc16` | `sp -= 2; (sp) = pc; pc = altacc16` | Yes |

**Affected Flags**

none

### 2.11.2 jp: jump

`jp #ii` ignores all flags, changes no flags, not even reserved ones.

| Assembler code | Operation | f8l |
|---|---|---|
| `jp #ii` | `pc = #ii` | Yes |
| `jp y` | `pc = y` | Yes |
| `jp altacc16` | `pc = altacc16` | Yes |

#### Affected Flags

none

### 2.11.3 ret: return

| Assembler code | Operation | f8l |
|---|---|---|
| `ret` | `pc = (sp); sp += 2` | Yes |

#### Affected Flags

none

### 2.11.4 reti: return from interrupt

Ignores all flags, changes no flags, not even reserved ones.

| Assembler code | Operation | f8l |
|---|---|---|
| `reti` | `pc = (sp); sp += 2` | Yes |

#### Affected Flags

none

### 2.11.5 trap

Opcode 0x00. Trap reset.

| Assembler code | Operation | f8l |
|---|---|---|
| `trap` | Trap reset | Yes |

# Chapter 3

# Opcode Map

todo - see opcodemap.ods for now.

# Chapter 4

# Peripherals

Unless otherwise noted, the vlaue of I/O registers on reset is unspecified.

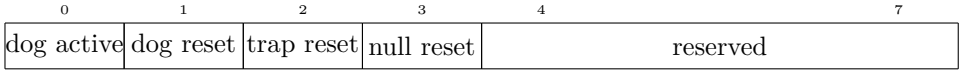
## 4.1 Watchdog and Reset

The watchdog has an 8-bit configuration register and a 16-bit counter register.

When the watchdog is active, the system clock is divided by 16, and then used to increment the counter register.

The system is reset when a power-on reset happens, the watchdog counter register reaches 0xffff, the trap instruction is executed, or the byte at memory address 0 is written.

### Configuration Register




The lowest bit of the configuration register decides if the watchdog is active. It is 0 on reset. The following three bits give the reason of the previous reset. On a power-on-reset they are all 0.

## 4.2 Interrupt Controller

The interrupt controller has a 16-bit enable register, and a 16-bit active register.

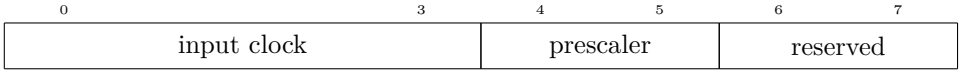

When an interrupt happens and the corresponding bit in the enable register is set, the corresponding bit in the active register is set. When a bit in the active register is set, and no interrupt routine is currently executing, the program

counter is put onto the stack and then set to 0x4004. From then on, an interrupt routine is considered to be executing until the `reti` instruction is executed.

Bit 0 of the enable register indicates that timer 0 overflow interrupts are enabled. Bit 0 of the active register indicates that a timer0 overflow interrupt is active. Bit 1 of the enable register indicates that timer 0 compare interrupts are enabled. Bit 1 of the active register indicates that a timer 0 compare interrupt is active. These bits are 0 on reset. All other bits are reserved.

## 4.3 Timer

The timer has an 8-bit configuration register and 16-bit counter, reload and comparison registers.




The lowest 4 bits of the configurationr egister select the clock source (0 none, 1 system clock, 2 to 15 for other inputs), the next 2 select the prescaler factor (0 for 1, 1 for 4, 2 for 16, 3 for 64). All 6 bits are 0 on reset.

The timer increments the 16-bit counter register. When incrementing from 0xffff, a timer overflow interrupt happens, and the value from the reload register gets loaded into the counter register instead. When the timer register gets incremented to the value of the compare register, a timer compare interrupt happens.

## 4.4 GPIO

The GPIO has (up to 16 bit) data direction, output data, input data, pull-up registers.