```
;Assembler Befehlsübersicht
; SUNriaX Technology!
; (c) 2016 Raffael GÄCHTER

;Wichtige Werte:
;-----------------------------------------------------------------
Dez | Hex   | Binär
0   | 0x00  | 0b00000000
127 | 0x7F  | 0b01111111
128 | 0x80  | 0b10000000
255 | 0xFF  | 0b11111111
;-----------------------------------------------------------------

;Register:
;-----------------------------------------------------------------
  Systemregister  r0  - 15
  Arbeitsregister r16 - 327
;-----------------------------------------------------------------

;Statusregister SREG (SRAM Adresse $5F, SFR-Adresse $3F)
; +-------------+-----------+-----------+-------------+-------------+-------------+-------+-----------+
; | Bit7        | Bit6      | Bit5      | Bit4        | Bit3        | Bit2        | Bit1  | Bit0      |
; |   I         |   T       |   H       |   S         |   V         | N           |   Z   |   C       |
; | Interrupt   | Transfer- | Halfcarry | Sign        | Overflow    | Negative    | Zero  | Carry     |
; | 1: Frei     | Register  | BCD-      | N XOR V     | Überlauf    | Vorzeigen   | Null  | Überlauf  |
; | 0: gesperrt | Bits      | Korrektur | signed Zahl | signed Zahl | signed Zahl |       | Übertrag  |
; +-------------+-----------+-----------+-------------+-------------+-------------+-------+-----------+
;-----------------------------------------------------------------

;Anweisungen:
;-----------------------------------------------------------------
  .include    Dateineinbindung              .include "m16.inc"
  .device     Bausteintyp                   .device ATmega16
  .list       Übersetzungsliste ein
  .nolist     Übersetzubgsliste aus
  .macro      Makrobeginn                   .MACRO division
                                              addition @0, +@1
  .endmacro   Makroende
  .endm       (abkürzung)
  .listmac    Makroerweiterung in Liste

  .def        Register Bezeichner           .def temp = r16
  .equ        Konstante (unveränderlich)    .equ takt = 16000000
  .set        veränderliche Definition      .set wert = 123 ;alt
                                            .set wert = 99  ;neu
  .org        Adresszähler (Interrupt)      .org $2A
  .exit       Ende des Quellcodes
;-----------------------------------------------------------------

;Speicherbereiche:
;-----------------------------------------------------------------
  .CSEG       FLASH (Programmbereich)
  .DSEG       SRAM-Bereich (Variablen)
  .ESEG       EEPROM-Bereich (Konstanten)

  .DB         8-Bit Werte (vorgeladen im FLASH/EEPROM),
              Bytekonstanten
                k1: .DB 1,2,3,4 ;Zahlen
                    .DB 'a','b' ;Zeichen

  .DW         16-Bit Werte (vorgeladen im FLASH/EEPROM),
              Wortkonstanten
                k2: .DW k1      ;Adresse
                    .DW 1234    ;Zahl

  .BYTE       Reservierung von n BYTES im SRAM/EEPROM
                k3: .BYTE 8   ;10 Bytes
;-----------------------------------------------------------------

;Operanden
;-----------------------------------------------------------------
  loop: rjmp loop       ;Schleifenkonstrukt
  rjmp PC               ;aktueller Adresszähler
  rcall loop            ;Schleifenkonstrukt mit Rücksprung
                        ;zur Aussprungadresse

  ldi r16, 123          ;Lade Register Dezimal
  ldi r17, $FF          ;Lade Register Hex
  ldi r18, 0b01010101   ;Lade Register Binär
  ldi r19, 010          ;Lade Register Oktal
  ldi r29, 'U'          ;Lade Register Zeichen ($55)

  text: .DB "Hallo"     ;Lade String
;-----------------------------------------------------------------
```

```
;Operatoren
;-----------------------------------------------------------------
  arithmetisch: -         ;2er Komplement (ldi r16,-2)
  (ganzzalig)   + - *     ;nur ganzzahlige Operationen
                /         ;Quotient Ganzzahlig, kein Rest

  bitoperation: ~         ;1er Komplement         ldi r16,~$A1
                & | ^     ;UND/ODER/EODER         ldi r17,x & $F1
                << >>     ;links/rechts schieben  ldi r18,x << 1

  vergleich:    < <= ==   ;ergibt 0 bei nein,     ldi r19,x == y
                != >= >   ;1 bei ja

  logische      !         ;ergibt 1 bei Wert 0    ldi r20,!$C4
  Verknüpfung:  && ||     ;0/1 je nach UND/ODER   ldi r21,a && b
;-----------------------------------------------------------------

;Funktionen
;-----------------------------------------------------------------
  LOW(cmd)  ;liefert Bit 0-7 Low-Byte   LOW($12345678)    ($78)
  HIGH(cmd) ;liefert Bit 8-15 High-Byte HIGH($12345678)   ($56)
  PAGE(cmd) ;liefert Bit16-21           PAGE($45678)      ($4 )

  BYTE2(cmd);liefert Bit 8-15 High-Byte BYTE2($12345678)  ($56)
  BYTE3(cmd);liefert Bit15-23           BYTE3($12345678)  ($34)
  BYTE4(cmd);liefert Bit24-31           BYTE4($12345678)  ($5678)

  LWRD(cmd) ;liefert Bit 0-15 Low-Word  LWRD($12345678)   ($5678)
  HWRD(cmd) ;liefert Bit16-31 High-Word HWRD($12345678)   ($1234)

  EXP2(cmd) ;liefert 2^AUSDRUCK         EXP2(4)          (2^4=16)
  LOG2(cmd) ;liefert log2 ganzzahlig    LOG2(17)=log2(17)=4.09=4
;-----------------------------------------------------------------

;Direktiven
;-----------------------------------------------------------------
  .IF         .IF RAMEND > 255              ;IF Anweisung
                  ldi r16,HIGH(RAMEND)
                  out SPH,akku
              .ENDIF

  .ELIF       else if Anweisung             ;ELSEIF Anweisung

  .IFDEF      .IFDEF SPH                    ;IFDEFINED Anweisung
                  ldi r16,HIGH(RAMEND)
                  out SPH,akku
              .ENDIF

  .IFNDEF     .IFNDEF SPH                   ;IFNOTDEFINED Anweisung
                .MESSAGE "kein SPH"
              .ENDIF

  .ELSE       beendet Ja Zweig, beginnt Nein Zweig
  .ENDIF      beendet Zweig
  .MESSAGE    Meldung Ausgeben
  .ERROR      Fehler Ausgeben Assemblierung anhalten
;-----------------------------------------------------------------

;Byteoperationen
;-----------------W-T-------------------------------------------
  mov   r16,r17   1 1 ;Verschiebe Register r16 mit r17
                      ;r16 <= r17
  swap  r16       1 1 ;Vertausche Halbbytes von r16
                      ;r16(7-4) <=> r16(3-0)
  ldi   r16,k8    1 1 ;Lade r16 mit Bytekonstante (nur r16-r31)
                      ;r16 <= k8
  in    r16,SFR   1 1 ;lade r16 mit SFR-Register
                      ;r16 <= SFR
  out   SFR,r16   1 1 ;lade SFR-Register mit r16
                      ;SFR <= r16
  lds   r16,adr   2 3 ;lade r16 direkt mit Byte aus SRAM
                      ;r16 <= SRAM
  sts   adr, r16  2 3 ;lade Byte im SRAM direkt mit r16
                      ;SRAM <= r16
  nop             1 1 ;NO-OPERATION

; *)W=Wörter
; *)T=Takte
;-----------------------------------------------------------------

;Arithmetische Befehle
;-------------------------W-T-------------------------------------
  add   r16,r17 (HSVNZC)  1 1 ;Addition zweier Register
                              ;r16 <= r16+r17
  adc   r16,r17 (HSVNZC)  1 1 ;Addition zweier Register und Carry
```

```
                              ;r16 <= r16+r17+C
  subi  r16,-k8 (------)  1 1 ;Addiere zum Register eine Konstante
                              ;r16 <= r16+kon
  sub   r16,r17 (HSVNZC)  1 1 ;Substraktion zweier Register
                              ;r16 <= r16-r17
  sbc   r16,r17 (HSVN-C)  1 1 ;Substraktion zweier Register und Carry
                              ;r16 <= r16-r17-C
  subi  r16,k8  (HSVNZC)  1 1 ;Subtrahiere vom Register eine Konstante
                              ;r16 <= r16-kon
  sbci  r16,k8  (HSVN-C)  1 1 ;Substraktion vom Register Konstante und C
                              ;r16 <= r16-k8-C
  neg   r16     (HSVNZC)  1 1 ;negiere Register (2er-Komplement)
                              ;r16 <= $00-r16
  cp    r16,r17 (HSVNZC)  1 1 ;Vergleiche zwei Register
                              ;r16-r17
  cpc   r16,r17 (HSVN*C)  1 1 ;Vergleiche zwei Register und Carry
                              ;r16-r17-C
  cpi   r16,k8  (HSVNZC)  1 1 ;Vergleiche Register mit Konstante

;*)ITHSVNZC = SREG (Status Register)
;*)SFR = Special Function Register
;---------------------------------------------------------------

;Pseudobefehle
;-------------------------W-T-----------------------------------
  clr r16                 1 1 ;löscht alle Bits im Register
                              ;r16 <= $00
  ser r16                 1 1 ;setzt alle Bits im Register
                              ;r16 <= $FF
  tst r16                 1 1 ;teste das Register auf Nulll und
                              ;Vorzeichen (nur signed Zahlen)
;---------------------------------------------------------------

;Zählebefehle
;-------------------------W-T-----------------------------------
  inc r16         (SVNZ)  1 1 ;inkrementiere Register
                              ;r16 <= r16+1
  dec r16         (SVNZ)  1 1 ;dektrementiere Register
                              ;r16 <= r16+1
;---------------------------------------------------------------

;logische Befehle
;-------------------------W-T-----------------------------------
  com   r16       (S0NZ)  1 1 ;komplementiere Bits (1er Komp.)
                              ;r16 <= NICHT r16
  and   r16,r17   (S0NZ)  1 1 ;logisches UND aller Bits
                              ;r16 <= r16 AND r17
  or    r16,r17   (S0NZ)  1 1 ;logisches ODER aller Bits
                              ;r16 <= r16 OR r17
  eor   r16,r17   (S0NZ)  1 1 ;logisches EODER aller Bits
                              ;r16 <= r16 EOR r17
  andi  r16,k8    (S0NZ)  1 1 ;UND mit einer konstanten Maske
                              ;r16 <= r16 UND k8
  ori   r16,k8    (S0NZ)  1 1 ;ODER mit einer konstanten Maske
                              ;r16 <= r16 ODER k8
  cbr   r16,k8    (S0NZ)  1 1 ;löscht Bit wennBit in k8=1
                              ;andi r16, ($FF-k8)
  sbr   r16,k8    (S0NZ)  1 1 ;setze Bit wennBit in k8=1
                              ;ori r16, k8
;---------------------------------------------------------------

;Schiebefehle
;-------------------------W-T-----------------------------------
  asr   r16       (SVNZC) 1 1 ;schiebe arithmetisch nach rechts
                              ;b7 -> [b7 >> b0] -> C
  lsr   r16       (SVNZC) 1 1 ;schiebe logisch nach rechts
                              ;0 -> [b7 >> b0] -> C
  ror   r16       (SVNZC) 1 1 ;rotiere rechts durch das Carry Bit
                              ;C -> [b7 >> b0] -> C
  lsl   r16     (HSVNZC)  1 1 ;schiebe logisch nach kinks
                              ;C <- [b7 << b0] <- 0
  rol   r16     (HSVNZC)  1 1 ;rotiere links durch das Carry Bit
                              ;C <- [b7 << b0] <- C
;---------------------------------------------------------------

;Arbeitsregister (Bit)Befehle
;-------------------------W-T-----------------------------------
  bld   r16, bit          1 1 ;lade Registerbit mit T-Bit
                              ;r16(bit) <= T
  bst   r16, bit    (T)   1 1 ;speichere Registerbit nach T-Bit
                              ;T <= r16(bit)
  sbrc  r16, bit          1 1 ;überspringe nächsten Befehl, wenn
                              ; Bit im Register gleich 0
  sbrs  r16, bit          1 1 ;überspringe nächsten Befehl, wenn
                              ; Bit im Register gleich 1
```

```
;---------------------------------------------------------------

;Statusregister (Bit)Befehle
;-------------------------W-T-----------------------------------
  bclr  bitpos            1 1 ;lösche Bitposition im SREG
                              ;SREG(bit) <= 0
  bset  bitpos            1 1 ;setze Bitposition im SREG
                              ;SREG(bit) <= 1

  clc                     1 1 ;C <= 0 lösche Übertragsbit
  sec                     1 1 ;C <= 1 setze Übertragsbit
  clz                     1 1 ;Z <= 0 lösche Nullanzeigebit
  sez                     1 1 ;Z <= 1 setze Nullanzeigebit
  cln                     1 1 ;N <= 0 lösche Negativanzeigebit
  sen                     1 1 ;N <= 1 setze Negativanzeigebit
  clv                     1 1 ;V <= 0 lösche Überlaufbit
  sev                     1 1 ;V <= 1 setze Überlaufbit
  cls                     1 1 ;S <= 0 lösche Vorzeichenbit
  ses                     1 1 ;S <= 1 setze Vorzeichenbit
  clh                     1 1 ;H <= 0 lösche Halbübertrag
  seh                     1 1 ;H <= 1 setze Halbübertrag
  clt                     1 1 ;T <= 0 lösche Transferbit
  set                     1 1 ;T <= 1 setze Transferbit

  cli                     1 1 ;C <= 0 lösche Interruptbit
                              ;Interrupts sperren
  sei                     1 1 ;C <= 1 setze Interruptbit
                              ;Interrupts freigeben
;---------------------------------------------------------------

;Wortoperationen
;---------------------------------------------------------------
  ldi r24,LOW($1234)    ;R24 <- $34 Low-Teil der Konstante
  ldi r25,HIGH($1234)   ;R25 <- $12 High-Teil der Konstante
  ;+------WORTREGISTER------------+----------LOW-BYTE---------+--------HIGH-BYTE------+
  ;|R24   kein Indexregister      | R24       Adresse = 24    | R25     Adresse = 25  |
  ;|R26 = XL = Indexregister X    | R26 = XL  Adresse = 26    | R27=XH  Adresse = 27  |
  ;|R28 = YL = Indexregister Y    | R28 = YL  Adresse = 28    | R29=YH  Adresse = 29  |
  ;|R30 = ZL = Indexregister Z    | R30 = ZL  Adresse = 30    | R31=ZH  Adresse = 31  |
  ;+------------------------------+---------------------------+-----------------------+

;---------------------------------------------------------------

;Wortbefehle
;-------------------------W-T-----------------------------------
  adiw  Rw,k6     (SVNZC) 1 2 ;Addiere Konstante 0 bis 63
                              ; nur r24,r26,r28,r30
                              ;Rw <= Rw+6bit Konst.
  sbiw  Rw,k6     (SVNZC) 1 2 ;Subtrahiere Konstante 0 bis 63
                              ; nur r24,r26,r28,r30
                              ;Rw <= Rw-6bit Konst.
  movw  r16,r17   (SVNZC) 1 1 ;kopiere 16bit Wort (nur ATmega)
                              ;Rd:Rd+1 <= Rr:Rr+1
                              ;d und r geradzahlig
;---------------------------------------------------------------

;Operation mit SFR-Registern
;-----------------W-T-------------------------------------------
  cbi   port,bit  1 2 ;lösche Port Bit
                      ;port(bit) <= 0
  sbi   port,bit  1 2 ;setze Port Bit
                      ;port(bit) <= 1
  sbic  port,bit  1 1 ;überspringe den nächsten Befehl,
                      ;wenn Bit im Port = 0
  sbis  port,bit  1 1 ;überspringe den nächsten Befehl,
                      ;wenn Bit im Port = 1
;---------------------------------------------------------------

;Multiplikationsbefehle Mega-Familie
;-------------------------W-T-----------------------------------
  mul   r16,r17     (ZC)  1 2 ;Vorzeichenlose Multiplikation
                              ;R1:R0 <- r16*17
                              ;alle Register R0 bis R31
  muls  r16,r17     (ZC)  1 2 ;Vorzeichenbehaftete Multiplikation
                              ;R1:R0 <- r16*17
                              ;nur Register R16 bis R31
  mulsu r16,r17     (ZC)  1 2 ;Vorzeichen -behaftete * -lose
                              ;R1:R0 <- r16*17
                              ;alle Register R16 bis R23
  movw  r16,r18           1 1 ;kopiert 16bit Wort alternativ auch
                              ;movw R16+1:R16, R17+1:R17
                              ;Register jeweils geradzahlig
;---------------------------------------------------------------

;Unbedingte Sprungbefehle
```

```
;-----------------W-T-----------------------------------------
  rjmp  Ziel      1 2 ;springe unbedingt relativ
                      ;PC <- PC + Abstand
  rcall Ziel      1 3 ;rufe Unterprogramm relativ
                      ;Stapel <- PC, SP <- SP - 2
                      ;PC <- PC Abstand
  ijmp            1 2 ;PC <- Z  springe unbedingt indirekt
  icall           1 3 ;rufe Unterprogramm indirekt
                      ;Stapel <- PC, SP <- SP -2

  ; Controler mit mehr als 8 kByte Programm-Flash (Mega-Familie)
  ; sind zusätzliche Befehle erforderlich

  jmp   Ziel      2 3 ;springe unbedingt direkt
                      ;PC <- Zieladresse
  call  Ziel      2 4 ;rufe Unterprogramm direkt
                      ;Stapel <- PC, SP <- SP-2
                      ;PC <- Zieladresse
  ret             1 4 ;Rücksprung aus Unterprogramm
                      ;SP <- SP+2
                      ;PC <- Stapel
;-------------------------------------------------------------

;Bedingte Sprungbefehle
;-----------------W-T-----------------------------------------
  brbc  bit,ziel  1   ;verzweige relativ für Bit = 0
                  2 ;ja:    PC <= PC + Abstand
                  1 ;nein:  PC <= PC + 1

  brbs  bit,ziel  1   ;verzweige relativ für Bit = 1
                  2 ;ja:    PC <= PC + Abstand
                  1 ;nein:  PC <= PC + 1

  brcc  ziel      1 1/2 ;Sprung bei C=0 unsigned
  brcs  ziel      1 1/2 ;Sprung bei C=1 unsigned
  brsh  ziel      1 1/2 ;Sprung bei >= unsigned
  brlo  ziel      1 1/2 ;Sprung bei <  unsigned
  brne  ziel      1 1/2 ;Sprung bei Z=0 ungleich
  breq  ziel      1 1/2 ;Sprung bei Z=1 gleich
  brpl  ziel      1 1/2 ;Sprung bei N=0 signed
  brmi  ziel      1 1/2 ;Sprung bei N=1 signed
  brvc  ziel      1 1/2 ;Sprung bei V=0 signed
  brvs  ziel      1 1/2 ;Sprung bei V=1 signed
  brge  ziel      1 1/2 ;Sprung bei >= signed
  brlt  ziel      1 1/2 ;Sprung bei <  signed
  brhc  ziel      1 1/2 ;Sprung bei H=0
  brhs  ziel      1 1/2 ;Sprung bei H=1
  brtc  ziel      1 1/2 ;Sprung bei T=0
  brts  ziel      1 1/2 ;Sprung bei T=1
  brid  ziel      1 1/2 ;Sprung bei I=0 gesperrt
  brie  ziel      1 1/2 ;Sprung bei I=1 frei

;Veränderung der Sprungbedingungen
;------------------------W-T-----------------------------------
  cp    r16,r17 (HSVNZC)  1 1 ;Vergleiche zwei Register durch
                              ;Testsubtraktion
                              ;r16-r17
  cpc   r16,r17 (HSVN*C)  1 1 ;Vergleiche Register und Carry
                              ;r16-r17-C
  cpi   r16,k8  (HSVNZC)  1 1 ;Vergleiche Register mit Konstante
  tst r16                 1 1 ;teste Register auf Null und
                              ;Vorzeichen (and r16, r16)
;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
;Bedingte relative Sprungbefehle haben wegen des 7bit Abstands nur
;einen eingeschränkten Sprungbereich von +63 und -64 Befehlen. Liegt
;der nächste Einsprungpunkt über diesem Limit so wird die maximale
;Sprungweite überschritten und der Assembler meldet einen Fehler. Die
;Logik muss umgekehrt werden.
;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Beispiel
*FALSCH
    in temp,PIND
    andi temp,0b00000011
    cpi  temp,0b00000011
    breq ende
  ;Assembler meldet Fehler, wenn ende weiter als 63 Befehle entfernt
  ;Eine umkehrung der Logik ist notwendig
  ende:

*RICHTIG
    in temp,PIND
    andi temp,0b00000011
    cpi  temp,0b00000011
    brne ersatz           ;springe wenn beide ungleich
```

```
      rjmp ende             ;springe bei gleich immer nach ende
    ersatz:
      ;ende kann beliebig weit entfernt sein
    ende:
;---------------------------------------------------------------

;Bedingte Skipbefehle
;-----------------W-T-----------------------------------------
  cpse  r16,r17   1 1 ;überspringe nächsten Befehl wenn r16 = r17
                      ;r16-r17
  sbrc  r16,bit   1 1 ;überspringe nächsten Befehl wenn r16(bit)=0
                      ;testeBit in Register r16
  sbrs  r16,bit   1 1 ;überspringe nächsten Befehl wenn r16(bit)=0
                      ;testeBit in Register r16
  sbic  port,bit  1 1 ;überspringe nächsten Befehl wenn Port-Bit=0
                      ;testeBit in Port
  sbis  port,bit  1 1 ;überspringe nächsten Befehl wenn Port-Bit=1
                      ;testeBit in Port
;---------------------------------------------------------------

;Adressierung von Konstanten im Flash
;-----------------W-T-----------------------------------------
  lpm             1 3 ;lade r0 mit Flash (alle AT-Familien)
                      ;r0 <- (Z)
  lpm   r16,Z     1 3 ;lade r16 mit Flash (nur ATmega)
                      ;r16 <- (Z)
  lpm   r16,Z+    1 3 ;lade r16 mit Flash erhöhe Z (nur ATmega)
                      ;r16 <- (Z); Z <- Z+1
  spm             1 - ;speichere Boot-Bereich (nur ATmega)
                      ;(Z) <- R1:R0

  Beispiel
    ldi ZL,LOW(konstante *2)  ;Z <- Adresse der Konstante *2
    ldi ZH,HIGH(konstante *2) ;
    lpm                       ;R0 <- Byte durch Z adressiert
    mov akku,r0               ;akku <- Byte aus R0

    ;Konstantenbereich hinter Befehlen
    konstante: .DB 123      ;eine Dezimale Bytekonstanten
;---------------------------------------------------------------

;Adressierung von Variablen im SRAM
;-----------------W-T-----------------------------------------
  lds r16,addr    2 3   ;lade r16 direkt mit Byte aus SRAM
                        ;r16 <- SRAM
  sts addr,r16    2 3   ;lade Byte im SRAM direkt mit r16
                        ;SRAM <- r16

  ;indirekte Adressierung
  ld  r16,X       1 2   ;lade r16 indirekt mit Byte aus SRAM
                        ;r16 <- (X)
  ld  r16,Y       1 2   ;lade r16 indirekt mit Byte aus SRAM
                        ;r16 <- (Y)
  ld  r16,Z       1 2   ;lade r16 indirekt mit Byte aus SRAM
                        ;r16 <- (Z)
  st  X,r16       1 2   ;speichert r16 indirekt nach SRAM
                        ;(X) <- r16
  st  Y,r16       1 2   ;speichert r16 indirekt nach SRAM
                        ;(Y) <- r16
st  Z,r16       1 2   ;speichert r16 indirekt nach SRAM
                      ;(Z) <- r16

;automatische erhöhung des Wortregisters nach Adressierung
ld  r16,X+      1 2   ;lade r16 indirekt mit Byte aus SRAM
                      ;r16 <- (X); X <- X+1
ld  r16,Y+      1 2   ;lade r16 indirekt mit Byte aus SRAM
                      ;r16 <- (Y); Y <- Y+1
ld  r16,Z+      1 2   ;lade r16 indirekt mit Byte aus SRAM
                      ;r16 <- (Z); Z <- Z+1
st  X+,r16      1 2   ;speichert r16 indirekt nach SRAM
                      ;(X) <- r16; X <- X+1
st  Y+,r16      1 2   ;speichert r16 indirekt nach SRAM
                      ;(Y) <- r16; Y <- Y+1
st  Z+,r16      1 2   ;speichert r16 indirekt nach SRAM
                      ;(Z) <- r16; Z <- Z+1

;automatische verminderung des Wortregisters nach Adressierung
ld  r16,X-      1 2   ;lade r16 indirekt mit Byte aus SRAM
                      ;r16 <- (X); X <- X-1
ld  r16,Y-      1 2   ;lade r16 indirekt mit Byte aus SRAM
                      ;r16 <- (Y); Y <- Y-1
ld  r16,Z-      1 2   ;lade r16 indirekt mit Byte aus SRAM
                      ;r16 <- (Z); Z <- Z-1
st  X-,r16      1 2   ;speichert r16 indirekt nach SRAM
                      ;(X) <- r16; X <- X-1
```

```
st  Y-,r16      1 2   ;speichert r16 indirekt nach SRAM
                      ;(Y) <- r16; Y <- Y-1
st  Z-,r16      1 2   ;speichert r16 indirekt nach SRAM
                      ;(Z) <- r16; Z <- Z-1

;indirekte Adressierung mit einem konstanten Abstand
ldd r16,Y+k6    1 2   ;lade r16 indirekt mit Byte aus SRAM
                      ;r16 <- (Y+konstante)
ldd r16,Z+k6    1 2   ;lade r16 indirekt mit Byte aus SRAM
                      ;r16 <- (Z+konstante)
std Y+k6,r16    1 2   ;speichert r16 indirekt nach SRAM
                      ;(Y+konstante) <- r16
std Z+k6,r16    1 2   ;speichert r16 indirekt nach SRAM
                      ;(Z+konstante) <- r16

;Stapelbefehle (sonderform indirekter Adressierung)
push  r16       1 2   ;speichert r16 auf den Stapel
                      ;(SP) <-r16; SP <- SP-1
pop   r16       1 2   ;lade r16 vom Sapel
                      ;SP <- SP+1; r16 <- (SP)

Beispiel
  lds temp,zaehl      ;temp <- alter Wert aus SRAM
  inc zaehl           ;Zähler um Arbeitsregister +1
  sts zaehl,temp      ;SRAM <- temp neuer Wert

  .DSEG               ;Datensegment mit Variablen
  zaehl .BYTE 1       ;eine Bytevariable reservieren
;---------------------------------------------------------------

;Adressierung von Daten im EEPROM
;---------------------------------------------------------------
  .ESEG   ;nichtflüchtige Variable und Konstanten
  .DB     ;Bytekonstantenliste  bk:   .DB 1,2,3,4
  .DW     ;Wortkonstante        wk:   .DW bk,2342
  .BYTE   ;Anzahl der Bytes     var: .BYTE 100

  ;EECR EEPROM Control Register (SRAM Adresse $3C SFR-Adresse $1C)
  ; +-----+-----+-----+-----+-----------------+-----------------+-------------------+---------------+
  ; | Bit7| Bit6| Bit5| Bit4| Bit3            | Bit2            | Bit1              | Bit0          |
  ; | -   | -   | -   | S   | EERIE           | EEMWE           | EEWE              | EERE          |
  ; |     |     |     |     | 0:kein Interrupt| 0:Schreibsperre | 0:nicht schreiben | 0:kein lesen  |
  ; |     |     |     |     | 1:Interrupt frei| 1:Schreiben frei| 1:schreiben       | 1:lesen       |
  ; |     |     |     |     |                 |                 | X:nach Reset      |               |
  ; +-----+-----+-----+-----+-----------------+-----------------+-------------------+---------------+

  Beispiel
    write_prom: sbic  EECR,EEWE   ;warte bis Schreibzugriff beendet
                rjmp write_prom

                .IFDEF  EEARH     ;für EEPROM >= 256 Bytes
                  out EEARH,ZH    ;Adresse High
                  out EEARL,ZL    ;Adresse Low
                .ENDIF
                .IFDEF  EEAR      ;für EEPROM < 256 Bytes
                  out EEAR,ZL     ;nur Adresse Low
                .ENDIF

                out EEDR, r16     ;Daten nach Schreibregister
                sbi EECR,EEMWE    ;Start des Schreibzugriffs
                sbi EECR,EEWE     ;Start des Schreibvorgangs
      ret                         ;Rücksprung

    read_prom:  sbic  EECR, EEWE  ;warte bis Schreibzugriff beendet
                rjmp read_prom

                .IFDEF  EEARH     ;für EEPROM >= 256 Bytes
                  out EEARH,ZH    ;Adresse High
                  out EEARL,ZL    ;Adresse Low
                .ENDIF
                .IFDEF EEAR
                  out EEAR,ZL
                .ENDIF

                sbi EECR,EERE     ;setze Lesebit
                in  r16,EEDR      ;Daten lesen
      ret                         ;Rücksprung
;---------------------------------------------------------------

;Interrupts
;-------------------------W-T-----------------------------------
  sei             1       1 1 ;alle Interrupts global freigeben
  cli             0       1 1 ;alle Interrupts global sperren
  reti            1       1 4 ;Rücksprung aus Serviceprogramm
                              ;alle Interrupts global freigeben
```

```
                              ;SP <- SP+2; PC <- Stapel 1 < -1

  in    r16,SREG          1 1 ;Statusregister (auch I-Bit) kopieren
                              ;r16 <- Statusbits
  out   SREG,r16          1 1 ;Statusregister (auch I-Bit) laden
                              ;Statusbits <- r16

  brid  Ziel              1 2 ;verzweige bei I=0 Interrupts gesperrt
  brie  Ziel              1 2 ;verzweige bei I=1 Interrupts frei

  sleep                   1 1 ;Ruhezustand für SE=1
  wdr                     1 1 ;Watchdog Timer zurücksetzen

  ;Einsprungpunkte Atmega16
  ;Adresse  Bezeichner  Auslösung
  $000                  Reset
  $002      INT0addr    externer Interrupt0
  $004      INT1addr    externer Interrupt1
  $006      OC2addr     Timer2 Compare Ausgang
  $008      OVF2addr    Timer2 Überlauf
  $00A      ICP1addr    Timer1 Capture Eingang
  $00C      OC1Aaddr    Timer1 Compare-Ausgang A
  $00E      OC1Baddr    Timer1 Compare-Ausgang B
  $010      OVF1addr    Timer1 Überlauf
  $012      OVF0addr    Timer0 Überlauf
  $014      SPIaddr     SPI-Schnittstelle
  $016      URXCaddr    USART Zeichen empfangen
  $018      UDREaddr    USART Datenregister leer
  $01A      UTXCaddr    USART Zeichen gesendet
  $01C      ADCCaddr    A/D-Wandlung fertig
  $01E      ERDYaddr    EEPROM Schreiboperation fertig
  $020      ACIaddr     Analogkomperator
  $022      TWIaddr     TWI-Schnittstelle
  $024      INT2addr    externer Interrupt2
  $026      OC0addr     Timer0 Compare Ausgang
  $028      SPMaddr     Flash-Programmierung fertig
  $02A      Ende der Einsprungtabelle erster Befehl des Programms
;-----------------------------------------------------------------

;Externe Interrupts
;-----------------------------------------------------------------
  ;MCUCR=MCU Control Register (SRAM Adresse $55 SFR-Adresse $35)
  ; +-----------+-----------+-------------+-------------+-----------+-------------+-----------+-------------+
  ; | Bit7      | Bit6      | Bit5        | Bit4        | Bit3      | Bit2        | Bit1      | Bit0        |
  ; | SM2       | SE        | SM1         | SM0         | ISC11     | ISC10       | ISC01     | ISC00       |
  ; | Sleep     | Sleep     | Sleep       | Sleep       | externer Interrupt INT1 | externer Interrupt INT0 |
  ; | Betriebs- | Betrieb   | Betriebsart | Betriebsart | 00:durch Low-Zustand    | 00:durch Low-Zustand    |
  ; | art       | 0:gesperrt|             |             | 01:Zustandsänderung*    | 01:Zustandsänderung*    |
  ; |           | 1:frei    |             |             | 10:fallende Flanke      | 10:fallende Flanke      |
  ; |           |           |             |             | 11:steigende Flanke     | 11:steigende Flanke     |
  ; +-----------+-----------+-------------+-------------+-----------+-------------+-----------+-------------+
  ;*)nicht bei allen Controllertypen vorhanden

  ;MCUCSR=MCU Control und Status Register (SRAM Adresse $54 SFR-Adresse $34)
  ; +-----------+-------------+-------------+-------------+-----------+-------------+-----------+-------------+
  ; | Bit7      | Bit6        | Bit5        | Bit4        | Bit3      | Bit2        | Bit1      | Bit0        |
  ; | JTD       | ISC2        |   -         | JTRF        | WDRF      | BORF        | EXTRF     | PORF        |
  ; |           | INT2 Flanke |             |             |           |             |           |             |
  ; |           | 0:fallend   |             |             |           |             |           |             |
  ; |           | 1:steigend  |             |             |           |             |           |             |
  ; +-----------+-------------+-------------+-------------+-----------+-------------+-----------+-------------+

  ;GICR=General Interrupt Control Register (SRAM-Adresse $5B SFR-Adresse $3B)
  ; +-------------+-------------+-------------+-------------+-----------+-------------+-----------+-------------+
  ; | Bit7        | Bit6        | Bit5        | Bit4        | Bit3      | Bit2        | Bit1      | Bit0        |
  ; | INT1        | INT0        | INT2        | -           | -         | -           | IVSEL     | IVCE        |
  ; | 0:gesperrt  | 0:gesperrt  | 0:gesperrt  |             |           |             |           |             |
  ; | 1:frei      | 1:frei      | 1:frei      |             |           |             |           |             |
  ; +-------------+-------------+-------------+-------------+-----------+-------------+-----------+-------------+

  ;GIFR=General Interrupt Flag Register (SRAM-Adresse $5A SFR-Adresse $3A)
  ; +-----------------+-----------------+-----------------+-------------+-----------+-------------+-----------+-------------+
  ; | Bit7            | Bit6            | Bit5            | Bit4        | Bit3      | Bit2        | Bit1      | Bit0        |
  ; | INTF1           | INTF0           | INTF2           | -           | -         | -           |           |             |
  ; | 0:n. anstehend  | 0:n. anstehend  | 0:n. anstehend  |             |           |             |           |             |
  ; | 1:anstehend     | 1:anstehend     | 1:anstehend     |             |           |             |           |             |
  ; +-----------------+-----------------+-----------------+-------------+-----------+-------------+-----------+-------------+
;-----------------------------------------------------------------

;Stack
;-----------------------------------------------------------------
  ;Stackpointer initialisieren
    ldi r16, high(RAMEND)
    out SPH, r16
    ldi r16, low(RAMEND)
```

```
    out SPL, r16
```