mnemonic assembler — **machine instruction** — **effect /meaning**

mnem.	ope-rands	FLAG	g	b	yy	xx	opcode/binary code (g group, b bit, yy,xx t.p. opera./reg.) — *addressing mode* —
NOP			000	0	00	00	No OPeration
CMP		NZVC	000	0	00	01	CoMPare D0, D1 (D0-D1)
SWD			000	0	00	10	SWap D0, D1
SWM			000	0	00	11	SWap Memory Adr(D0), Adr(D1)
							arithmetic-/ stack commands ----------
MUL		NZVC	000	0	01	00	MULtiplication D0 <- D0*D1
DIV		NZVC	000	0	01	01	DIVision D0 <- D0\D1 Rest SD0
PSA			000	0	01	10	PuSh All Stack <- A,D,SR reg.
POA		NZVC	000	0	01	11	POp All SR,D,A reg. <- Stack
							jump commands/ subroutine --- *IC relative* ---
JSR			000	0	10	00	Jump SubRoutine Stack<-IC+1, A0=0: IC<-IC+A1 else IC<-A0+A1+ST
JIN	+A		000	0	10	01	N=1: IC <- IC +A1 Jump If Negative
JIZ	+A		000	0	10	10	Z=1: IC <- IC +A1 Jump If Zero
JMP	+A		000	0	10	11	IC <- IC +A1 JuMP
				---			----- *Adr. with Offset* -----
RET			000	0	11	00	RETturn subroutine IC <-stack
JIN	IA		000	0	11	01	N=1: IC <- A0 +A1 +ST Jump If Negative
JIZ	IA		000	0	11	10	Z=1: IC <- A0 +A1 +ST Jump If Zero
JMP	IA		000	0	11	11	IC <- A0 +A1 +ST JuMP
							load constant --- *register direct* ---
LDC	Rg	NZ	000	1	00	xx	LoaD Constant reg xx <- ## next byte
##	CCCC CCCC						## Constant byte
				---			*register direct* ----------
JIN	RG		000	1	01	xx	N=1: IC <- reg xx Jump If Negative
JIZ	RG		000	1	10	xx	Z=1: IC <- reg xx Jump If Zero
JMP	RG		000	1	11	xx	IC <- reg xx JuMP
							in- output / stack commands -- *reg. direct* --
INP	RG (IO)	NZ	001	0	00	xx	reg xx <- INPut
OUT	RG (IO)		001	0	01	xx	OUTput <- reg xx
PSH	RG	NZ	001	0	10	xx	PuSH Stack <- reg xx
POP	RG	NZ	001	0	11	xx	POP reg xx <- Stack
							shadow reg./ bit test/nibble -- *Reg. direkt* --
SSR	RG	NZ	001	1	00	xx	Set Shadow Reg xx -> Sxx
GSR	RG	NZ	001	1	01	xx	Get Shadow Reg xx <- Sxx
BTS	RG	Z	001	1	10	xx	BitTest reg xx with Shadow r.
SWN	RG	NZ	001	1	11	xx	SWap Nibble reg xx
							arithmet.-log. commands --- *register direct* ---
SHL	RG	NZ C	010	0	00	xx	SHift Left reg xx
SHR	RG	Z C	010	0	01	xx	SHift Right reg xx
ROL	RG	NZ C	010	0	10	xx	ROtate Left reg xx
ROR	RG	NZ C	010	0	11	xx	ROtate Right reg xx

CLR	RG	Z	010	1	00	xx	CLear Register reg xx <- 0
INC	RG	NZVC	010	1	01	xx	INCrement reg xx
DEC	RG	NZVC	010	1	10	xx	DECrement reg xx
NOT	RG	NZ	010	1	11	xx	reg xx <- NOT reg xx
							--- reg.reg./*1234) transport commands with SR/ SP
AND	RG RG	NZ	011	0	yy	xx	reg yy <- reg yy AND reg xx *1)
OR	RG RG	NZ	011	1	yy	xx	reg yy <- reg yy OR reg xx *2)
				------			--- *register direct and indirect* ---
ADD	RG RG	NZVC	100	0	yy	xx	reg yy <- reg yy + reg xx *3)
ADD	RG [RG]	NZVC	100	1	yy	xx	reg yy <- reg yy + Adr(reg xx)

SUB	RG RG	NZVC	101	0	yy	xx	reg yy <- reg yy - reg xx *4)
SUB	RG [RG]	NZVC	101	1	yy	xx	reg yy <- reg yy - Adr(reg xx)
							--- transport commands --- *register indirect* -----
MOV	[RG] RG	NZ	110	0	yy	xx	MOVe Adr(reg yy) <- reg xx
MOV	RG [RG]	NZ	110	1	yy	xx	MOVe reg yy <- Adr(reg xx)
				------			--- *register direct* ---
MOV	RG RG	NZ	111	0	yy	xx	MOVe reg yy <- reg xx *5)
				---	Reg.		--- *register indirect* ---
MOV	RG IA	NZ	111	1	00	xx	MOVe reg xx <- Adr(A0+A1+ST)
MOV	IA RG	NZ	111	1	01	xx	MOVe Adr(A0+A1+ST)<- reg xx
LOD	RG	NZ	111	1	10	xx	LOad Data reg xx <- Adr(h80+A1+ST)
							- store commands /copy command /stop ------
STO	(IO)	Z	111	1	11	00	STOre file <- Adr(h80+A1+ST) D0 byte, via D1 (Data: IO)
RCL	(IO)	ZV	111	1	11	01	ReCalL Adr(h80+A1+ST) <- file via D1 (Data: IO) /D0 byte readed
CPY		ZV	111	1	11	10	CoPY Adr(A1)<-Adr(A0) D0 byte via D1
STP	(pre-cmd)		111	1	11	11	StoP STop Program, flags from pre-command

register (0 to 3 generic. reg.)

No	name	xx bin	type
0	D0	00	data register
1	D1	01	data register
2	A0	10	adr.-base reg.
3	A1	11	adr.-index reg.

SD0/SD1 shadow reg. to D0/D1
SA0/SA1 shadow reg. to A0/A1
IR Instruction Register
IC Instruction Counter
SR Status Register
SP Stack Pointer
ST STart adress

A1 is signed as offset +A, IA

Status Register SR
IOXYNZVC
In/ IO=00 Character
Out =01 decimal
=10 hexadecimal
=11 binary
free available XY
Flags N Negative
Z Zero
V oVerflow
C Carry

addressing mode in assembler

* Adr./register fix

value in/into register:
* register direct
reg xx, reg yy **.Rg.**

value in/into RAM(Adr.in Reg.)
* register indirect
Adr(reg xx), yy **[Rg]**

Adr. with offset
* IC relative
Adr.: IC + A1 **.+A.**

* Index Adress
Adr.: A0 + A1 + ST **.IA.**

* at LOD/STO/RCL
Adr.: h80 + A1 + ST

A1 as index with sign

|*12345) with xx≠yy,
| at xx=yy code with SR/SP:
|*1) MOV RG SR NZ reg yy <- SR
|*2) MOV RG SP NZ reg yy <- SP
|*3) MOV SR RG NZ SR <- reg xx
|*4) MOV SP RG NZ SP <- reg xx
|*5) MOV SR CC SR <- CC..
| CC sets data type IO in SR

RAM adresses:

h00 standard start address
h80 base adr. data (down)
hF5 base adr. stack (up)
hF6 address keyboard input
hF7 adresss display pixel
hF8 - hFF MGA display 8x8

If the IC runs under h00/over hFF it is cyclically set into the valid range, as well as the operands adr(h80/A0 +A1+ST).
No specification for FLAG: Flag is set to 0. Data IO: 00 character, 01 decimal, 10 hexadecimal, 11 binary (one value/row).

explanation and details

The FLAG's NZVC not specified in the commands are always set to 0.
The higher-value Nibble IOXY of the SR is not affected.

```
      ope-       opcode/bin.code     meaning /explanation
mnem. rands   FLAG   g  b yy xx  (g group, b bit, yy,xx t.p. opera./reg.)
-----------------------------------------------------------------
              NZVC
NOP                  000 0 00 00      No OPeration
```

No operation is executed, flags all 0, the instruction counter `IC` is incremented by 1.

```
CMP           NZVC 000 0 00 01      CoMPare D0, D1 (D0-D1)
```

Compares the data registers `D0` and `D1` by difference `D0-D1` and sets the corresponding flags.
The registers remain unchanged, the difference is discarded.

```
SWD                  000 0 00 10      SWap D0, D1
```

Swaps the contents of the two data registers `D0` and `D1`.

```
SWM                  000 0 00 11      SWap Memory  Adr(D0), Adr(D1)
```

Swaps the contents of the memory cells whose addresses are in registers `D0, D1`.

```
MUL           NZVC 000 0 01 00      MULtiplication D0 <- D0*D1
```

Multiplies `D0*D1` by Booth, result in `D0`. Also in case of overflow the low-value byte is returned.

```
DIV           NZVC 000 0 01 01      DIVision D0 <- D0\D1  Remainder SD0
```

Integer division `D0\D1`, result in `D0`, remainder in `SD0`. Sign remainder = sign dividend.

```
PSA                  000 0 01 10      PuSh All Stack <- A,D,SR reg.
```

Pushes registers `A1, A0, D1, D0, SR` on the stack in this order (`SR` on top) and decreases the `SP` by 5.

```
POA           NZVC 000 0 01 11      POp All SR,D,A reg. <- Stack
```

Sets the registers in the order `SR, D0, D1, A0, A1` with the values from the stack and increases the `SP` by 5.
The flags thus result from the corresponding bits on the stack.

```
JSR                  000 0 10 00      Jump SubRoutine Stack<-IC+1, A0=0: IC<-IC+A1 sonst IC<-A0+A1+ST
```

Jump to the subroutine. Places the return address`IC+1` on the stack.
If `A0=0`, `A1` is added to the `IC` and entered into the instruction counter where it is processed further. Otherwise the new `IC` is determined by the index address `A0+A1+ST`. `A1` is signed.

```
JIN +A               000 0 10 01      N=1: IC <- IC +A1       Jump If Negative
```

Jumps `A1` bytes relative to the instruction counter state when the N flag is set, where `A1` is signed.

```
JIZ +A               000 0 10 10      Z=1: IC <- IC +A1       Jump If Zero
```

Jumps `A1` bytes relative to the instruction counter state when the Z flag is set, where `A1` is signed.

```
JMP +A               000 0 10 11           IC <- IC +A1       JuMP
```

Jumps `A1` bytes relative to the instruction counter state, where `A1` is signed.

```
RET                  000 0 11 00      RETturn subroutine IC <-Stack
```

Return from the subroutine. Fetches the address of the instruction counter from the stack to continue working at the instruction after `JSR` if the stack is managed correctly.

```
JIN IA               000 0 11 01      N=1: IC <- A0 +A1 +ST   Jump If Negative
```

Jumps to the calculated index address `A0 +A1 +ST` when the N flag is set, where `A1` is signed.

```
JIZ IA               000 0 11 10      Z=1: IC <- A0 +A1 +ST   Jump If Zero
```

Jumps to the calculated index address `A0 +A1 +ST` when the Z flag is set, where `A1` is signed.

```
JMP IA               000 0 11 11           IC <- A0 +A1 +ST   JuMP
```

Jumps to the calculated index address `A0 +A1 +ST`, where `A1` is signed.

```
LDC Rg        NZ     000 1 00 xx      LoaD Constant  reg xx <- ## next byte
```

Loads the constant stored in the byte after the LDC instruction with the constant instruction ## into the register. The instruction counter is incremented by 2 when the instruction is executed, an wrong entered instruction is thus skipped.

```
##  CCCC CCCC                        ## Constant byte
```

Constant definition 1byte (left and right nibble) for the LDC command. Must immediately follow this command.
If the definition is not after LDC, the byte is interpreted as a machine instruction and executed accordingly.

```
JIN RG            000 1 01 xx      N=1: IC <- reg xx      Jump If Negative
```
Jumps to the address in the register when the N flag is set.

```
JIZ RG            000 1 10 xx      Z=1: IC <- reg xx      Jump If Zero
```
Jumps to the address in the register when the Z flag is set.

```
JMP RG            000 1 11 xx           IC <- reg xx      JuMP
```
Jumps to the address in the register.

```
INP RG (IO)   NZ    001 0 00 xx      reg xx         <- INPut
```
The program stops and waits for the input of a keyboard character, a decimal number -128 ... 127 or a hexadecimal number. The input is displayed in binary and alternatively in the forms not entered. The binary value is filed in the register and stored in the RAM in the keyboard byte `hF6`. The input appears in the input/output protocol in the form defined by the `IO` bits of the status register.

```
OUT RG (IO)       001 0 01 xx      OUTput         <- reg xx
```
The register contain is represented in the input/output system in binary, hexadecimal, decimal and as characters and is noted in the input/output protocol in the form defined by the `IO` bits of the status register.

```
PSH RG        NZ    001 0 10 xx      PuSH  Stack    <- reg xx
```
Puts the register on top of the stack and decreases the `SP` by 1.

```
POP RG        NZ    001 0 11 xx      POP  reg xx    <- Stack
```
Sets the register to the value from the stack and increases the `SP` by 1.

```
SSR RG        NZ    001 1 00 xx      Set Shadow Reg xx -> Sxx  (SDx, SAx)
```
Sets the shadow register to the register to its value.

```
GSR RG        NZ    001 1 01 xx      Get Shadow Reg xx <- Sxx  (SDx, SAx)
```
Sets the register to the value of its shadow register.

```
BTS RG         Z    001 1 10 xx      BitTest reg xx with Shadow register
```
Tests the corresponding bits of the register against the set bits in the shadow register. If all bits set in the shadow register are also set in the register, i.e. the masked difference is 0, the Zero flag is set.
The N flag is set when the shadow register is negative but the register is positive.

```
SWN RG        NZ    001 1 11 xx      SWap Nibble reg xx
```
Swaps the left and right nibble in the register. Corresponds to the digit swap of the hexadecimal number.

```
SHL RG        NZ C 010 0 00 xx      SHift Left   reg xx
```
Shifts the bits in the register one place to the left. The right bit becomes 0.
Corresponds to a multiplication by 2. The bit shifted out goes into the Carry flag.

```
SHR RG         Z C 010 0 01 xx      SHift Right  reg xx
```
Shifts the bits in the register one place to the right. The left bit becomes 0.
Corresponds to integer division \2. The bit shifted out (corresponds to remainder) goes into the carry flag.

```
ROL RG        NZ C 010 0 10 xx      ROtate Left  reg xx
```
Shifts the bits in the register one place to the left. The left bit is set to the right bit and also to the C flag.

```
ROR RG        NZ C 010 0 11 xx      ROtate Right reg xx
```
Shifts the bits in the register one place to the right. The right bit is placed in the left bit and also in the C flag.

```
CLR RG         Z    010 1 00 xx      CLear Register reg xx <- 0
```
Clears the register, i.e. sets all bits of the register to 0. The Z flag is thus always set.

```
INC RG        NZVC 010 1 01 xx      INCrement reg xx
```
Increases the register content by one.

```
DEC RG        NZVC 010 1 10 xx      DECrement reg xx
```
Decreases the register content by one.

```
NOT RG        NZ   010 1 11 xx      reg xx <- NOT reg xx
```
Negates the individual bits in the register. (If INC is executed afterwards, the sign of the decimal number is changed).

The commands `AND/OR/ADD/SUB/MOV RG RG`

do not exist for identical registers. With `AND/OR` *the result would be identical to the operand, with* `ADD` *it corresponds to the doubling, which is better covered by* `SHL`*. For* `SUB`, `CLR` *is also more efficient.* `MOV` *makes no sense. The resulting free binary codes are used for the* `MOV` *transport commands with the status register and the stack pointer and the data type* `IO` *can be set.*

`AND RG RG NZ 011 0 yy xx reg yy <- reg yy AND reg xx (xx ≠ yy)`
Operates a logical AND between the individual bits. Result is only 1 if both bits are 1, otherwise 0.

`MOV RG SR NZ 011 0 yy yy reg yy <- SR`
Moves/copies the contents of the status register to the register on the left.

`OR RG RG NZ 011 1 yy xx reg yy <- reg yy OR reg xx (xx ≠ yy)`
Operates a logical OR between the bits. Results in 0 only if both bits are 0, otherwise 1.

`MOV RG SP NZ 011 1 yy yy reg yy <- SP`
Moves/copies the contents of the stack pointer to the register on the left.

`ADD RG RG NZVC 100 0 yy xx reg yy <- reg yy + reg xx (xx ≠ yy)`
Adds the two registers and puts the result in the register on the left.

`MOV SR RG NZ 100 0 xx xx SR <- reg xx`
Moves/copies the content of the register on the right into the status register and thus sets the flags.

`ADD RG [RG] NZVC 100 1 yy xx reg yy <- reg yy + Adr(reg xx)`
Adds the left register with the byte whose address is in the right register. Result then in the left register.

`SUB RG RG NZVC 101 0 yy xx reg yy <- reg yy - reg xx (xx ≠ yy)`
Subtracts the right register from the left register and places the result in the leftregister.

`MOV SP RG NZ 101 0 xx xx SP <- reg xx`
Moves/copies the contents of the register on the right into the stack pointer, thereby resetting it new.

`SUB RG [RG] NZVC 101 1 yy xx reg yy <- reg yy - Adr(reg xx)`
Subtracts from the left register the byte whose address is in the right register. Result then in the left register.

`MOV [RG] RG NZ 110 0 yy xx MOVe Adr(reg yy) <- reg xx`
Moves/copies the content of the right register to the memory address in the left register.

`MOV RG [RG] NZ 110 1 yy xx MOVe reg yy <- Adr(reg xx)` *)
Moves/copies the content of the memory address in the right register to the left register.

`MOV RG RG NZ 111 0 yy xx MOVe reg yy <- reg xx (xx ≠ yy)`
Moves/copies the register on the right to the register on the left.

`MOV SR CC 111 0 CC CC MOVe SR <- CC.. (CC=CC)`
The constant bits CC are set in the left two bits of the status register. This defines the data type `IO` for input and output: `00` character, `01` decimal, `10` hexadecimal, `11` binary

`MOV RG IA NZ 111 1 00 xx MOVe reg xx <- Adr(A0+A1+ST)`
Moves/copies the byte from the calculated index address `A0+A1+ST` into the register.
The address is determined cyclically at <0 or >255.

`MOV IA RG NZ 111 1 01 xx MOVe Adr(A0+A1+ST)<- reg xx`
Moves/copies the register value into RAM to the calculated index address `A0+A1+ST`.
The address is determined cyclically at <0 or >255.

`LOD RG NZ 111 1 10 xx LOad Data reg xx <- Adr(h80+A1+ST)`
Loads/copies the data byte from the calculated index address `h80+A1+ST` into the register.
Base address data is `h80`, the address is determined cyclically at <0 or >255, `A1` with sign.

STO (IO) Z 111 1 11 00 STOre file <- Adr(h80+A1+ST) D0 Byte, via D1 (Data: IO)

Writes D0 byte (unsigned) from address h80+A1+ST into a text file.
The address is determined cyclically at <0 or >255.
The bytes are first brought to D1. One value is stored per line.
Depending on IO is written at 00 character, 01 decimal, 10 hexadecimal, 11 binary.
The Z flag is set if no data was written. The number of bytes actually written is then unsigned in D0, the last byte in D1.

RCL (IO) ZV 111 1 11 01 ReCalL Adr(h80+A1+ST) <- file via D1 (Data: IO) /D0 byte readed

Reads data from a text file into the RAM from address h80+A1+ST.
The address is determined cyclically at <0 or >255.
The bytes are first brought to D1. There must be one value per line.
Depending on IO, at 00 character, 01 decimal, 10 hexadecimal, 11 binary is assumed as type in the file.
The Z flag is set if no data was read. The V flag is set when there is more data in the file than can be stored to the end of RAM. The number of bytes actually read is then written unsigned in D0 , the last byte in D1.

CPY ZV 111 1 11 10 CoPY Adr(A1)<-Adr(A0) D0 byte via D1

Copies D0 bytes (unsigned) from RAM address in A0 starting into the area from RAM address in A1.
The Z flag is set if no data has been copied.
The V flag is set if more data is to be copied than can be stored to the end of RAM.
The number of bytes actually copied is then written unsigned in D0 , the last byte in D1.

STP (pre-cmd) 111 1 11 11 StoP STop Program, flags from pre-command

Halts the program. The flags are taken over from the preceding command. This allows STP to be inserted at any point in the program in addition to the breakpoints for test purposes.