f8 Manual

Philipp Klaus Krause

July 29, 2024

Chapter 1

Architecture

1.1 Introduction

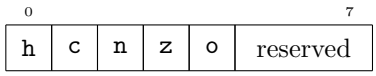
Little-endian. Stack grows downward. 16-bit flat address space.

`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.

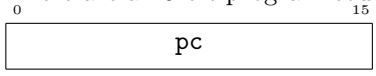
Safety features: trap on opcode 0x00. Trap on write to address 0x0000.

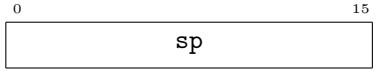
1.2 Registers

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.

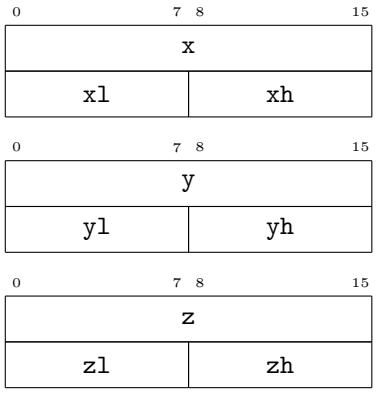


There are a 16-bit program counter `pc` and a 16-bit stack pointer `sp`.





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`	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

`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_2`	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)`.

Chapter 2

Instructions

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

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

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	Prefix
`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	Prefix
`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	Prefix
`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` `; op8 = op8 << 1`	Yes, except (n, y)
`sll altacc8`	`; c = (op8 & 0x80) >> 7` `; 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` `; op8 = op8 >> 1`	Yes (except (n, y)
`srl altacc8`	`; c = op8 & 0x01` `; 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`
	`; op8 = (op8 << 1) | c`
	`; c = tc`
		Yes (except (n, y)

`rlc altacc8`	`; tc = (altacc8 & 0x80) >> 7`
	`; altacc8 = (altacc8 << 1) | c`
	`; 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`
	`; op8 = (op8 >> 1) | (c << 7)`
	`; c = tc`
		Yes (except (n, y)

`rrc altacc8`	`; tc = altacc8 & 0x01`
	`; altacc8 = (altacc8 >> 1) | (c << 7)`
	`; 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

ld xl, #i (12)
ld xl, mm [zn] (12)
ld xl, (n, sp) [zn] (12)
ld xl, (nn, z) [zn] (12)
ld xl, (y) [zn] (12)
ld xl, (n, y) [zn] (12)
ld xl, xh (02)
ld xl, yl (012)
ld xl, yh (012)
ld xl, zl (012)
ld xl, zh (012)
ld mm, xl (12)
ld (n, sp), xl (12)
ld (nn, z), xl (12)
ld (y), xl (12)
ld (n, y), xl (12)

2.5.2 ldi

ldi (z), (y) (2) ; (z++) = (y)

Affected Flags

`nz`

2.6 16-bit loads

2.6.1 ldw

ldw y, #ii (2)
ldw y, mm [ZN] (2)

ldw y, (n, sp) [ZN] (2)
ldw y, (nn, z) [ZN] (2)
ldw y, (n, y) [ZN] (2)
ldw y, (y) [ZN] (2)
ldw y, x (2)
ldw y, #d (2)
ldw mm, y (2)
ldw (n, sp), y (2)
ldw (nn, z), y (2)
ldw x, y
ldw z, y (0)
ldw (y), x (2)
ldw (n, y), x
ldw y, sp (2)
ldw ((d, sp)), y (2)

2.6.2 ldwi

ldwi (z), (y) (2) ; (z++) = (y); (z++) = (y)

Affected Flags

nz

2.7 Other 8-bit instructions

2.7.1 bool: 8-bit cast to bool

bool xl (12) ; cast to bool

Affected Flags

z

2.7.2 cax: 8-bit compare and exchange

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

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

da xl [ocznh] (12) ; decimal adjust for addition / subtraction - binary coded decimal semantics

2.7.4 mad: multiply and add

mad x, mm, yl [zn] ; x <- m * yl + xh + c
mad x, (n, sp), yl [zn] ; x <- (n, sp) * yl + xh + c
mad x, (nn, z), yl [zn] ; x <- (nn, z) * yl + xh + c
mad x, (z), yl [zn] ; x <- (z) * yl + xh + c

2.7.5 msk: mask

msk (y), xl, #i [z](12) ; (y) = xl & i | (y) & i; z flag according to (y) & i

2.7.6 pop: 8-bit pop from stack

pop xl (12) ; xl = (sp++)

2.7.7 push: 8-bit push onto stack

push #i [] ; Ignores all flags, changes no flags, not even the reserved ones.

2.7.8 rot: 8-bit rotate

rot xl, #i (12) ; rotate xl left by i

2.7.9 sra: 8-bit shift right arithmetic

sra xl (12)

Affected Flags

`cz`

2.7.10 thrd

thrd xl (12) ; get current hardware thread number

Affected Flags

`z`

2.7.11 xch: 8-bit exchange

xch xl, (n, sp) (12)
xch xl, (y) (12)
xch yl, yh (2) ; exchange yl with yh.
xch f, (n, sp) [xxxocznh]

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.
addw y, #d [OCZN] (2)

2.8.2 boolw: 16-bit cast to bool

boolw y (2)

Affected Flags

z

2.8.3 caxw: 16-bit compare and exchange

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

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

Affected Flags

z

2.8.4 cpw: 16-bit comparison

cpw y, #ii (02)

Affected Flags

cnzo

2.8.5 decw: 16-bit decrement

decw (n, sp)

Affected Flags

cnzo

2.8.6 incnw: 16-bit increment without carry update

incnw y (2) ; Ignores all flags, changes no flags (except possibly the reserved ones).

Affected Flags

none

2.8.7 negw: 16-bit negation

negw y (2) ; y = y + 1

Affected Flags

`cnzo`

2.8.8 mul: multiplication

mul y [ZNC] (2) ; y = yl * yh , clears carry.

2.8.9 popw: 16-bit pop from stack

popw y (2)

2.8.10 pushw: 16-bit push onto stack

pushw #ii []

2.8.11 rlcw: 16-bit rotate left through carry

rlcw y [ZNC] (2)
rlcw (n, sp) [ZNC]

2.8.12 rrcw: 16-bit rotate right through carry

rrcw y [ZNC] (2)
rrcw (n, sp) [ZNC]

2.8.13 sex: sign-extend

sex y, xl (12) [ZN] ; sign-extend xl to y

2.8.14 sllw: 16-bit shift left logical

sllw y [ZNC] (2)
sllw y, xl [ZN] (12)

2.8.15 sraw: 16-bit shift right arithmetic

sraw y [ZNC] (2)

2.8.16 srlw: 16-bit shift right logic

srlw y [ZNC] (2)

2.8.17 xchw: 16-bit exchange

xchw x, (y) (2)
    xchw y, (n, sp) [] (2)

2.8.18 zex: zero-extend

zex y, xl (12) ; zero-extend xl to y

Affected Flags

z

2.9 Bit Instructions

2.9.1 xchb: exchange bit

xchb xl, mm, #b (12) ; exchange xl with bit b at mm. z flag according to new value of xl.

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 ; jump relative to pc. Ignores all flags, changes no flags, not even reserved ones.

2.10.3 jrc

jrc #d ; if c

2.10.4 jrle

jrle #d ; if !c or z

2.10.5 jrn

jrn #d ; if n

2.10.6 jrnc

jrnc #d ; if !c

2.10.7 jrnn

jrnn #d ; if !n

2.10.8 jrno

jrno #d ; if !o

2.10.9 jrnz

jrnz #d ; if !z

2.10.10 jrsge

jrsge #d ; if n xnor o

2.10.11 jrsle

jrsle #d ; if (z or (n xor o))

2.10.12 jrslt

jrslt #d ; if n xor o

2.10.13 jrz

jrz #d ; jump relative to pc if z flag is set.

2.11 Other Instructions

2.11.1 call

call #ii
    call y (2)

2.11.2 jp: jump

jp #ii ; Ignores all flags, changes no flags, not even reserved ones.
    jp y (2)

2.11.3 ret: return

ret

2.11.4 reti: return from interrupt

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

2.11.5 trap

trap ; Opcode 0x00. Trap reset.

Chapter 3

Opcode Map

todo