

1.1 Der Befehlssatz der AVR-RISC-Controller

Die Assemblerbefehle haben folgende Strukturen:

Befehl		; ohne Operanden
Befehl	Operand	; ein Operand
Befehl	Ziel,Quelle	; zwei Operanden
Befehl	Register,Bit	; zwei Operanden

In den Befehlslisten erscheinen folgende Abkürzungen:

Rd = Zielregister Rr = Quellregister K8 = 8bit Konstante K6 = 6bit Konstante
K16 = 16bit Adresskonstante

Das 16bit Befehlswort `XXXX XXXX XXXX XXXX` hat folgenden Aufbau:

d = Zielregister **r** = Quellregister **K** = Konstante **k** = Adresskonstante
A = SFR-Adresse **b** = Bit in Register **s** = Bit in SREG

Das Statusregister SREG enthält folgende Bitpositionen:

I	T	H	S	V	N	Z	C
Interrupt	Transfer	Halfcarry	Sign	oVerflow	Negative	Zero	Carry

I = globale Interruptfreigabe

T = Transferbit der Befehle BLD und BST

H = $Rd3 * Rr3 + Rr3 * /R3 + /R3 * Rd3$ Halbübertrag für BCD-Korrektur

S = $N [+]$ V signed Test

V = $Rd7 * Rr7 * /R7 + /Rd7 * /Rr7 * R7$ signed Überlauf

N = R7 signed Vorzeichen

Z = $/R7 * /R6 * /R5 * /R4 * /R3 * /R2 * /R1 * /R0$ Null wie NOR-Schaltung!

C = $Rd7 * Rr7 * Rr7 * /R7 * /R7 * Rd7$ Überlauf/Unterlauf Übertrag/Borgen

Operatoren: ***** = UND **+** = ODER **[+]** = EODER **/** = Negation

Die Wirkung eines Befehls auf die Statusbits:

- <> Bit wird durch den Befehl entsprechend dem Resultat verändert
- 0 Bit wird durch den Befehl gelöscht (0)
- 1 Bit wird durch den Befehl gesetzt (1)
- Bit wird durch den Befehl *nicht* verändert

ADC - Add with Carry

Addiere zum Inhalt des Zielregisters Rd den Inhalt des Quellregisters Rr und das Carrybit. Der Inhalt des Quellregisters Rr bleibt erhalten; Ziel Rd wird mit der Summe überschrieben.

adc Rd, Rr ; Rd <= Rd + Rr + Carry

Als Ziel und als Quelle sind alle Register von R0 bis R31 zugelassen. Die Summe verändert alle Statusbits außer I und T.

I	T	H	S	V	N	Z	C
-	-	<>	<>	<>	<>	<>	<>

Befehlswort: **0001 11**rd dddd rrrr ein Takt

Beispiel:

```
; 16bit Addition R17:R16 <= R17:R16 + R1:R0
add    r16,r0    ; addiere Low-Bytes
adc    r17,r1    ; addiere High-Bytes + Übertrag
brcs   fehler    ; verzweige bei C = 1 Überlauf
```

ADD - Add without Carry

Addiere zum Inhalt des Zielregisters Rd den Inhalt des Quellregisters Rr. Der Inhalt des Quellregisters Rr bleibt erhalten; das Zielregister Rd wird mit der Summe überschrieben.

add Rd, Rr ; Rd <= Rd + Rr

Als Ziel und als Quelle sind alle Register von R0 bis R31 zugelassen. Die Summe verändert alle Statusbits außer I und T.

I	T	H	S	V	N	Z	C
-	-	<>	<>	<>	<>	<>	<>

Befehlswort: **0000 11**rd dddd rrrr ein Takt

Beispiel:

```
; 8bit Addition R16 <= R16 + R17
add    r16,r17   ; addiere Bytes
brcs   fehler    ; verzweige bei C = 1 Überlauf
```

ADIW - Add Immediate to Word

Addiere zum Inhalt des Zielregisterpaars Rd+1:Rd die vorzeichenlose 6bit Konstante des zweiten Operanden. Die 16bit Summe überschreibt den Inhalt des Zielregisterpaars.

adiw Rd+1:Rd, K6 ; Rd+1:Rd <= Rd+1:Rd + Konstante 0 bis 63

adiw Rd, K6 ; nur Low-Register angegeben

Als Ziel sind nur die Registerpaare R25:R24, R27:R26, R29:R28 und R31:R30 zugelassen. Als Operanden können auch die Low-Register R24, XL, YL und ZL angegeben werden.

I	T	H	S	V	N	Z	C
-	-	-	<>	<>	<>	<>	<>

Befehlswort: **1001 0110** KKdd KKKK zwei Takte

Beispiel:

; 16bit Addition ZH:ZL <= ZH:ZL + 1

adiw r31:r30,1 ; Registerpaar angegeben

adiw ZL,1 ; nur Low-Register angegeben

AND - Logical AND

Verknüpfe den Inhalt des Zielregisters Rd mit dem Inhalt des Quellregisters Rr durch ein logisches UND. Der Inhalt des Quellregisters Rr bleibt erhalten. Das Ziel wird mit dem Resultat überschrieben.

and Rd, Rr ; Rd <= Rd UND Rr

Als Ziel Rd und als Quelle Rr sind alle Register von R0 bis R31 zugelassen. Das Resultat ist nur dann 1, wenn die entsprechenden Bitpositionen *beider* Operanden 1 sind.

I	T	H	S	V	N	Z	C
-	-	-	<>	0	<>	<>	-

Befehlswort: **0010 00rd** dddd rrrr ein Takt

Beispiel:

; lösche das linke Halbbyte von R0 durch eine UND-Maske

ldi r16,\$0F ; lade R16 mit der Maske 0000 1111

and r0,r16 ; R0 <- R0 UND R16 R0 <- 0000 xxxx

ANDI - Logical AND with Immediate

Verknüpfe den Inhalt des Zielregisters Rd mit der 8bit Konstanten des zweiten Operanden durch ein logisches UND. Das Ziel wird mit dem Resultat überschrieben.

andi Rd, K8 ; Rd <= Rd UND 8bit Konstante

Als Ziel Rd sind nur die Register von R16 bis R31 zugelassen. Das Resultat ist nur dann 1, wenn die entsprechenden Bitpositionen *beider* Operanden 1 sind.

I	T	H	S	V	N	Z	C
-	-	-	<>	0	<>	<>	-

Befehlsword: **0111** KKKK dddd KKKK ein Takt

Beispiel:

; lösche das linke Halbbyte von R16 durch eine UND-Maske

andi r16,\$0F ; R16 <- R16 UND 0000 1111 R16 <= 0000 xxxx

ASR - Arithmetic Shift Right

Verschiebe den Inhalt des Zielregisters Rd arithmetisch um eine Bitposition nach rechts.

asr Rd ; |B7 B6 . . B1 B0| -> |**B7** B7 B6 . . B1| **B0** -> C

Als Ziel Rd sind alle Register von R0 bis R31 zugelassen. Die links freiwerdende Bitposition, das Vorzeichen, bleibt erhalten. Das rechts herausgeschobene Bit gelangt in das Carrybit.

I	T	H	S	V	N	Z	C
-	-	-	<>	<>	<>	<>	<>

Befehlsword: **1001 010d** dddd **0101** ein Takt

Beispiele:

; Dividiere R16 durch 2 Rest im Carrybit

ldi r16,8 ; R16 = 0000 1000 = +8

asr r16 ; R16 = 0000 0100 = +4 Carry = 0

ldi r16,-8 ; R16 = 1111 1000 = -8

asr r16 ; R16 = 1111 1100 = -4 Carry = 0

BCLR - Bit Clear in SREG

Lösche die im Operandenteil angegebene Bitposition des Statusregisters SREG. Die Pseudobefehle **clx** enthalten die Bitposition im Befehl.

bclr bit ; lösche Bitposition 0 bis 7 in SREG

Die als Konstante von 0 bis 7 anzugebene Bitposition des Statusregisters wird gelöscht (0). Alle anderen Bitpositionen bleiben erhalten.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1001 0100 1sss 1000** ein Takt

Beispiel:

; lösche das Carrybit

bclr 0 ; lösche Bitposition 0 in SREG (das Carrybit)

clc ; Pseudobefehl lösche das Carrybit

BSET - Bit Set in SREG

Setze die im Operandenteil angegebene Bitposition des Statusregisters SREG auf 1. Die Pseudobefehle **sexx** enthalten die Bitposition im Befehl.

bset bit ; setze Bitposition 0 bis 7 in SREG

Die als Konstante von 0 bis 7 anzugebene Bitposition des Statusregisters wird gesetzt (1). Alle anderen Bitpositionen bleiben erhalten.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1001 0100 0sss 1000** ein Takt

Beispiel:

; setze das Carrybit

bset 0 ; setze Bitposition 0 in SREG (das Carrybit)

sec ; Pseudobefehl setze das Carrybit

BLD - Bit Load from T Flag in SREG to Bit in Register

Lade die Bitposition von Rd, die im zweiten Operanden angegeben ist, mit dem T-Bit des Statusregisters SREG.

bld Rd, bit ; lade Bitposition von Rd mit T-Bit

Als Ziel Rd sind alle Register von R0 bis R31 zugelassen. Die mit dem T-Bit zu ladende Bitposition ist eine Konstante von 0 bis 7. Alle anderen Bits des Registers bleiben erhalten.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1111 100d** dddd 0bbb ein Takt

Beispiel:

```
; kopiere Bit B7 von R16 nach Bit B0 von R17
    bst    r16,7      ; speichere Bit B7 von R16 nach T-Bit
    bld    r17,0      ; lade Bit B0 von R16 mit dem T-Bit
```

BST - Bit store from Bit in Register to T Flag in SREG

Speichere die Bitposition von Rd, die im zweiten Operanden angegeben ist, in das T-Bit des Statusregisters SREG.

bst Rd, bit ; speichere Bitposition von Rd nach T-Bit

Als Quelle Rd sind alle Register von R0 bis R31 zugelassen. Die in das T-Bit zu speichernde Bitposition ist eine Konstante von 0 bis 7. Alle anderen Bits von SREG bleiben erhalten.

I	T	H	S	V	N	Z	C
-	<>	-	-	-	-	-	-

Befehlswort: **1111 101d** dddd 0bbb ein Takt

Beispiel:

```
; kopiere Bit B7 von R16 nach Bit B0 von R17
    bst    r16,7      ; speichere Bit B7 von R16 nach T-Bit
    bld    r17,0      ; lade Bit B0 von R17 mit dem T-Bit
```

BRBC - Branch if Bit in SREG is Cleared

Verzweige zum Ziel im vorzeichenbehafteten (signed) 7bit Abstand k, wenn Bitposition s in SREG gelöscht (0) ist. Die Pseudobefehle **brxx** enthalten die Bitposition 0 bis 7 im Befehl. Der Assembler berechnet den Abstand k von -64 bis +63 aus der symbolischen Zieladresse.

brbc s, k ; verzweige im Abstand k wenn Bit s gelöscht ist

Ist Bitposition **s** = 0, so verzweigt das Programm zum Ziel $PC \leq PC + k + 1$

Ist Bitposition **s** = 1, so wird der nächste Befehl ausgeführt $PC \leq PC + 1$

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1111 01kk kkkk ksss** Sprung: 2 Takte / Nichtsprung: 1 Takt

Beispiel:

; verzweige, wenn kein Überlauf, zum Ziel nein

brbc 0,nein ; verzweige bei C = 0 Pseudobefehl brcc

BRBS - Branch if Bit in SREG is Set

Verzweige zum Ziel im vorzeichenbehafteten (signed) 7bit Abstand k, wenn Bitposition s in SREG gesetzt (1) ist. Die Pseudobefehle **brxx** enthalten die Bitposition 0 bis 7 im Befehl. Der Assembler berechnet den Abstand k von -64 bis +63 aus der symbolischen Zieladresse.

brbs s, k ; verzweige in Abstand k wenn Bit s gesetzt ist

Ist Bitposition **s** = 1, so verzweigt das Programm zum Ziel $PC \leq PC + k + 1$

Ist Bitposition **s** = 0, so wird der nächste Befehl ausgeführt $PC \leq PC + 1$

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1111 00kk kkkk ksss** Sprung: 2 Takte / Nichtsprung: 1 Takt

Beispiel:

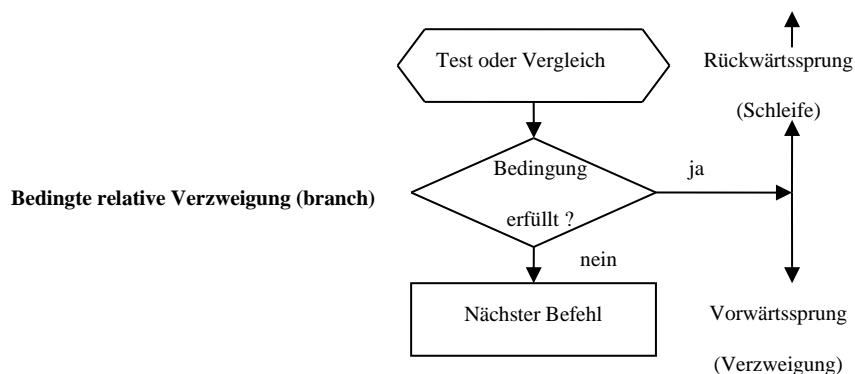
; verzweige bei Überlauf zum Ziel fehler

brbs 0,fehler ; verzweige bei C = 1 Pseudobefehl brcs

BRxx - Pseudobefehle Branch if . . .

Für die Pseudobefehle **BRxx** setzt der Assembler die entsprechende Bitposition s des Statusregisters in den Code der Befehle **brbc** und **brbs** ein und berechnet den Abstand k vom Stand des Adresszählers zur Zieladresse.

Befehl	Operand	Wirkung	Befehlswort	T
brbc	bit, ziel	nein: PC <= PC + 1 ja: PC <= PC + Abstand	1111 01kk kkkk ksss	1/2
brbs	bit, ziel	nein: PC <= PC + 1 ja: PC <= PC + Abstand	1111 00kk kkkk ksss	1/2
brcc	ziel	Verzweigung bei C = 0	1111 01kk kkkk k000	1/2
brcs	ziel	Verzweigung bei C = 1	1111 00kk kkkk k000	1/2
brsh	ziel	Verzweigung bei C = 0 unsigned größer/gleich	1111 01kk kkkk k000	1/2
brlo	ziel	Verzweigung bei C = 1 unsigned kleiner als	1111 00kk kkkk k000	1/2
brne	ziel	Verzweigung bei Z = 0 ungleich	1111 01kk kkkk k001	1/2
breq	ziel	Verzweigung bei Z = 1 gleich	1111 00kk kkkk k001	1/2
brpl	ziel	Verzweigung bei N = 0 signed plus	1111 01kk kkkk k010	1/2
brmi	ziel	Verzweigung bei N = 1 signed minus	1111 00kk kkkk k010	1/2
brvc	ziel	Verzweigung bei V = 0 signed kein Überlauf	1111 01kk kkkk k011	1/2
brvs	ziel	Verzweigung bei V = 1 signed Überlauf	1111 00kk kkkk k011	1/2
brge	ziel	Verzweigung bei S = 0 signed größer/gleich	1111 01kk kkkk k100	1/2
brlt	ziel	Verzweigung bei S = 1 signed kleiner als	1111 00kk kkkk k100	1/2
brhc	ziel	Verzweigung bei H = 0 kein Halbübertrag	1111 01kk kkkk k101	1/2
brhs	ziel	Verzweigung bei H = 1 Halbübertrag	1111 00kk kkkk k101	1/2
brtc	ziel	Verzweigung bei T = 0 gelöscht	1111 01kk kkkk k110	1/2
brts	ziel	Verzweigung bei T = 1 gesetzt	1111 00kk kkkk k110	1/2
brid	ziel	Verzweigung bei I = 0 Interrupts global gesperrt	1111 01kk kkkk k111	1/2
bric	ziel	Verzweigung bei I = 1 Interrupts global frei	1111 00kk kkkk k111	1/2



BREAK - Break

Der Befehl bringt den Baustein für Testzwecke in einen Stopp-Zustand. Er ist nur für Bausteine mit einem internen Testsystem (On-Chip Debug System) vorgesehen und nicht für den Anwender verfügbar. Er wird sonst wie ein NOP-Befehl ausgeführt.

break ; Stopp-Zustand für Testzwecke

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1001 0101 1001 1000** ein Takt

CALL - Long Call to a Subroutine

Rufe das Unterprogramm an der angegebenen Zieladresse auf. Der Assembler setzt den Absolutwert der Zieladresse in die beiden Befehlswörter ein. Die Rücksprungadresse wird auf den Stapel gerettet. Der Befehl ist nicht für alle Bausteine verfügbar.

call Ziel ; Unterprogrammaufruf

Bausteine mit 16bit Befehlszähler $SP \leq SP - 2$ *Retten der Rücksprungadresse*

Bausteine mit 22bit Befehlszähler $SP \leq SP - 3$ *Retten der Rücksprungadresse*

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Zwei Befehlswörter: **1001 010k kkkk 111k kkkk kkkk kkkk kkkk**

Bausteine mit 16bit Befehlszähler: 4 Takte

Bausteine mit 22bit Befehlszähler: 5 Takte

Beispiel:

```
; Rufe das Unterprogramm warte
call warte ; Sprung zum Warte-Unterprogramm
```

CBI - Clear Bit in I/O Register

Lösche im SFR-Register des ersten Operanden die im zweiten Operanden angegebene Bitposition. Alle anderen Bitpositionen bleiben unverändert. Das SFR-Register wird intern gelesen und verändert zurückgeschrieben. Vorsicht: Dabei können in Steuer- und Statusregistern z.B. andere Anzeigeflags unbeabsichtigt zurückgesetzt werden!

cbi Port, bit ; lösche im SFR-Register die Bitposition

Der Befehl ist nur für SFR-Register auf den unteren Adressen 0 bis 31 (\$00 bis \$1F) anwendbar. Die Statusbits bleiben unverändert.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1001 1000** AAAA Abbb zwei Takte

Beispiel:

; Lösche im SFR-Register PORTB die Bitposition PB7

cbi PORTB,PB7 ; lösche Bit PB7, lege Portausgang auf Low

CBR - Clear Bits in Register

Lösche im Arbeitsregister Rd des ersten Operanden die Bitpositionen, die im Maskenbyte des zweiten Operanden 1 sind. Alle anderen Bitpositionen bleiben unverändert. Der Assembler setzt für den Pseudobefehl `cbr` den Code des Befehls `andi` mit der komplementierten Maske als Konstante ein.

cbr Rd, Maske ; lösche Bits, die in der Maske eine **1** haben

Als Ziel Rd sind nur die Register von R16 bis R31 zugelassen. Die Statusbits werden entsprechend dem `andi`-Befehl verändert!

I	T	H	S	V	N	Z	C
-	-	-	<>	0	<>	<>	-

Befehlswort: **0111** KKKK dddd KKKK ein Takt

Beispiel:

; lösche das linke Halbyte von R16

cbr r16,\$F0 ; Maske 1111 0000 gibt ANDI R16,0b00001111

CLxx - Pseudobefehle Clear

Für die Bitbefehle **clxx** setzt der Assembler die zu löschende Bitposition in den Befehl **bclr** ein. Der Befehl **clr** löscht *alle* Bits des Registers durch einen **eor**-Befehl.

Befehl	Operand	ITHSVNZC	Wirkung	Befehlswort	T
bclr	bitpos	<i>bit</i> <= 0	SREG(bit) <= 0 <i>lösche Bit in SREG</i>	1001 0100 1sss 1000	1
clc		-----0	C <= 0 <i>lösche Übertragbit</i>	1001 0100 1000 1000	1
clz		-----0-	Z <= 0 <i>lösche Nullanzeigebit</i>	1001 0100 1001 1000	1
cln		-----0--	N <= 0 <i>lösche Negativanzeigebit</i>	1001 0100 1010 1000	1
clv		----0---	V <= 0 <i>lösche Überlaufbit</i>	1001 0100 1011 1000	1
cls		---0----	S <= 0 <i>lösche Vorzeichenbit</i>	1001 0100 1100 1000	1
clh		--0-----	H <= 0 <i>lösche Halbübertrag</i>	1001 0100 1101 1000	1
clt		-0-----	T <= 0 <i>lösche Transferbit</i>	1001 0100 1110 1000	1
cli		0-----	I <= 0 <i>lösche Interruptbit, Sperre</i>	1001 0100 1111 1000	1
clr	Rd	---0001-	eor rd,rd <i>lösche alle Bits in Rd</i>	0010 01rd dddd rrrr	1

Beispiele:

```
; lösche das Carrybit und lösche das gesamte Register R16
    clc                ; lösche das Carrybit wie Befehl BCLR 0
    clr    r16          ; lösche das gesamte Register R16 wie EOR R16,R16
```

COM - One's Complement

Komplementiere den Inhalt des Zielregisters Rd nach der Regel „aus 0 mach 1 und aus 1 mach 0“. Der Befehl wird wie eine Subtraktion \$FF – Rd ausgeführt.

com Rd ; Rd <= \$FF - Rd (Einerkomplement)

Als Ziel Rd sind alle Register von R0 bis R31 zugelassen.

I	T	H	S	V	N	Z	C
-	-	-	<>	0	<>	<>	1

Befehlswort: **1001 010d dddd 0000** ein Takt

Beispiel:

```
; komplementiere R16
    ldi    r16,$F0    ; R16 <= 1111 0000 alter Wert
    com    r16        ; R16 <= 0000 1111 neuer Wert
```

CP - Compare

Vergleiche den Inhalt des Zielregisters Rd mit dem Inhalt des Quellregisters Rr durch eine Testsubtraktion $Rd - Rr$. Der Inhalt beider Register bleibt erhalten.

cp Rd, Rr ; Testsubtraktion $Rd - Rr$

Als Ziel und als Quelle sind alle Register von R0 bis R31 zugelassen. Die Differenz verändert alle Statusbits außer I und T.

I	T	H	S	V	N	Z	C
-	-	<>	<>	<>	<>	<>	<>

Befehlswort: **0001 01**rd dddd rrrr ein Takt

Beispiel:

```
; vergleiche das Register R16 mit dem Register R17
cp      r16,r17    ; Testsubtraktion R16 - R17
breq    gleich     ; verzweige bei Gleichheit
```

CPC - Compare with Carry

Vergleiche den Inhalt des Zielregisters Rd mit dem Inhalt des Quellregisters Rr und dem Carrybit durch eine Testsubtraktion $Rd - Rr - \text{Carry}$. Der Inhalt der Register bleibt erhalten.

cpc Rd, Rr ; Testsubtraktion $Rd - Rr - \text{Carry}$

Als Ziel und als Quelle sind alle Register von R0 bis R31 zugelassen. Das neue Z-Bit ist nur dann 1, wenn das alte Z-Bit 1 war UND die neue Differenz Null ist.

I	T	H	S	V	N	Z	C
-	-	<>	<>	<>	<>	<>	<>

Befehlswort: **0000 01**rd dddd rrrr ein Takt

Beispiel:

```
; 16bit Vergleich R17:R16 mit R1:R0
cp      r16,r0     ; Testsubtraktion Low-Bytes
cpc     r17,r1     ; Testsubtraktion High-Bytes und Borgen
breq    gleich     ; verzweige bei Gleichheit
```

CPI - Compare with Immediate

Vergleiche den Inhalt des Zielregisters Rd mit der 8bit Konstanten im zweiten Operanden durch eine Testsubtraktion $Rd - \text{Konstante}$. Der Inhalt des Registers bleibt erhalten.

cp Rd, K8 ; Testsubtraktion $Rd - \text{Konstante}$

Als Ziel sind nur die Register von R16 bis R31 zugelassen. Die Differenz verändert alle Statusbits außer I und T. Einen Vergleichsbefehl `cpic` mit zusätzlichem Carry gibt es nicht.

I	T	H	S	V	N	Z	C
-	-	<>	<>	<>	<>	<>	<>

Befehlswort: **0011** KKKK dddd KKKK ein Takt

Beispiel:

```
; vergleiche das Register R16 mit der Ziffer 0
    cpi    r16,'0'    ; Testsubtraktion R16- $30
    brlo   fehler     ; verzweige wenn R16 < Ziffer 0
```

CPSE - Compare Skip if Equal

Vergleiche den Inhalt des Zielregisters Rd mit dem Inhalt des Quellregisters Rr durch eine Testsubtraktion $Rd - Rr$. Überspringe den nächsten Befehl, wenn beide Registerinhalte gleich sind. Der Inhalt beider Register bleibt erhalten.

cpse Rd, Rr ; Testsubtraktion $Rd - Rr$

Als Ziel und als Quelle sind alle Register von R0 bis R31 zugelassen. Alle Statusbits bleiben unverändert.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **0001 00**rd dddd rrrr kein Sprung: ein Takt
 Sprung über Ein-Wort-Befehl: zwei Takte
 Sprung über Zwei-Wort-Befehl: drei Takte

Beispiel:

```
; vergleiche R16 mit R17
    cpse   r16,r17    ; überspringe wenn R16 gleich R17
    inc    r16         ; wird nur bei R16 ungleich R17 ausgeführt
```

DEC - Decrement

Subtrahiere eine 1 vom Inhalt des Zielregisters Rd.

dec Rd ; Rd <= Rd - 1

Als Ziel sind alle Register von R0 bis R31 zugelassen. Die Differenz verändert *nicht* die beiden Übertragbits Halfcarry und Carry; vorzeichenlose Dualzahlen lassen sich daher nur auf Gleichheit oder Ungleichheit prüfen.

I	T	H	S	V	N	Z	C
-	-	-	<>	<>	<>	<>	-

Befehlswort: **1001 0101d dddd 1010** ein Takt

Beispiel:

```
; Abwärts-Zählschleife in R16 für 100 Durchläufe
    ldi    r16,100    ; Zähler Anfangswert
warte: dec    r16      ; Zähler um 1 vermindern
    brne   warte      ; bis Zähler Null
```

EICALL EIJMP ELPM - Extended Operations ...

Die drei Befehle mit indirekter Adressierung sind nur für Controller mit mehr als 128 kByte Programmspeicher verfügbar. Sie benötigen zusätzliche Zeigerregister im SFR-Bereich.

eicall ; Unterprogrammaufruf Adresse in EIND und Z
eijmp ; unbedingter Sprung Adresse in EIND und Z
elpm ; R0 <= Flashspeicher Adresse in RAMPZ und Z
elpm Rd, Z ; Rd <= Flashspeicher Adresse in RAMPZ und Z
elpm Rd, Z+ ; Rd <= Flashspeicher Adresse in RAMPZ und Z, Z+1

Die Befehle verändern keine Statusbits.

Befehlswort: **1001 0101 0001 1001** vier Takte für **eicall**
1001 0100 0001 1001 zwei Takte für **eijmp**
1001 0101 1101 1000 drei Takte für **elpm**
1001 000d dddd 0110 drei Takte für **elpm Rd, Z**
1001 000d dddd 0011 drei Takte für **elpm Rd, Z+**

EOR - Exclusive OR

Verknüpfe den Inhalt des Zielregisters Rd mit dem Inhalt des Quellregisters Rr durch eine logische EODER-Funktion. Der Inhalt des Quellregisters Rr bleibt erhalten.

eor Rd, Rr ; Rd <= Rd EODER Rr

Als Ziel Rd und als Quelle Rr sind alle Register von R0 bis R31 zugelassen. Das Resultat ist nur dann 1, wenn die Bitpositionen der Operanden *ungleich* sind. Eine 1 in einer EODER-Maske komplementiert; eine 0 übernimmt. Der Befehl eori mit einer Konstanten fehlt!

I	T	H	S	V	N	Z	C
-	-	-	<>	0	<>	<>	-

Befehlswort: **0010 01**rd dddd rrrr ein Takt

Beispiel:

```
; komplementiere das linke Halbbyte von R16 durch eine EODER-Maske
ldi    r16,$55    ; R16 <= 0101 0101 Anfangswert
ldi    r17,$F0    ; R17 <= 1111 0000 EODER-Maske in R17 statt eori!
eor    r16,r17    ; R16 <= 1010 0101 linkes Halbbyte komplementiert
```

FMUL FMULS FMULSU - Fractional Multiply

Die drei Befehle multiplizieren Festpunktzahlen im Format 1.7 in Rd und Rr zu einem 16bit Produkt in den Registern R1 (High-Byte) und R0 (Low-Byte) im Format 1.15. Sie sind nicht bei allen Controllern verfügbar. Der Inhalt von Rd und Rr bleibt erhalten.

```
fmul    Rd, Rr    ; R1:R0 unsigned <= Rd unsigned * Rr unsigned
fmuls   Rd, Rr    ; R1:R0 signed   <= Rd signed   * Rr signed
fmulsu  Rd, Rr    ; R1:R0 signed   <= Rd signed   * Rr unsigned
```

Als Rd und Rr sind nur die Register R16 bis R23 zugelassen.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	<>	R16

Befehlswort: **0000 0011 0**ddd **1**rrr Befehl fmul zwei Takte

Befehlswort: **0000 0011 1**ddd **0**rrr Befehl fmuls zwei Takte

Befehlswort: **0000 0011 1**ddd **1**rrr Befehl fmulsu zwei Takte

ICALL - Indirect Call to Subroutine

Rufe das Unterprogramm auf, dessen Adresse im Z-Register steht. Für Adressen größer als 16bit ist `icall` zu verwenden. Der Befehl ist nicht für alle Bausteine verfügbar.

icall ; Unterprogrammaufruf Adresse in ZH:ZL

Bausteine mit 16bit Befehlszähler $SP \leq SP - 2$ *Retten der Rücksprungadresse*

Bausteine mit 22bit Befehlszähler $SP \leq SP - 3$ *Retten der Rücksprungadresse*

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1001 0101 0000 1001**

Bausteine mit 16bit Befehlszähler: 3 Takte

Bausteine mit 22bit Befehlszähler: 4 Takte

Beispiel:

```
; indirekter Unterprogrammaufruf
ldi    ZL,LOW(ziel) ; ZL <= Low-Adresse
ldi    ZH,HIGH(ziel) ; ZH <= High-Adresse
icall ; Sprung zum Unterprogramm, Adresse in Z
```

IJMP - Indirect Jump

Springe unbedingt zu einem Ziel, dessen Adresse im Z-Register steht. Für Adressen größer als 16bit ist `ijmp` zu verwenden. Der Befehl ist nicht für alle Bausteine verfügbar.

ijmp ; unbedingter Sprung Adresse in ZH:ZL

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1001 0100 0000 1001** zwei Takte

Beispiel:

```
; indirekter unbedingter Sprung
ldi    ZL,LOW(ziel) ; ZL <= Low-Adresse
ldi    ZH,HIGH(ziel) ; ZH <= High-Adresse
ijmp ; unbedingter Sprung, Adresse in Z
```


IN - Load an I/O Location to a Register

Lade das Arbeitsregister Rd mit dem Inhalt des SFR-Registers.

```
in    Rd, Port ; Rd <= SFR-Register
```

Als Ziel sind alle Register von R0 bis R31 zugelassen. Als Port sind alle SFR-Register bis \$3F zugelassen. Für SFR-Adressen über \$3F sind die SRAM-Befehle `ld` zu verwenden.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1011 0AA**d dddd AAAA ein Takt

Beispiel:

```
; lade R16 mit den an PIND anliegenden Potentialen
in    r16,PIND ; R16 <= Eingabe von PIND
```

INC - Increment

Addiere eine 1 zum Inhalt des Zielregisters Rd.

```
inc   Rd      ; Rd <= Rd + 1
```

Als Ziel sind alle Register von R0 bis R31 zugelassen. Die Summe verändert *nicht* die beiden Übertragbits **Halfcarry** und **Carry**; vorzeichenlose Dualzahlen lassen sich daher nur auf Gleichheit oder Ungleichheit prüfen.

I	T	H	S	V	N	Z	C
-	-	-	<>	<>	<>	<>	-

Befehlswort: **1001 010**d dddd **0011** ein Takt

Beispiel:

```
; Aufwärts-Zählschleife in R16 für Werte von 1 bis 100
    ldi    r16,1      ; Zähler Anfangswert
warte: inc   r16        ; Zähler um 1 erhöhen
    cpi    r16,100     ; vergleiche Zähler mit Endwert
    brne   warte       ; bis Zähler Endwert
```

JMP - Jump

Springe unbedingt zur angegebenen Zieladresse. Der Assembler setzt den Absolutwert k der Zieladresse in die beiden Befehlswörter ein. Der Befehl ist nicht für alle Bausteine verfügbar.

jmp Ziel ; unbedingter Sprung

Bedingungsbits werden nicht verändert.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Zwei Befehlswörter: **1001 010**k kkkk **110**k kkkk kkkk kkkk kkkk drei Takte

Beispiel:

```
; springe unbedingt zum Ziel fehler
    jmp fehler ; Sprung zur Behandlung des Fehlers
```

LDI - Load Immediate

Lade das Zielregister Rd mit der 8bit Konstanten von 0 bis 255 des zweiten Operanden. Das Ziel wird überschrieben.

ldi Rd, K8 ; Rd <= 8bit Konstante

Als Ziel Rd sind nur die Register von R16 bis R31 zugelassen. Die Register R0 bis R15 müssen über Hilfsregister geladen werden. Die Bedingungsbits werden nicht verändert.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1110** KKKK dddd KKKK ein Takt

Beispiel:

```
; lade R0 mit der Konstante $55 über R16 als Hilfsregister
    ldi r16,$55 ; R16 <= Konstante
    mov r0,r16 ; R0 <= Konstante
```

LD LDD - Load Indirect from data space to register

Lade das Zielregister Rd mit einem durch XH:XL, YH:YL bzw. ZH:ZL indirekt adressierten Byte aus dem SRAM. Alle Statusbits bleiben unverändert.

Das Wortregister mit der Speicheradresse wird durch die Operation *nicht* verändert.

Befehl	Operand	Wirkung		Befehlswort	T
ld	Rd, X	Rd <= (X)	<i>lade Rd indirekt mit SRAM</i>	1001 000 d dddd 1100	2
ld	Rd, Y	Rd <= (Y)	<i>lade Rd indirekt mit SRAM</i>	1000 000 d dddd 1001	2
ld	Rd, Z	Rd <= (Z)	<i>lade Rd indirekt mit SRAM</i>	1000 000 d dddd 0000	2

Das Wortregister mit der Speicheradresse wird *nach* der Operation um 1 *erhöht*.

Befehl	Operand	Wirkung		Befehlswort	T
ld	Rd, X+	Rd <= (X) X <= X+1	<i>lade Rd indirekt mit SRAM</i>	1001 000 d dddd 1101	2
ld	Rd, Y+	Rd <= (Y) Y <= Y+1	<i>lade Rd indirekt mit SRAM</i>	1001 000 d dddd 1001	2
ld	Rd, Z+	Rd <= (Z) Z <= Z+1	<i>lade Rd indirekt mit SRAM</i>	1001 000 d dddd 0001	2

Das Wortregister mit der Speicheradresse wird *vor* der Operation um 1 *vermindert*.

Befehl	Operand	Wirkung		Befehlswort	T
ld	Rd, -X	X <= X-1 Rd <= (X)	<i>lade Rd indirekt mit SRAM</i>	1001 000 d dddd 1110	2
ld	Rd, -Y	Y <= Y-1 Rd <= (Y)	<i>lade Rd indirekt mit SRAM</i>	1001 000 d dddd 1010	2
ld	Rd, -Z	Z <= Z-1 Rd <= (Z)	<i>lade Rd indirekt mit SRAM</i>	1001 000 d dddd 0010	2

Die Speicheradresse im SRAM ergibt sich aus dem vorzeichenlosen Abstand (**D**isplacement) von 1 bis 63 plus dem Inhalt des Wortregisters; das Wortregister bleibt unverändert erhalten.

Befehl	Operand	Wirkung		Befehlswort	T
ldd	Rd, Y+q	Rd <= (Y + konst.)	<i>lade Rd indirekt mit SRAM</i>	10q0 qq0 d dddd 1qqq	2
ldd	Rd, Z+q	Rd <= (Z + konst.)	<i>lade Rd indirekt mit SRAM</i>	10q0 qq0 d dddd 0qqq	2

Beispiel:

```
; lade R16 mit Speicherbyte von der Adresse tab
    ldi    ZL,LOW(tab)  ; ZL <= Low-Adresse
    ldi    ZH,HIGH(tab) ; ZH <= High-Adresse
    ld     r16,Z         ; R16 <= SRAM-Byte durch Z adressiert
    .DSEG                                ; Datensegment im SRAM
tab:  .BYTE  1           ; 1 Byte reserviert
```

LDS - Load Direct from data space

Lade das Zielregister Rd mit dem direkt adressierten Byte aus dem SRAM.

lds Rd, Adresse ; Rd <= SRAM-Byte

Als Ziel Rd sind alle Register von R0 bis R31 zugelassen. Die Bedingungsbits werden nicht verändert.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

2 Befehlswörter: **1001 000d dddd 0000** kkkk kkkk kkkk kkkk zwei Takte

```
; lade R16 mit Speicherbyte von der Adresse tab
    lds    r16,tab    ; R16 <= SRAM-Byte direkt adressiert
    .DSEG                      ; Datensegment im SRAM
tab:    .BYTE    1      ; 1 Byte reserviert
```

LPM - Load Program Memory

Lade ein Arbeitsregister mit einem durch Z adressierten Byte im Flash-Programmspeicher. Nur der Befehl lpm ohne Operandenteil ist für alle Bausteine verfügbar.

Befehl	Operand	Wirkung		Befehlswort	T
lpm		R0 <= (Z)	<i>lade R0 indirekt mit Flash</i>	1001 0101 1100 1000	3
lpm	Rd, Z	Rd <= (Z)	<i>lade Rd indirekt mit Flash</i>	1001 000d dddd 0100	3
lpm	Rd, Z+	Rd <= (Z) Z <= Z+1	<i>lade Rd indirekt mit Flash</i>	1001 000d dddd 0101	3

Für die Befehle mit Rd als Ziel sind alle Register von R0 bis R31 zugelassen. Die Bedingungsbits werden nicht verändert.

Beispiel:

```
; lade R0 mit Byte von der Adresse koni
    ldi    ZL,LOW(koni*2) ; ZL <= Low-Adresse
    ldi    ZH,HIGH(koni*2) ; ZH <= High-Adresse
    lpm                                ; R0 <= Flash-Byte durch Z adressiert
; Konstantenbereich im Flash hinter den Befehlen
koni:    .DB    $55                ; Konstante 0101 0101 im Flash
```

LSL - Logical Shift Left

Verschiebe den Inhalt des Zielregisters Rd logisch um eine Bitposition nach links. Der Pseudobefehl wird als `add rd, rd` ausgeführt.

ls1 Rd ; C <- B7 | B6 B5 . . B1 0 | <- | B7 B6 . . B1 B0 |

Als Ziel Rd sind alle Register von R0 bis R31 zugelassen. Das links herausgeschobene Bit gelangt in das Carrybit. Die rechts frei werdende Bitposition wird mit einer 0 aufgefüllt.

I	T	H	S	V	N	Z	C
-	-	<>	<>	<>	<>	<>	<>

Befehlswort: **0000 11dd dddd dddd** ein Takt

Beispiel:

```
; Multipliziere R16 mit 2 Übertrag im Carrybit
ldi    r16,8      ; R16 <= 0000 1000 = 8
ls1    r16         ; R16 <= 0001 0000 = 16 C = 0 kein Übertrag
```

LSR - Logical Shift Right

Verschiebe den Inhalt des Zielregisters Rd logisch um eine Bitposition nach rechts.

lsr Rd ; | B7 B6 . . B1 B0 | -> | 0 B7 B6 . . B1 | B0 -> C

Als Ziel Rd sind alle Register von R0 bis R31 zugelassen. Das rechts herausgeschobene Bit gelangt in das Carrybit. Die links frei werdende Bitposition wird mit einer 0 aufgefüllt.

I	T	H	S	V	N	Z	C
-	-	-	<>	<>	0	<>	<>

Befehlswort: **1001 010d dddd 0110** ein Takt

Beispiele:

```
; Dividiere R16 durch 2 Rest im Carrybit
ldi    r16,8      ; R16 <= 0000 1000 = 8
lsr    r16         ; R16 <= 0000 0100 = 4 C = 0 Rest = 0
ldi    r16,9      ; R16 <= 0000 1001 = 9
lsr    r16         ; R16 <= 0000 0100 = 4 C = 1 Rest
```

MOV - Copy Register

Lade das Zielregister Rd mit dem Inhalt des Quellregisters Rr. Das Ziel wird überschrieben, der Inhalt der Quelle bleibt erhalten.

mov Rd, Rr ; Rd <= Rr

Als Ziel Rd und als Quelle Rr sind alle Register von R0 bis R31 zugelassen. Die Bedingungsbits werden nicht verändert.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlsword: **0010 11**rd dddd rrrr ein Takt

Beispiel:

```
; lade R17 mit R1 und R16 mit R0
mov    r17,r1    ; R17 <= R1
mov    r16,r0    ; R16 <= R0
```

MOVW - Copy Register Word

Lade die Zielregister Rd und Rd+1 mit dem Inhalt der Quellregister Rr und Rr+1. Die Wortoperation ist nicht für alle Bausteine verfügbar. Der Assembler lässt zwei Schreibweisen zu.

movw Rd+1:Rd, Rr+1:Rr ; *d r {0,2,4,...,26,28,30} geradzahlig*
movw Rd, Rr ; *d r {0,2,4,...,26,28,30} geradzahlig*

Als Ziel Rd und als Quelle Rr sind nur die geradzahligen Register von R0, R2 bis R28, R30 zugelassen. Die Bedingungsbits werden nicht verändert.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlsword: **0000 0001** dddd rrrr ein Takt

Beispiel:

```
; lade R17 mit R1 und R16 mit R0
movw    r17:r16,r1:r0 ; R17:R16 <= R1:R0
movw    r16,r0        ; R17:R16 <= R1:R0
```

MUL MULS MULSU - Multiply

Die drei Befehle multiplizieren ganze Zahlen in Rd und Rr zu einem 16bit Produkt in den Registern R1 (High-Byte) und R0 (Low-Byte). Sie sind nicht bei allen Controllern verfügbar. Der Inhalt von Rd und Rr bleibt erhalten.

```
mul      Rd, Rr  ; R1:R0 unsigned <= Rd unsigned * Rr unsigned
muls     Rd, Rr  ; R1:R0 signed   <= Rd signed   * Rr signed
mulsu    Rd, Rr  ; R1:R0 signed   <= Rd signed   * Rr unsigned
```

mul-Befehl: als Rd und Rr sind alle Register von R0 bis R31 zugelassen.

muls-Befehl: als Rd und Rr sind nur die Register von R16 bis R31 zugelassen.

mulsu-Befehl: als Rd und Rr sind nur die Register von R16 bis R23 zugelassen.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	<>	R15

Befehlswort: **1001 11**rd dddd rrrr Befehl mul zwei Takte

Befehlswort: **0000 0010** dddd rrrr Befehl muls zwei Takte

Befehlswort: **0000 0011 0**ddd 0rrr Befehl mulsu zwei Takte

Beispiel:

```
; unsigned 16bit Multiplikation R3:R2:R1:R0 <- R17:R16 * R19:R18
```

```
; Hilfsregister R4, R5 und R6 retten!
```

```
clr      r6          ; Hilfsregister für Addition löschen
mul     r17,r19      ; R1:R0 <= AH * BH
movw     r5:r4,r1:r0 ; R5:R4 <= High-Teilprodukt
mul     r16,r18      ; R1:R0 <= AL * BL
movw     r3:r2,r1:r0 ; R3:R2 <= Low-Teilprodukt
mul     r17,r18      ; R1:R0 <= AH * BL
add      r3,r0        ; Teilprodukte addieren
adc      r4,r1        ;
adc      r5,r6        ;
mul     r19,r16      ; R1:R0 <= BH * AL
add      r3,r0        ; Teilprodukte addieren
adc      r4,r1        ;
adc      r5,r6        ; Produkt in R5:R4:R3:R2
movw     r1:r0,r3:r2 ; R1:R0 <= Produkt_Low
movw     r3:r2,r5:r4 ; R3:R2 <= Produkt_High
```

NEG - Two's Complement

Negiere den Inhalt des Zielregisters Rd. Der Befehl wird wie eine Subtraktion $\$00 - \text{Rd}$ ausgeführt. Das Zweierkomplement entsteht aus dem Einerkomplement + 1.

neg Rd ; Rd <= $\$00 - \text{Rd}$ (Zweierkomplement)

Als Ziel Rd sind alle Register von R0 bis R31 zugelassen.

I	T	H	S	V	N	Z	C
-	-	<>	<>	<>	<>	<>	<>

Befehlswort: **1001 010d dddd 0001** ein Takt

Beispiel:

```
; negiere R16
ldi    r16,8      ; R16 <= 0000 1000 alter Wert +8
neg   r16        ; R16 <= 1111 1000 neuer Wert -8
```

NOP - No Operation

Führe keine Operation aus. Der Befehl dient als Platzhalter für Befehle und zum Einfügen von Wartetakten bei Zeitschleifen.

nop ; tu nix

Es werden keine Bedingungsbits verändert

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **0000 0000 0000 0000** ein Takt

Beispiel:

```
; Warteschleife für ca. 1 ms bei 1 MHz Systemtakt 1 µs Periode
ldi    r17,250    ; Wartezeit 250 µs * 4 Takte ca. 1 ms
warte: nop        ; 1 Takt: Zeitverzögerung
dec    r17        ; 1 Takt Wartezähler - 1
brne   warte      ; 2 Takte Sprung bis Wartezähler Null
```


OR - Logical OR

Verknüpfe den Inhalt des Zielregisters Rd mit dem Inhalt des Quellregisters Rr durch ein logisches ODER. Der Inhalt des Quellregisters Rr bleibt erhalten. Das Ziel wird mit dem Resultat überschrieben.

OR Rd, Rr ; Rd <= Rd ODER Rr

Als Ziel Rd und als Quelle Rr sind alle Register von R0 bis R31 zugelassen. Das Resultat ist dann 0, wenn die entsprechenden Bitpositionen *beider* Operanden 0 sind.

I	T	H	S	V	N	Z	C
-	-	-	<>	0	<>	<>	-

Befehlswort: **0010 10**rd dddd rrrr ein Takt

Beispiel:

```
; packe die beiden BCD-Ziffern in R16 und R17 zusammen
ldi    r16,$01 ; R16 <= $01 Zehner rechtsbündig
swap   r16      ; R16 <= $10 Zehner linksbündig
ldi    r17,$02 ; R17 <= $02 Einer rechtsbündig
or    r16,r17 ; R16 <= $12 gepackte zweistellige BCD-Zahl
```

ORI- Logical OR with Immediate

Verknüpfe den Inhalt des Zielregisters Rd mit der 8bit Konstanten von 0 bis 255 des zweiten Operanden durch ein logisches ODER. Das Ziel wird mit dem Resultat überschrieben.

ori Rd, K8 ; Rd <= Rd ODER 8bit Konstante

Als Ziel Rd sind nur die Register von R16 bis R31 zugelassen. Das Resultat ist nur dann 0, wenn die entsprechenden Bitpositionen *beider* Operanden 0 sind.

I	T	H	S	V	N	Z	C
-	-	-	<>	0	<>	<>	-

Befehlswort: **0110** KKKK dddd KKKK ein Takt

Beispiel:

```
; setze alle Bitposition im rechten Halbbyte von R16 auf 1
ori    r16,$0F ; R16 <= R16 ODER 0b00001111 R16 <= xxxx 1111
```

OUT - Store Register to I/O Location

Lade das SFR-Register mit dem Inhalt des Arbeitsregisters Rr.

```
out    Port, Rr    ; SFR-Register <= Rr
```

Als Quelle sind alle Register von R0 bis R31 zugelassen. Als Port sind alle SFR-Register bis \$3F zugelassen. Für SFR-Adressen über \$3F sind die SRAM-Befehle `ld` zu verwenden.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1011 1AA**d dddd AAAA ein Takt

Beispiel:

```
; programmiere den Port B als Ausgang
ldi    r16,$ff    ; R16 <= 1111 1111 für Ausgang
out    DDRB,r16    ; Richtungsregister DDRB <= R16
```

POP - Pop Register from Stack

Erhöhe den Stapelzeiger SP um 1. Lade das Zielregister Rd mit dem durch den neuen Stapelzeiger adressierten Byte. Der Befehl ist nicht bei allen Bausteinen verfügbar.

```
pop    Rd          ; SP <= SP + 1  Rd <= vom Stapel
```

Als Ziel sind alle Register von R0 bis R31 zugelassen.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1001 000**d dddd **1111** zwei Takte

Beispiel:

```
; Retten und Zurückladen von Registern in einem Unterprogramm
warte: push    r16          ; Register R16 nach Stapel retten
;
pop    r16          ; Register R16 vom Stapel zurückholen
ret          ; Rücksprung aus Unterprogramm
```

PUSH - Push Register on Stack

Speichere das Zielregister Rd in das durch den Stapelzeiger adressierte Byte. Vermindere anschließend den Stapelzeiger um 1. Der Befehl ist nicht bei allen Bausteinen verfügbar.

push Rr ; nach Stapel \leq Rd SP \leq SP - 1

Als Quelle sind alle Register von R0 bis R31 zugelassen.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1001 001d** dddd **1111** zwei Takte

Beispiel:

```
; Retten und Zurückladen von Registern in einem Unterprogramm
warte: push r16 ; Register R16 nach Stapel retten
;
      pop  r16 ; Register R16 vom Stapel zurückholen
      ret   ; Rücksprung aus Unterprogramm
```

RCALL - Relative Call to Subroutine

Rufe das Unterprogramm auf der angegebenen Zieladresse auf. Der Assembler setzt den Abstand zur Zieladresse in das Befehlswort ein.

rcall Ziel ; Unterprogrammaufruf

Bausteine mit 16bit Befehlszähler SP \leq SP - 2 *Rücksprungadresse retten*

Bausteine mit 22bit Befehlszähler SP \leq SP - 3 *Rücksprungadresse retten*

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1101** kkkk kkkk kkkk

Bausteine mit 16bit Befehlszähler: 3 Takte

Bausteine mit 22bit Befehlszähler: 4 Takte

Beispiel:

```
; Rufe das Unterprogramm warte
      rcall warte ; Sprung zum Warte-Unterprogramm
```

RET - Return from Subroutine

Rücksprung aus einem Unterprogramm durch Zurückladen des Befehlszählers vom Stapel.

ret ; Rücksprung aus Unterprogramm

Bausteine mit 16bit Befehlszähler $SP \leq SP + 2$ *Rücksprungadresse laden*

Bausteine mit 22bit Befehlszähler $SP \leq SP + 3$ *Rücksprungadresse laden*

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1001 0101 0000 1000**

Bausteine mit 16bit Befehlszähler: 4 Takte

Bausteine mit 22bit Befehlszähler: 5 Takte

Beispiel:

; Rücksprung aus Unterprogramm

ret ; Rücksprung

RETI - Return from Interrupt

Rücksprung aus einem Interrupt-Serviceprogramm durch Zurückladen des Befehlszählers vom Stapel. Das I-Bit des Statusregisters wird auf 1 gesetzt; Interrupts global wieder frei.

reti ; Rücksprung aus Serviceprogramm

Bausteine mit 16bit Befehlszähler $SP \leq SP + 2$ *Rücksprungadresse laden*

Bausteine mit 22bit Befehlszähler $SP \leq SP + 3$ *Rücksprungadresse laden*

I	T	H	S	V	N	Z	C
1	-	-	-	-	-	-	-

Befehlswort: **1001 0101 0001 1000**

Bausteine mit 16bit Befehlszähler: 4 Takte

Bausteine mit 22bit Befehlszähler: 5 Takte

Beispiel:

; Rücksprung aus Interrupt-Serviceprogramm

reti ; zurück zur Unterbrechung

RJMP - Relative Jump

Springe unbedingt zur angegebenen Zieladresse. Der Assembler setzt den Abstand zur Zieladresse in das Befehlswort ein.

rjmp Ziel ; unbedingter Sprung zur Zieladresse

Die Bedingungsbits werden nicht verändert.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1100** kkkk kkkk kkkk zwei Takte

Beispiel:

; springe unbedingt zum Ziel fehler

rjmp fehler ; Sprung zur Behandlung des Fehlers

ROL - Rotate Left through Carry

Verschiebe den Inhalt des Zielregisters Rd zyklisch um eine Bitposition nach links durch das Carrybit. Der Pseudobefehl wird als `adc rd, rd` ausgeführt.

rol Rd ; $C \leftarrow B7 \mid B6 \ B5 \ . \ . \ B0$ $C \mid \leftarrow B7 \ B6 \ . \ . \ B1 \ B0$

Als Ziel Rd sind alle Register von R0 bis R31 zugelassen. Das links herausgeschobene Bit gelangt in das Carrybit; das rechts frei werdende Bit wird mit dem alten Carry aufgefüllt.

I	T	H	S	V	N	Z	C
-	-	<>	<>	<>	<>	<>	<>

Befehlswort: **0001 11dd dddd dddd** ein Takt

Beispiel:

; 16bit Multiplikation R17:R16 mit 2 Übertrag im Carrybit

lsl r16 ; Low-Byte logisch links

rol r17 ; High-Byte mit Carry zyklisch links

ROR - Rotate Right through Carry

Verschiebe den Inhalt des Zielregisters Rd zyklisch um eine Bitposition nach rechts durch das Carrybit.

ror Rd ; |B7 B6 . . B1 B0| -> |C B7 B6 . . B1| **B0** -> C

Als Ziel Rd sind alle Register von R0 bis R31 zugelassen. In die links frei werdende Bitposition gelangt das alte Carrybit. Das rechts herausgeschobene Bit gelangt in das Carrybit.

I	T	H	S	V	N	Z	C
-	-	-	<>	<>	<>	<>	<>

Befehlswort: **1001 010d** dddd **0111** ein Takt

Beispiel:

```
; 16bit Division R17:R16 durch 2 Rest im Carrybit
    lsr    r17    ; High-Byte logisch rechts
    ror    r16    ; Low-Byte mit Carry zyklisch rechts
```

SBC - Subtract with Carry

Subtrahiere vom Inhalt des Zielregisters Rd den Inhalt des Quellregisters Rr und das Carrybit. Der Inhalt des Quellregisters Rr bleibt erhalten; Zielregister Rd und das Carrybit werden überschrieben.

sbc Rd, Rr ; Rd <= Rd - Rr - Carry

Als Ziel und als Quelle sind alle Register von R0 bis R31 zugelassen. Das neue Z-Bit ist nur dann 1, wenn das alte Z-Bit 1 war UND die neue Differenz Null ist.

I	T	H	S	V	N	Z	C
-	-	<>	<>	<>	<>	<>	<>

Befehlswort: **0000 10rd** dddd rrrr ein Takt

Beispiel:

```
; 16bit Subtraktion R17:R16 <= R17:R16 - R1:R0
    sub    r16,r0    ; subtrahiere Low-Bytes
    sbc    r17,r1    ; subtrahiere High-Bytes - Übertrag (Borgen)
    breq   ende      ; verzweige bei Differenz Null
```

SBCI - Subtract Immediate with Carry

Subtrahiere vom Inhalt des Zielregisters Rd die 8bit Konstante im zweiten Operanden und das Carrybit.

sbci Rd, K8 ; Rd <= Rd - Konstante - Carry

Als Ziel sind nur die Register von R16 bis R31 zugelassen. Das neue Z-Bit ist nur dann 1, wenn das alte Z-Bit 1 war UND die neue Differenz Null ist.

I	T	H	S	V	N	Z	C
-	-	<>	<>	<>	<>	<>	<>

Befehlswort: **0100** KKKK dddd KKKK ein Takt

Beispiel:

```
; 16bit Subtraktion R17:R16 <= R17:R16 - 16bit Konstante
subi    r16,LOW(1234) ; subtrahiere Low-Bytes
sbci    r17,HIGH(1234) ; subtrahiere High-Bytes - Übertrag (Borgen)
breq     ende          ; verzweige bei Differenz Null
```

SBIW - Subtract Immediate from Word

Subtrahiere vom Inhalt des Zielregisterpaars Rd+1:Rd die vorzeichenlose 6bit Konstante des zweiten Operanden. Die 16bit Differenz überschreibt den Inhalt des Zielregisterpaars.

sbiw Rd+1:Rd, K6 ; Rd+1:Rd <= Rd+1:Rd + Konstante 0 bis 63
sbiw Rd, K6 ; nur Low-Register angegeben

Als Ziel sind nur die Registerpaare R25:R24, R27:R26, R29:R28 und R31:R30 zugelassen. Als Operanden können auch die Low-Register R24, XL, YL und ZL angegeben werden.

I	T	H	S	V	N	Z	C
-	-	-	<>	<>	<>	<>	<>

Befehlswort: **1001 0111** KKdd KKKK zwei Takte

Beispiel:

```
; 16bit Subtraktion ZH:ZL <= ZH:ZL - 1
sbiw    r31:r30,1 ; Registerpaar angegeben
sbiw    ZL,1      ; nur Low-Register angegeben
```

SBIC SBIS SBRC SBRS - Skip if Bit is . . .

Die **bedingten Skipbefehle** (skip = übergehen) *überspringen* den nächsten Befehl, wenn die Bedingung erfüllt ist; ist sie nicht erfüllt, so wird der nächste Befehl ausgeführt. Sie verändern und werten keine Bedingungsbits aus und enthalten keine Zieladresse.

Befehl	Operand	Wirkung	Befehlswort	T
sbic	port,bit	überspringe, wenn Bit im Port gelöscht (0) ist	1001 1001 AAAA Abbb	1/2/3
sbis	port,bit	überspringe, wenn Bit im Port gesetzt (1) ist	1001 1011 AAAA Abbb	1/2/3
sbrc	Rd,bit	überspringe, wenn Bit im Register gelöscht (0) ist	1111 110r rrrr 0bbb	1/2/3
sbrs	Rd,bit	überspringe, wenn Bit im Register gesetzt (1) ist	1111 111r rrrr 0bbb	1/2/3

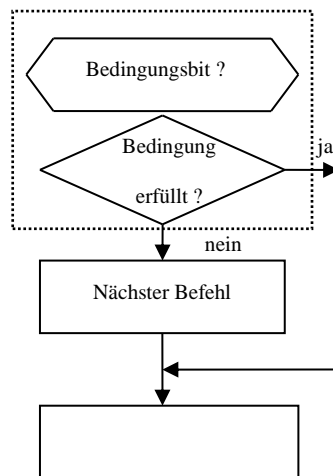
Für SFR-Adressen über \$3F sind die SRAM-Befehle `ld` mit einem Hilfsregister und einer Registeroperation zu verwenden. Als Arbeitsregister sind alle Register von R0 bis R31 zugelassen. Die Anzahl der Takte ist abhängig von der Ausführung des Sprungs:

- 1 Takt: kein Sprung und Ausführung des folgenden Befehls,
- 2 Takte: Sprung über einen 1-Wort-Befehl (z.B. `rjmp`) bzw.
- 3 Takte: Sprung über einen 2-Wort-Befehl (z.B. `jmp`)

Beispiel:

```
; warte auf fallende Flanke an PD7
warte: sbic  PIND,PD7    ; überspringe wenn Taste Low
      rjmp  warte       ; Warteschleife solange Taste High
```

Bedingter Sprung (skip)



SBI - Set Bit in I/O Register

Setze im SFR-Register des ersten Operanden die im zweiten Operanden angegebene Bitposition auf 1. Alle anderen Bitpositionen bleiben unverändert. Das SFR-Register wird intern gelesen und verändert zurückgeschrieben. Vorsicht: Dabei können in Steuer- und Statusregistern z.B. andere Anzeigeflags unbeabsichtigt zurückgesetzt werden!

sbi Port, bit ; setze im SFR-Register die Bitposition

Der Befehl ist nur für SFR-Register auf den unteren Adressen 0 bis 31 (\$00 bis \$1F) anwendbar. Die Statusbits bleiben unverändert.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1001 1010** AAAA Abbb zwei Takte

Beispiel:

; setze im SFR-Register PORTB die Bitposition PB7

sbi PORTB,PB7 ; setze Bit PB7, lege Portausgang auf High

SBR - Set Bits in Register

Setze im Arbeitsregister Rd des ersten Operanden die Bitpositionen, die im Maskenbyte des zweiten Operanden 1 sind. Alle anderen Bitpositionen bleiben unverändert. Der Assembler setzt für den Pseudobefehl **sbr** den Code des Befehls **ori** mit der Maske als Konstante ein.

sbr Rd, Maske ; setze Bits, die in der Maske eine 1 haben

Als Ziel Rd sind nur die Register von R16 bis R31 zugelassen. Die Statusbits werden entsprechend dem **ori**-Befehl verändert!

I	T	H	S	V	N	Z	C
-	-	-	<>	0	<>	<>	-

Befehlswort: **0110** KKKK dddd KKKK ein Takt

Beispiel:

; setze das linke Halbbyte von R16

sbr r16,\$F0 ; Maske 1111 0000 gibt ORI R16,0b1111 0000

SExx - Pseudobefehle Set . . .

Für die Bitbefehle `sexx` setzt der Assembler die zu setzende Bitposition in den Befehl `bset` ein. Der Befehl `ser` setzt alle Bits des Registers (nur R16 bis R31) durch `ldi Rd, $FF`.

Befehl	Operand	ITHSVNZC	Wirkung	Befehlswort	T
<code>bset</code>	<code>bitpos</code>	<i>bit</i> <= 1	SREG(bit) <= 1 <i>setze Bit in SREG</i>	1001 0100 0sss 1000	1
<code>sec</code>		-----1	C <= 1 <i>setze Übertragbit</i>	1001 0100 0000 1000	1
<code>sez</code>		-----1-	Z <= 1 <i>setze Nullanzeigebit</i>	1001 0100 0001 1000	1
<code>sen</code>		-----1--	N <= 1 <i>setze Negativanzeigebit</i>	1001 0100 0010 1000	1
<code>sev</code>		----1---	V <= 1 <i>setze Überlaufbit</i>	1001 0100 0011 1000	1
<code>ses</code>		---1----	S <= 1 <i>setze Vorzeichenbit</i>	1001 0100 0100 1000	1
<code>seh</code>		--1-----	H <= 1 <i>setze Halbübertrag</i>	1001 0100 0101 1000	1
<code>set</code>		-1-----	T <= 1 <i>setze Transferbit</i>	1001 0100 0110 1000	1
<code>sei</code>		1-----	I <= 1 <i>setze Interruptbit</i>	1001 0100 0111 1000	1
<code>ser</code>	Rd	-----	<code>ldi Rd, \$FF</code> <i>setze alle Registerbits</i>	1110 1111 dddd 1111	1

Beispiel:

```
; setze das Carrybit und alle Bits von R16 auf 1
    sec                ; setze Bit wie Befehl BSET 0
    ser r16            ; setze alle Bits wie Befehl ldi R16,0b11111111
```

SLEEP -

Startet den zuvor ins Registers MCUCR eingetragenen Ruhezustand.

`sleep` ; bringt Controller in Ruhezustand

Es werden keine Bedingungsbits verändert.

Befehlswort: 1001 0101 1000 1000 ein Takt

SPM - Store Program Memory

Der Befehl `spm` speichert in Verbindung mit Steuerregistern Daten in den Boot-Bereich und ist nicht für alle Bausteine verfügbar.

`spm` ; Daten nach Boot-Bereich

Befehlswort: 1001 0101 1110 1000 Ausführungszeit mehrere Millisekunden

ST STD - Store Indirect from Register to data space

Speichere das Quellregister Rr in ein durch XH:XL, YH:YL bzw. ZH:ZL indirekt adressiertes Byte im SRAM. Quellregister Rr und alle Statusbits bleiben unverändert.

Das Wortregister mit der Speicheradresse wird durch die Operation *nicht* verändert.

Befehl	Operand	Wirkung		Befehlswort	T
st	X, Rr	(X) <= Rr	<i>speichere Rr indirekt</i>	1001 001r rrrr 1100	2
st	Y, Rr	(Y) <= Rr	<i>speichere Rr indirekt</i>	1000 001r rrrr 1001	2
st	Z, Rr	(Z) <= Rr	<i>speichere Rr indirekt</i>	1000 001r rrrr 0000	2

Das Wortregister mit der Speicheradresse wird *nach* der Operation um 1 *erhöht*.

Befehl	Operand	Wirkung		Befehlswort	T
st	X+, Rr	(X) <= Rr X <= X+1	<i>speichere Rr indirekt</i>	1001 001r rrrr 1101	2
st	Y+, Rr	(Y) <= Rr Y <= Y+1	<i>speichere Rr indirekt</i>	1001 001r rrrr 1001	2
st	Z+, Rr	(Z) <= Rr Z <= Z+1	<i>speichere Rr indirekt</i>	1001 001r rrrr 0001	2

Das Wortregister mit der Speicheradresse wird *vor* der Operation um 1 *vermindert*.

Befehl	Operand	Wirkung		Befehlswort	T
st	-X, Rr	X <= X-1 (X) <= Rr	<i>speichere Rr indirekt</i>	1001 001r rrrr 1110	2
st	-Y, Rr	Y <= Y-1 (Y) <= Rr	<i>speichere Rr indirekt</i>	1001 001r rrrr 1010	2
st	-Z, Rr	Z <= Z-1 (Z) <= Rr	<i>speichere Rr indirekt</i>	1001 001r rrrr 0010	2

Die Speicheradresse im SRAM ergibt sich aus dem vorzeichenlosen Abstand (**D**isplacement) von 1 bis 63 plus dem Inhalt des Wortregisters; das Wortregister bleibt unverändert erhalten.

Befehl	Operand	Wirkung		Befehlswort	T
std	Y+q, Rr	(Y + konst.) <= Rr	<i>speichere Rr indirekt</i>	10q0 qq1r rrrr 1qqq	2
std	Z+q, Rr	(Z + konst.) <= Rr	<i>speichere Rr indirekt</i>	10q0 qq1r rrrr 0qqq	2

Beispiel:

```
; speichere R16 nach SRAM mit der Adresse tab
    ldi    ZL,LOW(tab)  ; ZL <= Low-Adresse
    ldi    ZH,HIGH(tab) ; ZH <= High-Adresse
    st     Z,r16         ; SRAM-Byte durch Z adressiert <= R16
    .DSEG                                ; Datensegment im SRAM
tab:  .BYTE  1           ; 1 Byte reserviert
```

STS - Store Direct to data space

Speichere das Quellregister Rr in das direkt adressierte Byte im SRAM.

sts Adresse, Rr ; SRAM-Byte <= Rr

Als Quelle Rr sind alle Register von R0 bis R31 zugelassen.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

2 Befehlswörter: **1001 001r rrrr 0000** kkkk kkkk kkkk kkkk zwei Takte

```
; speichere R16 nach Speicherbyte auf der Adresse tab
    sts    tab,r16    ; SRAM-Byte direkt adressiert <= R16
    .DSEG                ; Datensegment im SRAM
tab:  .BYTE    1        ; 1 Byte reserviert
```

SUB - Subtract without Carry

Subtrahiere vom Inhalt des Zielregisters Rd den Inhalt des Quellregisters Rr. Der Inhalt des Quellregisters Rr bleibt erhalten; das Zielregister Rd wird mit der Differenz überschrieben.

sub Rd, Rr ; Rd <= Rd - Rr

Als Ziel und als Quelle sind alle Register von R0 bis R31 zugelassen.

I	T	H	S	V	N	Z	C
-	-	<>	<>	<>	<>	<>	<>

Befehlswort: **0001 10rd dddd rrrr** ein Takt

Beispiel:

```
; 8bit Subtraktion R16 <= R16 - R17
    sub    r16,r17    ; subtrahiere Bytes
    brcs   fehler     ; verzweige bei C = 1 Unterlauf
```

SUBI - Subtract Immediate

Subtrahiere vom Inhalt des Zielregisters Rd die 8bit Konstante im zweiten Operanden. Der fehlende Befehl `addi` kann durch Subtraktion der negativen Konstanten ersetzt werden.

```
subi    Rd, K8      ; Rd <= Rd - Konstante
subi    Rd, -K8     ; Rd <= Rd + Konstante      H und C negiert!
```

Als Ziel und als Quelle sind nur die Register von R16 bis R31 zugelassen.

I	T	H	S	V	N	Z	C
-	-	<>	<>	<>	<>	<>	<>

Befehlswort: **0101** KKKK dddd KKKK ein Takt

Beispiel:

```
; subtrahiere von R16 die Konstante $30
    subi    r16,$30    ; subtrahiere Konstante
; addiere zu R16 die Konstante $30 durch Subtraktion der negativen Konstanten
    subi    r16,-$30   ; addiere Konstante Bedingungsbits H und C negiert
```

SWAP - Swap Nibbles

Vertausche die beiden Halfen des Zielregisters Rd.

```
swap    Rd          ; Rd(7:4) <= Rd(3:0)    Rd(3:0) <= Rd(7-4)
```

Als Ziel sind alle Register von R0 bis R31 zugelassen. Die Bedingungsbits werden nicht verandert.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1001 010d** dddd **0010** ein Takt

Beispiel:

```
; packe die beiden BCD-Ziffern in R16 und R17 zusammen
    ldi     r16,$01     ; R16 <= $01 Zehner rechtsbundig
    swap    r16          ; R16 <= $10 Zehner linksbundig
    ldi     r17,$02     ; R17 <= $02 Einer rechtsbundig
    or      r16,r17     ; R16 <= $12 gepackte zweistellige BCD-Zahl
```

TST - Test for Zero or Minus

Der Pseudobefehl wird als logische Operation mit dem Befehl `and Rd, Rd` ausgeführt, der den Inhalt von `Rd` nicht verändert.

tst `Rd` ; teste `Rd` auf Null und Vorzeichen

Als Operanden sind alle Register von `R0` bis `R31` zugelassen.

I	T	H	S	V	N	Z	C
-	-	-	<>	0	<>	<>	-

Befehlswort: **0010 00dd dddd dddd** ein Takt

Beispiel:

```
; abweisende Schleife verhindert 256 Durchläufe für den Wert 0
    in     r16,PIND ; R16 <- Eingabeport
    tst   r16      ; Testbefehl auf Null
    breq   ende    ; bei Null kein Durchlauf
loop: dec  r16     ; sonst bis Null herunterzählen
    brne  loop     ;
ende:                      ;
```

WDR - Watchdog Reset

Der Watchdog Timer wird zurückgesetzt.

wdr ; Watchdog Timer zurücksetzen

Die Bedingungsbits werden nicht verändert.

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Befehlswort: **1001 0101 1010 1000** ein Takt

Beispiel:

```
; Watchdog Timer zurücksetzen
    wdr                      ; Wachhund beruhigt
```