Detailbeschreibung der Befehle der AVR - Mikrocontroller Familie in deutscher Fassung

Originalquelle: http://www.avr-roboter.de/controller/befehle/beschreibung/beschreibung.html

ADC - Add with Carry

Syntax:	ADC Rd, Rr
Funktion:	Addiert zwei Register und den Inhalt des Carry-Flag. Das Ergebnis wird im Quellregister Rd abgelegt.
Operation:	Rd ← Rd + Rr + C
Operanden:	$0 \leq d \leq 31$, $0 \leq r \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0001 11rd dddd rrrr

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	↔	↔	↔	↔	↔	↔

H:	Das H-Flag wird gesetzt, wenn ein Übertrag von Bit 3 auf Bit 4 erfolgte, andernfalls wird das Flag gelöscht.
S:	Das S-Flag wird gesetzt, wenn entweder das N-Flag oder das V-Flag gesetzt ist. Wenn beide Flags gleich sind, wird das S-Flag gelöscht.
V:	Das V-Flag wird gesetzt, wenn aus der Operation ein Zweierkomplement- Überlauf erfolgt, andernfalls wird das Flag gelöscht.
N:	Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag wird gesetzt, wenn im Ergebnis ein Überlauf von Bit 8 erfolgte, andernfalls wird es gelöscht.

Beispiel:

```
add r2,r0    ;Addieren der Low-Bytes
adc r3,r1    ;Addieren der High-Bytes und des Carry-Flags
```

ADD - Add without Carry

Syntax:	ADD Rd, Rr
Funktion:	Addiert zwei Register ohne das Carry-Flag. Das Ergebnis wird im Quellregister Rd abgelegt.
Operation:	Rd ← Rd + Rr
Operanden:	$0 \leq d \leq 31$, $0 \leq r \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0000 11rd dddd rrrr

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	↔	↔	↔	↔	↔	↔

H:	Das H-Flag wird gesetzt, wenn ein Übertrag von Bit 3 auf Bit 4 erfolgte, andernfalls wird das Flag gelöscht.
S:	Das S-Flag wird gesetzt, wenn entweder das N-Flag oder das V-Flag gesetzt ist. Wenn beide Flags gleich sind, wird das S-Flag gelöscht.
V:	Das V-Flag wird gesetzt, wenn aus der Operation ein Zweierkomplement- Überlauf erfolgt, andernfalls wird das Flag gelöscht.
N:	Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag wird gesetzt, wenn im Ergebnis ein Überlauf von Bit 8 erfolgte, andernfalls wird es gelöscht.

Beispiel:

```
add r1,r0    ;Addieren der Register R0 und R1
```

ADIW - Add Immediate to Word

Syntax:	ADIW Rd+1:Rd,K
Funktion:	Addiert eine Konstante zu einem Registerpaar. Das Ergebnis wird in diesem Registerpaar abgelegt. Der Befehl arbeitet nur mit den oberen vier Registerpaaren und unterstützt somit das Arbeiten mit den Pointer-Registern. Der Befehl ist nicht in allen AVR-Bausteinen verfügbar.

Operation:	Rd+1:Rd ← Rd+1:Rd + K
Operanden:	d ε {24, 26, 28, 30}, 0 ≤ K ≤ 63
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	2
16 Bit Operations Code:	1001 0110 KKdd KKKK

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	↔	↔	↔	↔	↔

S:	Das S-Flag wird gesetzt, wenn entweder das N-Flag oder das V-Flag gesetzt ist. Wenn beide Flags gleich sind, wird das S-Flag gelöscht.
V:	Das V-Flag wird gesetzt, wenn aus der Operation ein Zweierkomplement- Überlauf an Bit 15 erfolgt, andernfalls wird das Flag gelöscht.
N:	Das N-Flag wird gesetzt, wenn das MSB (Bit 15) des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 0000h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag wird gesetzt, wenn im Ergebnis ein Überlauf von Bit 15 erfolgte, andernfalls wird es gelöscht.

Beispiel:

```
adiw r25:r24,1    ;Addiert 1 zu dem Registerpaar R25:R24
adiw ZH:ZL,34    ;Addiert 34 zum Z-Pointer (R31:R30)
```

AND - Logical AND

Syntax:	AND Rd, Rr
Funktion:	Führt eine logische UND-Verknüpfung der Inhalte zweier Register durch, das Ergebnis wird im Quellregister Rd abgelegt.
Operation:	Rd ← Rd • Rr
Operanden:	0 ≤ d ≤ 31, 0 ≤ r ≤ 31
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0010 00rd dddd rrrr

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	↔	0	↔	↔	-

S: Das S-Flag wird gesetzt, wenn entweder das N-Flag oder das V-Flag gesetzt ist. Wenn beide Flags gleich sind, wird das S-Flag gelöscht.

V: Das V-Flag wird gelöscht.

N: Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.

Z: Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.

Beispiel:

```
and r1, r0    ;UND-Verknüpfung der Register R0 und R1

oder:

ldi r16,$01    ;Maskierungsbyte 0000 0001 in R16 schreiben
and r2,r16    ;Löschen der oberen 7 Bit in R2
```

ANDI - Logical AND with Immediate

Syntax:	ANDI Rd,K
Funktion:	Führt eine logische UND-Verknüpfung zwischen einem Register und einer Konstante durch, das Ergebnis wird im Quellregister Rd abgelegt. Für diesen Befehl können nur die Register R16 bis R31 verwendet werden.
Operation:	Rd ← Rd • K
Operanden:	$16 \leq d \leq 31, 0 \leq K \leq 255$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0111 KKKK dddd KKKK

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	↔	0	↔	↔	-

S: Das S-Flag wird gesetzt, wenn entweder das N-Flag oder das V-Flag gesetzt ist. Wenn beide Flags gleich sind, wird das S-Flag gelöscht.

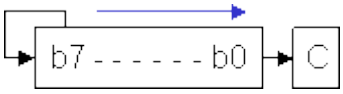
V: Das V-Flag wird gelöscht.

N:	Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.

Beispiel:

```
andi r1,$0F    ;Löschen der oberen 4 Bit in R1
```

ASR - Arithmetic Shift Right

Syntax:	ASR Rd
Funktion:	Mit diesem Befehl werden alle Bits des Registers Rd eine Position nach rechts geschoben. Das Bit 7 verändert sich dabei nicht, das Bit 0 wird in das Carry-Flag geschoben. Damit stellt dieser Befehl eine Division durch 2 dar, wobei sich das Vorzeichen nicht verändert. Das Carry-Flag kann dazu benutzt werden, um das Ergebnis zu runden.
Operation:	
Operanden:	$0 \leq d \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 010d dddd 0101

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	↔	↔	↔	↔	↔

S:	Das S-Flag wird gesetzt, wenn entweder das N-Flag oder das V-Flag gesetzt ist. Wenn beide Flags gleich sind, wird das S-Flag gelöscht.
V:	Das V-Flag wird gesetzt, wenn entweder das N-Flag oder das C-Flag gesetzt ist. Wenn beide Bits gleich sind, wird das V-Flag gelöscht.
N:	Das N-Flag wird gesetzt, wenn das MSB (Bit 7) des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag wird gesetzt, wenn das Bit 0 im Register Rd vor dem Schieben gesetzt war. War das Bit 0 vorher gelöscht, so wird auch das Carry-Flag gelöscht.

Beispiel:

```
ldi r16,$10    ;Die Dezimalzahl 16 in R16 laden
asr r16        ;R16 = R16/2 = 8

oder:

ldi r16,$FC    ;Die Dezimalzahl -4 in R16 laden
asr r16     ;R16 = R16/2 = -2
```

BCLR - Bit Clear in SREG

Syntax:	BCLR s
Funktion:	Löscht einzelne Flags im Status-Register SREG.
Operation:	SREG(s) ← 0
Operanden:	0 ≤ s ≤ 7
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 0100 1sss 1000

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
↔	↔	↔	↔	↔	↔	↔	↔

I: Wird mit dem Befehl BCLR 7 gelöscht, bei anderen BCLR Befehlen bleibt das Flag unverändert.

T: Wird mit dem Befehl BCLR 6 gelöscht, bei anderen BCLR Befehlen bleibt das Flag unverändert.

H: Wird mit dem Befehl BCLR 5 gelöscht, bei anderen BCLR Befehlen bleibt das Flag unverändert.

S: Wird mit dem Befehl BCLR 4 gelöscht, bei anderen BCLR Befehlen bleibt das Flag unverändert.

V: Wird mit dem Befehl BCLR 3 gelöscht, bei anderen BCLR Befehlen bleibt das Flag unverändert.

N: Wird mit dem Befehl BCLR 2 gelöscht, bei anderen BCLR Befehlen bleibt das Flag unverändert.

Z: Wird mit dem Befehl BCLR 1 gelöscht, bei anderen BCLR Befehlen bleibt das Flag unverändert.

C:	Wird mit dem Befehl BCLR 0 gelöscht, bei anderen BCLR Befehlen bleibt das Flag unverändert.

Beispiel:

```
bclr 0    ;Löscht das Carry-Flag

bclr 7    ;Sperrt alle Interrupts
```

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

Syntax:	BLD Rd,b
Funktion:	Kopiert das T Flag aus dem Status-Register (SREG) in das Bit b im Register Rd.
Operation:	Rd(b) ← T
Operanden:	0 ≤ d ≤ 31, 0 ≤ b ≤ 7
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1111 100d dddd 0bbb

Flags im Status-Register (SREG):

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

Beispiel:

```
bst r1,3    ;Speichert Bit 3 aus R1 in das T-Flag

bld r0,4    ;Speichert das T-Flag in das Bit 4 von R0
```

BRBC - Branch if Bit in SREG is Cleared

Syntax:	BRBC s,k
Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet ein einzelnes Bit im Status-Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Bit gelöscht ist. Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben.
Operation:	(a) Wenn SREG(s) = 0, PC ← PC + k + 1 (b) Wenn SREG(s) = 1, PC ← PC + 1

Operanden:	-64 ≤ k ≤ +63, 0 ≤ s ≤ 7
Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations Code:	1111 01kk kkkk ksss

Flags im Status-Register (SREG):

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

Beispiel:

```
    bclr 1        ;Löschen des Z-Flags
    brbc 1,null    ;Verzweigen zum Label null, wenn Z-Flag gelöscht.
    ...

null:    nop        ;Verzweigungsziel, Leerbefehl ausführen...
```

BRBS - Branch if Bit in SREG is Set

Syntax:	BRBS s,k
Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet ein einzelnes Bit im Status-Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Bit gesetzt ist. Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben.
Operation:	(a) Wenn SREG(s) = 1, PC ← PC + k + 1 (b) Wenn SREG(s) = 0, PC ← PC + 1
Operanden:	-64 ≤ k ≤ +63, 0 ≤ s ≤ 7
Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations Code:	1111 00kk kkkk ksss

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C

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

Beispiel:

```
    bset 0        ;Setzt das Carry-Flag
    brbs 0,eins    ;Verzweigen zum Label eins, wenn C-Flag gesetzt.
    ...

eins:     nop        ;Verzweigungsziel, Leerbefehl ausführen...
```

BRCC - Branch if Carry Cleared

Syntax:	BRCC k
Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet das Carry-Flag C im Status-Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Flag gelöscht ist. Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben. Dieser Befehl entspricht dem Befehl BRBC 0,k.
Operation:	(a) Wenn C = 0, PC ← PC + k + 1 (b) Wenn C = 1, PC ← PC + 1
Operanden:	-64 ≤ k ≤ +63
Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations Code:	1111 01kk kkkk k000

Flags im Status-Register (SREG):

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

Beispiel:

```
    bclr 0        ;Löschen des C-Flags
    brcc null    ;Verzweigen zum Label null, wenn C-Flag gelöscht.
    ...

null:    nop        ;Verzweigungsziel, Leerbefehl ausführen...
```

BRCS - Branch if Carry Set

Syntax:	BRCS k

Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet das Carry-Flag C im Status-Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Flag gesetzt ist. Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben. Dieser Befehl entspricht dem Befehl BRBS 0,k.
Operation:	(a) Wenn C = 1, PC ← PC + k + 1 (b) Wenn C = 0, PC ← PC + 1
Operanden:	-64 ≤ k ≤ +63
Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations Code:	1111 00kk kkkk k000

Flags im Status-Register (SREG):

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

Beispiel:

```
    bset 0        ;Setzt das Carry-Flag
    brcs eins     ;Verzweigen zum Label eins, wenn C-Flag gesetzt.
    ...

eins:     nop        ;Verzweigungsziel, Leerbefehl ausführen...
```

BREAK - Break

Syntax:	BREAK
Funktion:	Dieser Befehl wird nur im Zusammenhang mit dem On-Chip Debug System benutzt und kommt eigentlich im normalen Anwenderprogramm nicht vor. Wenn ein BREAK-Befehl ausgeführt wird, wird die AVR-CPU in den Stop-Modus gesetzt, so dass der On-Chip- Debugger Zugriff auf die internen Ressourcen bekommt. Wenn eines der Lock-Bits gesetzt ist oder eine der beiden Fuses JTAGEN oder OCDEN unprogrammiert ist, dann wird die CPU den BREAK-Befehl wie einen NOP-Befehl betrachten und nicht in den Stop-Modus gehen. Der BREAK-Befehl ist nicht in allen AVR-Mikrocontrollern verfügbar.
Operation:	On-Chip Debug System break
Operanden:	keine
Programmzähler:	PC ← PC + 1

Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 0101 1001 1000

Flags im Status-Register (SREG):

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

BREQ - Branch if Equal

Syntax:	BREQ k
Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet das Zero-Flag Z im Status-Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Flag gesetzt ist. Wenn dieser Befehl unmittelbar auf einen der Befehle CP, CPI, SUB oder SUBI folgt, wird genau dann verzweigt, wenn der Wert in den Registern Rd und Rr bei den genannten Befehlen gleich war, das Vorzeichen der Werte in Rd und Rr spielt dabei keine Rolle. Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben. Dieser Befehl entspricht dem Befehl BRBS 1,k.
Operation:	(a) Wenn Z = 1 (Rd = Rr), PC ← PC + k + 1 (b) Wenn Z = 0, PC ← PC + 1
Operanden:	-64 ≤ k ≤ +63
Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations Code:	1111 00kk kkkk k001

Flags im Status-Register (SREG):

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

Beispiel:

```
    cp r0,r1    ;Vergleich der Register R0 und R1
    breq gleich    ;Verzweigen zum Label gleich, wenn R0 = R1
    ...

gleich:    nop        ;Verzweigungsziel, Leerbefehl ausführen...
```

BRGE - Branch if Greater or Equal (Signed)

Syntax:	BRGE k
Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet das Signed-Flag S im Status- Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Flag gelöscht ist. Wenn dieser Befehl unmittelbar auf einen der Befehle CP, CPI, SUB oder SUBI folgt, wird genau dann verzweigt, wenn der vorzeichenbehaftete Wert im Register Rd größer oder gleich dem vorzeichenbehafteten Wert im Register Rr bei den genannten Befehlen war. Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben. Dieser Befehl entspricht dem Befehl BRBC 4,k.
Operation:	(a) Wenn S = 0 (Rd ≥ Rr), PC ← PC + k + 1 (b) Wenn S = 1, PC ← PC + 1
Operanden:	-64 ≤ k ≤ +63
Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations Code:	1111 01kk kkkk k100

Flags im Status-Register (SREG):

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

Beispiel:

```
    cp r1,r2    ;Vergleichen der Register R1 und R2
    brge grgleich    ;Verzweigen zum Label grgleich, wenn R1 ≥ R2.
    ...

grgleich:    nop    ;Verzweigungsziel, Leerbefehl ausführen...
```

BRHC - Branch if Half-Carry-Flag is Cleared

Syntax:	BRHC k
Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet das Half-Carry-Flag H im Status- Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Flag gelöscht ist. Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben. Dieser Befehl entspricht dem Befehl BRBC 5,k.

Operation:	(a) Wenn H = 0, PC ← PC + k + 1 (b) Wenn H = 1, PC ← PC + 1
Operanden:	-64 ≤ k ≤ +63
Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations Code:	1111 01kk kkkk k101

Flags im Status-Register (SREG):

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

Beispiel:

```
    bclr 5         ;Löschen des H-Flags
    brhc null     ;Verzweigen zum Label null, wenn das H-Flag gelöscht ist.
    ...
null:    nop         ;Verzweigungsziel, Leerbefehl ausführen...
```

BRHS - Branch if Half-Carry is Set

Syntax:	BRHS k
Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet das Half-Carry-Flag H im Status- Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Flag gesetzt ist. Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben. Dieser Befehl entspricht dem Befehl BRBS 5,k.
Operation:	(a) Wenn H = 1, PC ← PC + k + 1 (b) Wenn H = 0, PC ← PC + 1
Operanden:	-64 ≤ k ≤ +63
Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations Code:	1111 00kk kkkk k101

Flags im Status-Register (SREG):

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

Beispiel:

```
    bset 5        ;Setzt das Half-Carry-Flag
    brhs eins     ;Verzweigen zum Label eins, wenn das H-Flag gesetzt ist.
    ...

eins: nop         ;Verzweigungsziel, Leerbefehl ausführen...
```

BRID - Branch if Global Interrupt is Disabled

Syntax:	BRID k
Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet das Global Interrupt-Flag I im Status-Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Flag gelöscht ist (Flag gelöscht bedeutet, dass die Interrupts global gesperrt sind). Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben. Dieser Befehl entspricht dem Befehl BRBC 7,k.
Operation:	(a) Wenn I = 0, PC ← PC + k + 1 (b) Wenn I = 1, PC ← PC + 1
Operanden:	-64 ≤ k ≤ +63
Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations Code:	1111 01kk kkkk k111

Flags im Status-Register (SREG):

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

Beispiel:

```
    bclr 7        ;Löschen des I-Flags
    brid null     ;Verzweigen zum Label null, wenn das I-Flag gelöscht ist.
    ...

null: nop         ;Verzweigungsziel, Leerbefehl ausführen...
```

BRIE - Branch if Global Interrupt is Enabled

Syntax:	BRIE k
Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet das Global Interrupt-Flag I im Status-Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Flag gesetzt ist (Flag gesetzt bedeutet, dass die Interrupts global freigegeben sind). Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben. Dieser Befehl entspricht dem Befehl BRBS 7,k.
Operation:	(a) Wenn I = 1, PC ← PC + k + 1 (b) Wenn I = 0, PC ← PC + 1
Operanden:	-64 ≤ k ≤ +63
Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations Code:	1111 00kk kkkk k111

Flags im Status-Register (SREG):

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

Beispiel:

```
    bset 7         ;Setzt das I-Flag
    brie eins     ;Verzweigen zum Label eins, wenn das I-Flag gesetzt ist.
    ...

eins: nop         ;Verzweigungsziel, Leerbefehl ausführen...
```

BRLO - Branch if Lower (Unsigned)

Syntax:	BRLO k
Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet das Carry-Flag C im Status-Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Flag gesetzt ist. Wenn dieser Befehl unmittelbar auf einen der Befehle CP, CPI, SUB oder SUBI folgt, wird genau dann verzweigt, wenn der Wert im Register Rd kleiner als der Wert im Register Rr bei den genannten Befehlen war. Das Vorzeichen der Werte in Rd und Rr spielt dabei keine Rolle. Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben. Dieser Befehl entspricht dem Befehl BRBS 0,k.

Operation:	(a) Wenn C = 1 (Rd < Rr), PC ← PC + k + 1 (b) Wenn C = 0, PC ← PC + 1
Operanden:	-64 ≤ k ≤ +63
Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations Code:	1111 00kk kkkk k000

Flags im Status-Register (SREG):

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

Beispiel:

```
    cp r0,r1    ;Vergleich der Register R0 und R1
    brlo kleiner    ;Verzweigen zum Label kleiner, wenn R0 < R1 ist.
    ...

kleiner: nop        ;Verzweigungsziel, Leerbefehl ausführen...
```

BRLT - Branch if Less Than (Signed)

Syntax:	BRLT k
Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet das Signed-Flag S im Status- Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Flag gesetzt ist. Wenn dieser Befehl unmittelbar auf einen der Befehle CP, CPI, SUB oder SUBI folgt, wird genau dann verzweigt, wenn der vorzeichenbehaftete Wert im Register Rd kleiner als der vorzeichenbehafteten Wert im Register Rr bei den genannten Befehlen war. Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben. Dieser Befehl entspricht dem Befehl BRBS 4,k.
Operation:	(a) Wenn S = 1 (Rd < Rr), PC ← PC + k + 1 (b) Wenn S = 0, PC ← PC + 1
Operanden:	-64 ≤ k ≤ +63
Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations	1111 00kk kkkk k100

Code:

Flags im Status-Register (SREG):

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

Beispiel:

```
    cp r1,r2    ;Vergleichen der Register R1 und R2
    brlt kleiner    ;Verzweigen zum Label kleiner, wenn R1 < R2 ist.
    ...

kleiner: nop        ;Verzweigungsziel, Leerbefehl ausführen...
```

BRMI - Branch if Minus

Syntax:	BRMI k
Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet das Negativ-Flag N im Status- Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Flag gesetzt ist. Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben. Dieser Befehl entspricht dem Befehl BRBS 2,k.
Operation:	(a) Wenn N = 1, PC ← PC + k + 1 (b) Wenn N = 0, PC ← PC + 1
Operanden:	-64 ≤ k ≤ +63
Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations Code:	1111 00kk kkkk k010

Flags im Status-Register (SREG):

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

Beispiel:

```
    bset 2        ;Setzt das N-Flag
    brmi eins    ;Verzweigen zum Label eins, wenn das N-Flag gesetzt ist.
    ...

eins: nop        ;Verzweigungsziel, Leerbefehl ausführen...
```

BRNE - Branch if Not Equal

Syntax:	BRNE k
Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet das Zero-Flag Z im Status-Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Flag gelöscht ist. Wenn dieser Befehl unmittelbar auf einen der Befehle CP, CPI, SUB oder SUBI folgt, wird genau dann verzweigt, wenn der Wert im Register Rd ungleich dem Wert im Register Rr bei den genannten Befehlen war. Das Vorzeichen der Werte in Rd und Rr spielt dabei keine Rolle. Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben. Dieser Befehl entspricht dem Befehl BRBC 1,k.
Operation:	(a) Wenn Z = 0 (Rd ≠ Rr), PC ← PC + k + 1 (b) Wenn Z = 1, PC ← PC + 1
Operanden:	-64 ≤ k ≤ +63
Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations Code:	1111 01kk kkkk k001

Flags im Status-Register (SREG):

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

Beispiel:

```
    cp r1,r2    ;Vergleichen der Register R1 und R2
    brne ungleich    ;Verzweigen zum Label ungleich, wenn R1 ≠ R2 ist.
    ...

ungleich: nop        ;Verzweigungsziel, Leerbefehl ausführen...
```

BRPL - Branch if Plus

Syntax:	BRPL k
Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet das Negativ-Flag N im Status- Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Flag gelöscht ist. Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben. Dieser Befehl entspricht dem Befehl BRBC 2,k.
Operation:	(a) Wenn N = 0, PC ← PC + k + 1

	(b) Wenn N = 1, PC ← PC + 1
Operanden:	-64 ≤ k ≤ +63
Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations Code:	1111 01kk kkkk k010

Flags im Status-Register (SREG):

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

Beispiel:

```
    bclr 2         ;Löscht das N-Flag
    brpl null     ;Verzweigen zum Label null, wenn das N-Flag gelöscht ist.
    ...

null: nop         ;Verzweigungsziel, Leerbefehl ausführen...
```

BRSH - Branch if Same or Higher (Unsigned)

Syntax:	BRSH k
Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet das Carry-Flag C im Status-Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Flag gelöscht ist. Wenn dieser Befehl unmittelbar auf einen der Befehle CP, CPI, SUB oder SUBI folgt, wird genau dann verzweigt, wenn der Wert im Register Rd größer oder gleich dem Wert im Register Rr bei den genannten Befehlen war. Das Vorzeichen der Werte in Rd und Rr spielt dabei keine Rolle. Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben. Dieser Befehl entspricht dem Befehl BRBC 0,k.
Operation:	(a) Wenn C = 0 (Rd ≥ Rr), PC ← PC + k + 1 (b) Wenn C = 1, PC ← PC + 1
Operanden:	-64 ≤ k ≤ +63
Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations	1111 01kk kkkk k000

Code:

Flags im Status-Register (SREG):

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

Beispiel:

```
    cp r0,r1    ;Vergleich der Register R0 und R1
    brsh grgleich    ;Verzweigen zum Label grgleich, wenn R0 ≥ R1 ist.
    ...

grgleich: nop        ;Verzweigungsziel, Leerbefehl ausführen...
```

BRTC - Branch if T-Flag is Cleared

Syntax:	BRTC k
Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet das T-Flag im Status-Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Flag gelöscht ist. Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben. Dieser Befehl entspricht dem Befehl BRBC 6,k.
Operation:	(a) Wenn T = 0, PC ← PC + k + 1 (b) Wenn T = 1, PC ← PC + 1
Operanden:	-64 ≤ k ≤ +63
Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations Code:	1111 01kk kkkk k110

Flags im Status-Register (SREG):

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

Beispiel:

```
    bclr 6        ;Löschen des T-Flags
    brtc null    ;Verzweigen zum Label null, wenn das T-Flag gelöscht ist.
    ...

null: nop        ;Verzweigungsziel, Leerbefehl ausführen...
```

BRTS - Branch if T-Flag is Set

Syntax:	BRTS k
Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet das T-Flag im Status-Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Flag gesetzt ist. Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben. Dieser Befehl entspricht dem Befehl BRBS 6,k.
Operation:	(a) Wenn T = 1, PC ← PC + k + 1 (b) Wenn T = 0, PC ← PC + 1
Operanden:	-64 ≤ k ≤ +63
Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations Code:	1111 00kk kkkk k110

Flags im Status-Register (SREG):

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

Beispiel:

```
    bset 6        ;Setzt das T-Flag
    brts eins     ;Verzweigen zum Label eins, wenn das T-Flag gesetzt ist.
    ...

eins: nop         ;Verzweigungsziel, Leerbefehl ausführen...
```

BRVC - Branch if Overflow Cleared

Syntax:	BRVC k
Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet das Overflow-Flag V im Status- Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Flag gelöscht ist. Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben. Dieser Befehl entspricht dem Befehl BRBC 3,k.
Operation:	(a) Wenn V = 0, PC ← PC + k + 1 (b) Wenn V = 1, PC ← PC + 1
Operanden:	-64 ≤ k ≤ +63

Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations Code:	1111 01kk kkkk k011

Flags im Status-Register (SREG):

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

Beispiel:

```
    bclr 3        ;Löschen des V-Flags
    brvc null     ;Verzweigen zum Label null, wenn das V-Flag gelöscht ist.
    ...

null: nop         ;Verzweigungsziel, Leerbefehl ausführen...
```

BRVS - Branch if Overflow Set

Syntax:	BRVS k
Funktion:	Bedingte relative Verzweigung. Dieser Befehl testet das Overflow-Flag V im Status- Register SREG und verzweigt zu einer relativ zum aktuellen PC angegebenen Adresse, wenn das Flag gesetzt ist. Die Verzweigung kann relativ zum aktuellen PC in beide Richtungen erfolgen und im Bereich zwischen -63 und +64 liegen. Der Parameter k wird als Zweierkomplement angegeben. Dieser Befehl entspricht dem Befehl BRBS 3,k.
Operation:	(a) Wenn V = 1, PC ← PC + k + 1 (b) Wenn V = 0, PC ← PC + 1
Operanden:	-64 ≤ k ≤ +63
Programmzähler:	(a) PC ← PC + k + 1 (b) PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	(a) 2 (b) 1
16 Bit Operations Code:	1111 00kk kkkk k011

Flags im Status-Register (SREG):

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

Beispiel:

```
    bset 3        ;Setzt das V-Flag
    brvs eins     ;Verzweigen zum Label eins, wenn das V-Flag gesetzt ist.
    ...

eins: nop         ;Verzweigungsziel, Leerbefehl ausführen...
```

BSET - Bit Set in SREG

Syntax:	BSET s
Funktion:	Setzen eines einzelnen Flags im Status-Register SREG.
Operation:	SREG(s) ← 1
Operanden:	$0 \leq s \leq 7$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 0100 0sss 1000

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
↔	↔	↔	↔	↔	↔	↔	↔

I:	Wird mit dem Befehl BSET 7 gesetzt, bei anderen BSET Befehlen bleibt das Flag unverändert.
T:	Wird mit dem Befehl BSET 6 gesetzt, bei anderen BSET Befehlen bleibt das Flag unverändert.
H:	Wird mit dem Befehl BSET 5 gesetzt, bei anderen BSET Befehlen bleibt das Flag unverändert.
S:	Wird mit dem Befehl BSET 4 gesetzt, bei anderen BSET Befehlen bleibt das Flag unverändert.
V:	Wird mit dem Befehl BSET 3 gesetzt, bei anderen BSET Befehlen bleibt das Flag unverändert.
N:	Wird mit dem Befehl BSET 2 gesetzt, bei anderen BSET Befehlen bleibt das Flag unverändert.
Z:	Wird mit dem Befehl BSET 1 gesetzt, bei anderen BSET Befehlen bleibt das Flag unverändert.

C:	Wird mit dem Befehl BSET 0 gesetzt, bei anderen BSET Befehlen bleibt das Flag unverändert.

Beispiel:

```
bset 0    ;Setzt das Carry-Flag

bset 7    ;Gibt alle Interrupts frei
```

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

Syntax:	BST Rd,b
Funktion:	Speichert das Bit b aus dem Register Rd in das T-Flag im Status-Register SREG.
Operation:	T ← Rd(b)
Operanden:	$0 \leq d \leq 31$, $0 \leq b \leq 7$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1111 101d dddd 0bbb

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	↔	-	-	-	-	-	-

T:	Das T-Flag wird gelöscht, wenn das Bit b im Register Rd gleich 0 ist, andernfalls wird es gesetzt.

Beispiel:

```
bst r1,2    ;Speichert das Bit 2 aus R1 in das T-Flag.
bld r0,4    ;Speichert das T-Flag in Bit 4 des Registers R0.
```

CALL - Long Call to a Subroutine

Syntax:	CALL k
Funktion:	Ruft ein Unterprogramm innerhalb des zur Verfügung stehenden Programmspeichers auf. Die Rücksprungadresse (aktueller PC + 2), die den Befehl nach der Call-Anweisung enthält, wird in den Stack gesichert (siehe auch RCALL). Dieser Befehl ist nicht in allen AVR-Mikrocontrollern verfügbar.
Operation:	(a) PC ← k, Bausteine mit 16 Bit PC (128k) (b) PC ← k, Bausteine mit 22 Bit PC (8M)
Operanden:	(a) 0 ≤ k ≤ 64k, STACK ← PC+2, SP ← SP-2 (b) 0 ≤ k ≤ 4M, STACK ← PC+2, SP ← SP-3
Programmzähler:	PC ← k
Words:	2 (4 Byte)
Zyklen:	(a) 4 (b) 5
32 Bit Operations Code:	1001 010k kkkk 111k kkkk kkkk kkkk kkkk

Flags im Status-Register (SREG):

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

Beispiel:

```
    call ausgabe    ;Aufruf des Unterprogramms ausgabe
    nop         ;Leerbefehl, der nach dem Rücksprung ausgeführt wird.
    ...

ausgabe: add r0,r1    ;Beginn des Unterprogramms ausgabe
    ...
    ret         ;Rücksprung aus dem Unterprogramm ins Hauptprogramm
```

CBI - Clear Bit in I/O-Register

Syntax:	CBI A,b
Funktion:	Dieser Befehl löscht ein einzelnes Bit in einem der I/O-Register. Mit CBI können allerdings nur die Bits in den unteren 32 I/O-Registern gelöscht werden (Adresse 0 bis 31).
Operation:	I/O(A,b) ← 0
Operanden:	$0 \leq A \leq 31$, $0 \leq b \leq 7$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 1000 AAAA Abbb

Flags im Status-Register (SREG):

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

Beispiel:

```
cbi $12,7    ;Löscht das Bit 7 in Port D
```

CBR - Clear Bits in Register

Syntax:	CBR Rd,K
Funktion:	Dieser Befehl löscht ein oder mehrer Bits in einem Register. Dabei führt er eine logische UND-Verknüpfung zwischen dem Register Rd und dem Komplement der Konstante K durch. Mit CBR können allerdings nur die Bits in den oberen 32 Registern gelöscht werden (R16 bis R31).
Operation:	Rd ← RD • (FFh - K)
Operanden:	$16 \leq d \leq 31$, $0 \leq K \leq 255$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0111 KKKK dddd KKKK

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	↔	0	↔	↔	-

S:	Das S-Flag wird gesetzt, wenn entweder das N-Flag oder das V-Flag gesetzt ist. Wenn beide Flags gleich sind, wird das S-Flag gelöscht.

V:	Das V-Flag wird gelöscht.
N:	Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.

Beispiel:

```
cbr r16,$F0    ;Löscht die oberen 4 Bit im Register R16

cbr r18,1    ;Löscht das Bit 0 im Register R18
```

CLC - Clear Carry-Flag

Syntax:	CLC
Funktion:	Dieser Befehl löscht das Carry-Flag C im Status-Register SREG.
Operation:	C ← 0
Operanden:	keine
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 0100 1000 1000

Flags im Status-Register (SREG):

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

C:	Das C-Flag wird gelöscht.

Beispiel:

```
sec    ;Setzt das Carry-Flag
clc    ;Löscht das Carry-Flag
```

CLH - Clear Half-Carry-Flag

Syntax:	CLH
Funktion:	Dieser Befehl löscht das Half-Carry-Flag H im Status-Register SREG.
Operation:	H ← 0
Operanden:	keine

Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 0100 1001 1000

Flags im Status-Register (SREG):

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

H: Das H-Flag wird gelöscht.

Beispiel:

```
seh    ;Setzt das Half-Carry-Flag
clh    ;Löscht das Half-Carry-Flag
```

CLI - Clear Global Interrupt-Flag

Syntax:	CLI
Funktion:	Dieser Befehl löscht das Global Interrupt-Flag I im Status-Register SREG. Damit werden alle Interrupts sofort gesperrt, so dass keine weiteren Interrupts ausgeführt werden, selbst wenn der Interrupt zeitgleich mit dem CLI Befehl auftritt.
Operation:	I ← 0
Operanden:	keine
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 0100 1111 1000

Flags im Status-Register (SREG):

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

I: Das I-Flag wird gelöscht.

Beispiel:

```
in temp, SREG    ;SREG in Variable temp zwischenspeichern
cli        ;Alle Interrupts sperren
sbi EECR, EEMWE    ;Beginn Schreiben EEPROM
```

CLN - Clear Negativ-Flag

Syntax:	CLN
Funktion:	Dieser Befehl löscht das Negativ-Flag N im Status-Register SREG.
Operation:	N ← 0
Operanden:	keine
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 0100 1010 1000

Flags im Status-Register (SREG):

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

N: Das N-Flag wird gelöscht.

Beispiel:

```
sen    ;Setzt das Negativ-Flag
cln    ;Löscht das Negativ-Flag
```

CLR - Clear Register

Syntax:	CLR Rd
Funktion:	Dieser Befehl löscht das Register Rd, indem eine Exclusiv-Oder - Operation zwischen dem Register und sich selbst durchgeführt wird. Dadurch werden alle Bits in dem Register gelöscht.
Operation:	C ← 0 Rd ← Rd ⊕ Rd
Operanden:	0 ≤ d ≤ 31
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0010 01dd dddd dddd

Flags im Status-Register (SREG):

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

S: Das S-Flag wird gelöscht.

V: Das V-Flag wird gelöscht.

N: Das N-Flag wird gelöscht.

Z: Das Z-Flag wird gesetzt.

Beispiel:

```
clr r18    ;Alle Bits im Register R18 werden gelöscht.
```

CLS - Clear Signed-Flag

Syntax:	CLS
Funktion:	Dieser Befehl löscht das Signed-Flag S im Status-Register SREG.
Operation:	S ← 0
Operanden:	keine
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 0100 1100 1000

Flags im Status-Register (SREG):

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

S: Das S-Flag wird gelöscht.

Beispiel:

```
ses    ;Setzt das Signed-Flag
cls    ;Löscht das Signed-Flag
```

CLT - Clear T-Flag

Syntax:	CLT
Funktion:	Dieser Befehl löscht das T-Flag im Status-Register SREG.
Operation:	T ← 0
Operanden:	keine
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1

16 Bit Operations Code:	1001 0100 1110 1000

Flags im Status-Register (SREG):

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

T: Das T-Flag wird gelöscht.

Beispiel:

```
set    ;Setzt das T-Flag
clt    ;Löscht das T-Flag
```

CLV - Clear Overflow-Flag

Syntax:	CLV
Funktion:	Dieser Befehl löscht das Overflow-Flag V im Status Register SREG.
Operation:	V ← 0
Operanden:	keine
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 0100 1011 1000

Flags im Status-Register (SREG):

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

V: Das V-Flag wird gelöscht.

Beispiel:

```
sev    ;Setzt das Overflow-Flag
clv    ;Löscht das Overflow-Flag
```

CLZ - Clear Zero-Flag

Syntax:	CLZ
Funktion:	Dieser Befehl löscht das Zero-Flag Z im Status-Register SREG.
Operation:	Z ← 0
Operanden:	keine

Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 0100 1001 1000

Flags im Status-Register (SREG):

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

Z: Das Z-Flag wird gelöscht.

Beispiel:

```
sez    ;Setzt das Zero-Flag
clz    ;Löscht das Zero-Flag
```

COM - One's Complement

Syntax:	COM Rd
Funktion:	Dieser Befehl bildet das Einerkomplement des Registers Rd (Zustand aller Bits wechselt). Das Ergebnis wird im Register Rd abgelegt.
Operation:	Rd ← FFh - Rd
Operanden:	$0 \leq d \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 010d dddd 0000

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	↔	0	↔	↔	1

S: Das S-Flag wird gesetzt, wenn das N-Flag gesetzt ist, andernfalls wird das S-Flag gelöscht.

V: Das V-Flag wird gelöscht.

N: Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.

Z: Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.

C: Das C-Flag wird gesetzt.

Beispiel:

```
com r1    ;Bildet das Einerkomplement von R1
```

CP - Compare

Syntax:	CP Rd,Rr
Funktion:	Dieser Befehl vergleicht die beiden Register Rd und Rr miteinander, indem er die Inhalte beider Register subtrahiert. Die Inhalte beider Register bleiben dabei unverändert. Alle bedingten Verzweigungen können nach diesem Befehl ausgeführt werden.
Operation:	Rd - Rr
Operanden:	$0 \leq d \leq 31$, $0 \leq r \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0001 01rd dddd rrrr

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	↔	↔	↔	↔	↔	↔

H: Das H-Flag wird gesetzt, wenn bei der Subtraktion ein Übertrag von Bit 3 erfolgt, andernfalls wird es gelöscht.

S: Das S-Flag wird gesetzt, wenn entweder das V-Flag oder das N-Flag gesetzt ist. Wenn beide Bits gleich sind, wird das S-Flag gelöscht.

V: Das V-Flag wird gesetzt, wenn aus der Operation ein Zweierkomplement- Überlauf erfolgt, andernfalls wird das Flag gelöscht.

N: Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.

Z: Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.

C: Das C-Flag wird gesetzt, wenn der absolute Wert von Rr größer ist als der absolute Wert von Rd, andernfalls wird das C-Flag gelöscht.

Beispiel:

```
cp r3,r20    ;Vergleicht die Register R3 und R20 miteinander.
```

```
    brne ungleich    ;Verzweigung zum Label ungleich, wenn R3 ≠ R20 ist.
    ...

ungleich: nop        ;Ziel der Verzweigung, Leerbefehl ausführen
```

CPC - Compare with Carry

Syntax:	CPC Rd,Rr
Funktion:	Dieser Befehl vergleicht die beiden Register Rd und Rr miteinander, indem er die Inhalte beider Register und das Carry subtrahiert. Die Inhalte beider Register bleiben dabei unverändert. Alle bedingten Verzweigungen können nach diesem Befehl ausgeführt werden.
Operation:	Rd - Rr - C
Operanden:	$0 \leq d \leq 31$, $0 \leq r \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0000 01rd dddd rrrr

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	↔	↔	↔	↔	↔	↔

H:	Das H-Flag wird gesetzt, wenn bei der Subtraktion ein Übertrag von Bit 3 erfolgt, andernfalls wird es gelöscht.
S:	Das S-Flag wird gesetzt, wenn entweder das V-Flag oder das N-Flag gesetzt ist. Wenn beide Bits gleich sind, wird das S-Flag gelöscht.
V:	Das V-Flag wird gesetzt, wenn aus der Operation ein Zweierkomplement- Überlauf erfolgt, andernfalls wird das Flag gelöscht.
N:	Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag wird gesetzt, wenn der absolute Wert von Rr plus dem Carry größer ist als der absolute Wert von Rd, andernfalls wird das C-Flag gelöscht.

Beispiel:

```
Vergleich zweier 16-Bit Register (R3:R2) und (R1:R0)

    cp r2,r0    ;Vergleicht die Low-Bytes der Register miteinander.
    cpc r3,r1    ;Vergleicht die High-Bytes der Register miteinander
```

```
    brne ungleich    ;Verzweigung zum Label ungleich, wenn R3:R2 ≠ R1:R0 ist.

ungleich: nop        ;Ziel der Verzweigung, Leerbefehl
```

CPI - Compare with Immediate

Syntax:	CPI Rd,K
Funktion:	Dieser Befehl vergleicht das Register Rd mit einer Konstanten, indem er die Konstante vom Register subtrahiert. Der Inhalt des Registers bleibt dabei unverändert. Der Befehl kann nur mit den oberen 32 Register R16 bis R31 durchgeführt werden. Alle bedingten Verzweigungen können nach diesem Befehl ausgeführt werden.
Operation:	Rd - K
Operanden:	$16 \leq d \leq 31$, $0 \leq K \leq 255$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0011 KKKK dddd KKKK

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	↔	↔	↔	↔	↔	↔

H:	Das H-Flag wird gesetzt, wenn bei der Subtraktion ein Übertrag von Bit 3 erfolgt, andernfalls wird es gelöscht.
S:	Das S-Flag wird gesetzt, wenn entweder das V-Flag oder das N-Flag gesetzt ist. Wenn beide Bits gleich sind, wird das S-Flag gelöscht.
V:	Das V-Flag wird gesetzt, wenn aus der Operation ein Zweierkomplement- Überlauf erfolgt, andernfalls wird das Flag gelöscht.
N:	Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag wird gesetzt, wenn der absolute Wert von K größer ist als der absolute Wert von Rd, andernfalls wird das C-Flag gelöscht.

Beispiel:

```
    cpi r3,$03    ;Vergleicht das Register R3 mit 3
    brne ungleich    ;Verzweigung zum Label ungleich, wenn R3 ≠ 3 ist.
```

```
    ...
ungleich: nop        ;Ziel der Verzweigung, Leerbefehl ausführen
```

CPSE - Compare Skip if Equal

Syntax:	CPSE Rd,Rr
Funktion:	Dieser Befehl vergleicht die beiden Register Rd und Rr miteinander und überspringt den nächsten Befehl, wenn beide Register gleich sind.
Operation:	(a) PC ← PC + 1, wenn Rd ≠ Rr (b, c) PC ← PC + 2 (oder 3), wenn Rd = Rr
Operanden:	$0 \leq d \leq 31$, $0 \leq r \leq 31$
Programmzähler:	(a) PC ← PC + 1, wenn Rd ≠ Rr (b) PC ← PC + 2, wenn Rd = Rr und der nächste Befehl 1 Word lang ist (c) PC ← PC + 3, wenn Rd = Rr, und der nächste Befehl 2 Word lang ist
Words:	1 (2 Byte)
Zyklen:	(a) 1 (b) 2 (c) 3
16 Bit Operations Code:	0001 00rd dddd rrrr

Flags im Status-Register (SREG):

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

Beispiel:

```
oben:    inc r4          ;Register R4 um eins erhöhen
    cpse r4,r0    ;Nächsten Befehl überspringen, wenn R4 = R0 ist.
    jmp oben    ;Sprung zum Label oben, da R4 ≠ R0
    nop        ;Weitermachen, wenn R4 = R0
```

DEC - Decrement

Syntax:	DEC Rd
Funktion:	Dieser Befehl subtrahiert eine 1 vom Inhalt des angegebenen Registers und legt das Ergebnis im Register ab. Das Carry-Flag wird dabei nicht verändert, so dass der Befehl in einem Schleifenzähler verwendet werden kann, ohne die eigentlichen Berechnungen zu beeinflussen. Wenn mit vorzeichenlosen Werten gearbeitet wird, arbeiten nur die BREQ und BRNE Verzweigungsbefehle korrekt. Wenn mit Zweierkomplement-Werten gearbeitet wird, können alle vorzeichenbehaftete Verzweigungsbefehle verwendet werden.
Operation:	Rd ← Rd - 1
Operanden:	$0 \leq d \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 010d dddd 1010

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	↔	↔	↔	↔	-

S:	Das S-Flag wird gesetzt, wenn entweder das V-Flag oder das N-Flag gesetzt ist. Wenn beide Bits gleich sind, wird das S-Flag gelöscht.
V:	Das V-Flag wird gesetzt, wenn aus der Operation ein Zweierkomplement- Überlauf erfolgt, andernfalls wird das Flag gelöscht. Ein Zweierkomplement- Überlauf kann nur auftreten, wenn Rd vor der Operation den Wert 80h hatte.
N:	Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es

gelöscht.

Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag wird gesetzt, wenn im Ergebnis ein Überlauf von Bit 8 erfolgte, andernfalls wird es gelöscht.

Beispiel:

```
    ldi r3,$10    ;Das Register R3 mit 10h laden
loop:    add r1,r2    ;R1 und R2 addieren
    dec r3        ;R3 um 1 vermindern
    brne loop    ;Verzweigung zum Label loop, wenn R3 ≠ 0
    nop        ;Weiter, wenn R3 = 0
```

EICALL - Extended Indirect Call to Subroutine

Syntax:	EICALL
Funktion:	Dieser Befehl ruft ein Unterprogramm auf, das über den Z-Pointer und das EIND Register im I/O-Speicher adressiert wird. Mit diesem Befehl kann also der vollständige Programmspeicher adressiert werden. Die Rücksprungadresse wird im STACK zwischengespeichert. Der Befehl kann nur in Bausteinen mit einem 22 Bit breiten PC benutzt werden (bei 16 Bit PC siehe ICALL)
Operation:	PC(15:0) ← Z(15:0) PC(21:16) ← EIND STACK ← PC + 1 SP ← SP -3
Operanden:	keine
Programmzähler:	siehe Operation
Words:	1 (2 Byte)
Zyklen:	4
16 Bit Operations Code:	1001 0101 0001 1001

Flags im Status-Register (SREG):

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

Beispiel:

```
ldi r16,$05    ;Das Register EIND mit
out EIND,r16    ;dem Wert 05h laden.

ldi r30,$00    ;Low-Byte Z-Pointer laden
```

```
ldi r31,$10    ;High-Byte Z-Pointer laden

eicall        ;Unterprogramm an Adresse 051000h aufrufen
```

EIJMP - Extended Indirect Jump

Syntax:	EIJMP
Funktion:	Dieser Befehl springt zu einer Adresse, die über den Z-Pointer und das EIND-Register im I/O-Speicher adressiert wird. Mit diesem Befehl kann also der vollständige Programmspeicher adressiert werden. Der Befehl ist nicht in allen Bausteinen verfügbar.
Operation:	PC(15:0) ← Z(15:0) PC(21:16) ← EIND
Operanden:	keine
Programmzähler:	siehe Operation
Words:	1 (2 Byte)
Zyklen:	2
16 Bit Operations Code:	1001 0100 0001 1001

Flags im Status-Register (SREG):

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

Beispiel:

```
ldi r16,$05    ;Das Register EIND mit
out EIND,r16    ;dem Wert 05h laden.

ldi r30,$00    ;Low-Byte Z-Pointer laden
ldi r31,$10    ;High-Byte Z-Pointer laden

eijmp        ;Sprung zu Adresse 051000h
```

ELPM - Extended Load Program Memory

Syntax:	(a) ELPM (b) ELPM Rd,Z (c) ELPM Rd,Z+
Funktion:	Dieser Befehl lädt ein Byte, das über das Z-Register und das RAMPZ-Register adressiert ist, aus dem Programmspeicher in das Register Rd. Mit dem Befehl kann der gesamte Programmspeicher adressiert werden. Der Programmspeicher enthält an jeder Adresse 2 Byte (1 Word). Daher adressiert das Z-Register, wenn das LSB 0 ist, das niederwertige Byte der Adresse und mit LSB = 1 das höherwertige Byte des Wortes an der Adresse. Das Z-Register kann mit dem Befehl entweder unverändert bleiben oder automatisch inkrementiert werden. Das Inkrementieren wirkt sich auf die

	verketteten RAMPZ- und Z-Register aus. Bei Bausteinen mit der Möglichkeit des Selbstprogrammierens kann der LPM - Befehl verwendet werden, um die Werte der Fuse- und Lock-Bits zu lesen. Der Befehl ist nicht in allen Bausteinen enthalten.
Operation:	(a) R0 ← (RAMPZ:Z) (b) Rd ← (RAMPZ:Z) (c) Rd ← (RAMPZ:Z), RAMPZ:Z ← RAMPZ:Z + 1
Operanden:	(a) keiner, R0 ist das Ziel (b) $0 \leq d \leq 31$ (c) $0 \leq d \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	3
16 Bit Operations Code:	(a) 1001 0101 1101 1000 (b) 1001 000d dddd 0110 (c) 1001 000d dddd 0111

Flags im Status-Register (SREG):

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

Beispiel:

```
ldi ZH,$01    ;ZH-Register mit 01h laden
out RAMPZ,ZH    ;01h in RAMPZ schreiben

ldi r30,$00    ;Low-Byte Z-Pointer mit 00h laden
elpm r20,Z+    ;Low-Byte aus Adresse 010100h in R20 laden,
             ;anschließend Zeiger inkrementieren
...
elpm r20,Z    ;High-Byte aus Adresse 010101h in R20 laden
```

EOR - Exclusive OR

Syntax:	EOR Rd,Rr
Funktion:	Dieser Befehl führt eine Exclusiv-Oder - Verknüpfung zwischen den Registern Rd und Rr durch. Das Ergebnis wird im Register Rd abgelegt.
Operation:	Rd ← Rd ⊕ Rr
Operanden:	$0 \leq d \leq 31$, $0 \leq r \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0010 01rd dddd rrrr

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	↔	0	↔	↔	-

S:	Das S-Flag wird gesetzt, wenn das N-Flag gesetzt ist, andernfalls wird das S-Flag gelöscht.
V:	Das V-Flag wird gelöscht.
N:	Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.

Beispiel:

```
eor r3,r3    ;Das Register R3 wird gelöscht
eor r30,r23    ;Exclusiv-Oder - Verknüpfung zwischen R30 und R23, Ergebnis in
R30
```

FMUL - Fractional Multiply Unsigned

Syntax:	FMUL Rd,Rr
Funktion:	Dieser Befehl multipliziert eine 8 Bit - Zahl mit einer 8 Bit - Zahl und schiebt das 16 Bit - Ergebnis eine Position nach links. Der Multiplikand Rd und der Multiplikator Rr sind zwei Register, die vorzeichenlose gebrochene Zahlen enthalten, bei denen der Dezimalpunkt zwischen dem 6ten und 7ten Bit liegt. Das 16 Bit - Ergebnis ist ebenfalls vorzeichenlos und hat seinen Dezimalpunkt zwischen dem 14ten und 15ten Bit. Das Ergebnis wird im Register R1 (High-Byte) und R0 (Low-Byte) abgelegt. Eine Multiplikation zweier Zahlen im Format N1.Q1 und N2.Q2 ergibt ein Ergebnis im Format (N1+N2).(Q1+Q2). N bedeutet dabei die Anzahl der binären Stellen links vom Dezimalpunkt, Q bedeutet die Anzahl der binären Stellen rechts vom Dezimalpunkt. In Mikrocontroller-Anwendungen liegt der Input oft im Format 1.7, womit sich ein Ergebnis im Format 2.14 ergibt. Ein Linksschieben des Ergebnisses ermöglicht, dass das High- Byte des Ergebnisses im gleichen Format wie der Input vorliegt. Das 1.7 Format wird für gewöhnlich bei vorzeichenbehafteten Zahlen verwendet, während FMUL eine Multiplikation vorzeichenloser Zahlen durchführt. Daher wird FMUL für eine Teilmultiplikation von 16 Bit - Zahlen im 1.15 Format verwendet, wenn ein Ergebnis im 1.31 Format benötigt wird. Das Ergebnis der FMUL-Operation kann an einem Zweierkomplement-Überlauf leiden, wenn das Ergebnis als eine Zahl im 1.15 Format interpretiert wird. Das MSB des Ergebnisses vor dem Linksschieben wird in das Carry-Flag gesichert. Dieser Befehl ist nicht in allen Bausteinen verfügbar. Als Operanden stehen nur die Register R16 bis R23 zur Verfügung.
Operation:	R1:R0 ← Rd x Rr (vorzeichenlos: (1.15) ← (1.7) x (1.7))
Operanden:	$16 \leq d \leq 23$, $16 \leq r \leq 23$
Programmzähler:	PC ← PC + 1

Words:	1 (2 Byte)
Zyklen:	2
16 Bit Operations Code:	0000 0011 0ddd 1rrr

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	↔	↔

Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 0000h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag wird gesetzt, wenn das Bit 15 des Ergebnisses vor dem Linksschieben gesetzt war. War das Bit vorher gelöscht, so wird auch das Carry- Flag gelöscht.

Beispiel:

```
fmul r23,r22    ;Multipliziert vorzeichenlos R23 und R22 im 1.7 Format.
                ;Ergebnis in R1:R0 im 1.15 Format
```

FMULS - Fractional Multiply Signed

Syntax:	FMULS Rd,Rr
Funktion:	Dieser Befehl multipliziert eine 8 Bit - Zahl mit einer 8 Bit - Zahl und schiebt das 16 Bit - Ergebnis eine Position nach links. Der Multiplikand Rd und der Multiplikator Rr sind zwei Register, die vorzeichenbehaftete gebrochene Zahlen enthalten, bei denen der Dezimalpunkt zwischen dem 6ten und 7ten Bit liegt. Das 16 Bit - Ergebnis ist ebenfalls vorzeichenbehaftet und hat seinen Dezimalpunkt zwischen dem 14ten und 15ten Bit. Das Ergebnis wird im Register R1 (High-Byte) und R0 (Low-Byte) abgelegt. Eine Multiplikation zweier Zahlen im Format N1.Q1 und N2.Q2 ergibt ein Ergebnis im Format (N1+N2).(Q1+Q2). N bedeutet dabei die Anzahl der binären Stellen links vom Dezimalpunkt, Q bedeutet die Anzahl der binären Stellen rechts vom Dezimalpunkt. In Mikrocontroller-Anwendungen liegt der Input oft im Format 1.7 vor, womit sich ein Ergebnis im Format 2.14 ergibt. Ein Linksschieben des Ergebnisses ermöglicht, dass das High-Byte des Ergebnisses im gleichen Format wie der Input vorliegt. Die Multiplikation von 80h (-1) mit 80h (-1) ergibt nach dem Linksschieben das Ergebnis 8000h (-1). Das Linksschieben erzeugt in diesem Fall einen Zweierkomplement- Überlauf, der durch die Software ausgewertet werden muss. Dieser Befehl ist nicht in allen Bausteinen verfügbar. Als Operanden stehen nur die Register R16 bis R23 zur Verfügung.
Operation:	R1:R0 ← Rd x Rr (mit Vorzeichen: (1.15) ← (1.7) x (1.7))
Operanden:	$16 \leq d \leq 23$, $16 \leq r \leq 23$
Programmzähler:	PC ← PC + 1

Words:	1 (2 Byte)
Zyklen:	2
16 Bit Operations Code:	0000 0011 1ddd 0rrr

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	↔	↔

Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 0000h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag wird gesetzt, wenn das Bit 15 des Ergebnisses vor dem Linksschieben gesetzt war. War das Bit vorher gelöscht, so wird auch das Carry- Flag gelöscht.

Beispiel:

```
fmuls r23,r22    ;Multipliziert vorzeichenbehaftet R23 und R22 im 1.7 Format.
                 ;Ergebnis in R1:R0 im 1.15 Format
```

FMULSU - Fractional Multiply Signed with Unsignedd

Syntax:	FMULSU Rd,Rr
Funktion:	Dieser Befehl multipliziert eine 8 Bit - Zahl mit einer 8 Bit - Zahl und schiebt das 16 Bit - Ergebnis eine Position nach links. Der Multiplikand Rd ist eine vorzeichenbehaftete gebrochene Zahl und der Multiplikator Rr ist eine vorzeichenlose gebrochene Zahl. Beide enthalten einen unterstellten Dezimalpunkt, der zwischen dem 6ten und 7ten Bit liegt. Das 16 Bit - Ergebnis ist ebenfalls vorzeichenbehaftet und hat seinen Dezimalpunkt zwischen dem 14ten und 15ten Bit. Das Ergebnis wird im Register R1 (High-Byte) und R0 (Low-Byte) abgelegt. Eine Multiplikation zweier Zahlen im Format N1.Q1 und N2.Q2 ergibt ein Ergebnis im Format (N1+N2).(Q1+Q2). N bedeutet dabei die Anzahl der binären Stellen links vom Dezimalpunkt, Q bedeutet die Anzahl der binären Stellen rechts vom Dezimalpunkt. In Mikrocontroller-Anwendungen liegt der Input oft im Format 1.7, womit sich ein Ergebnis im Format 2.14 ergibt. Ein Linksschieben des Ergebnisses ermöglicht, dass das High- Byte des Ergebnisses im gleichen Format wie der Input vorliegt. Das 1.7 Format wird für gewöhnlich bei vorzeichenbehafteten Zahlen verwendet, während FMULSU eine Multiplikation einer vorzeichenbehafteten mit einer vorzeichenlosen Zahl durchführt. Daher wird FMULSU für zwei Teilmultiplikation von 16 Bit - Zahlen im 1.15 Format verwendet, wenn ein Ergebnis im 1.31 Format benötigt wird. Das Ergebnis der FMULSU-Operation kann einen Zweierkomplement-Überlauf haben, wenn das Ergebnis als eine Zahl im 1.15 Format interpretiert wird. Das MSB des Ergebnisses vor dem Linksschieben wird in das Carry-Flag

	gesichert. Dieser Befehl ist nicht in allen Bausteinen verfügbar. Als Operanden stehen nur die Register R16 bis R23 zur Verfügung.
Operation:	R1:R0 ← Rd x Rr (mit Vorzeichen (1.15) ← mit Vorzeichen (1.7) x ohne Vorzeichen (1.7))
Operanden:	16 ≤ d ≤ 23, 16 ≤ r ≤ 23
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	2
16 Bit Operations Code:	0000 0011 1ddd 1rrr

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	↔	↔

Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 0000h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag wird gesetzt, wenn das Bit 15 des Ergebnisses vor dem Linksschieben gesetzt war. War das Bit vorher gelöscht, so wird auch das Carry- Flag gelöscht.

Beispiel:

```
fmulsu r23,r22    ;Multipliziert R23 und R22 im 1.7 Format.
                  ;Ergebnis in R1:R0 im 1.15 Format
```

ICALL - Indirect Call to Subroutine

Syntax:	ICALL
Funktion:	Dieser Befehl ruft ein Unterprogramm auf, das über den Z-Pointer adressiert wird. Da das Z-Register ein 16 Bit - Register ist, können auch in Bausteinen mit 22 Bit breiter Adresse nur die unteren 64k adressiert werden. Die Rücksprungadresse wird im STACK zwischengespeichert. Dieser Befehl ist nicht in allen Bausteinen verfügbar.
Operation:	(a) bei 16 Bit Adressen PC(15:0) ← Z(15:0) STACK ← PC + 1 SP ← SP - 2 (b) bei 22 Bit Adressen PC(15:0) ← Z(15:0), PC(21:16) ← 0 STACK ← PC + 1 SP ← SP - 3
Operanden:	keine
Programmzähler:	siehe Operation

Words:	1 (2 Byte)
Zyklen:	(a) 3 (b) 4
16 Bit Operations Code:	1001 0101 0000 1001

Flags im Status-Register (SREG):

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

Beispiel:

```
ldi r30,$00    ;Low-Byte Z-Pointer laden
ldi r31,$10    ;High-Byte Z-Pointer laden
icall        ;Unterprogramm an Adresse 1000h aufrufen
```

IJMP - Indirect Jump

Syntax:	IJMP
Funktion:	Dieser Befehl springt zu einer Adresse, die über den Z-Pointer adressiert wird. Da das Z- Register ein 16 Bit - Register ist, können auch in Bausteinen mit 22 Bit breiter Adresse nur die unteren 64k adressiert werden. Dieser Befehl ist nicht in allen Bausteinen verfügbar.
Operation:	(a) PC(15:0) ← Z(15:0), bei 16 Bit Adressen (b) PC(15:0) ← Z(15:0), PC(21:16) ← 0 bei 22 Bit Adressen
Operanden:	keine
Programmzähler:	siehe Operation
Words:	1 (2 Byte)
Zyklen:	2
16 Bit Operations Code:	1001 0100 0000 1001

Flags im Status-Register (SREG):

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

Beispiel:

```
ldi r30,$00    ;Low-Byte Z-Pointer laden
ldi r31,$10    ;High-Byte Z-Pointer laden
ijmp         ;Sprung an Adresse 1000h
```

IN - Load an I/O-Location to Register

Syntax:	IN Rd,A
Funktion:	Dieser Befehl lädt Daten aus dem I/O-Speicher (Ports, Timer, etc) in das Register Rd.
Operation:	Rd ← Rd,I/O(A)
Operanden:	0 ≤ d ≤ 31, 0 ≤ A ≤ 63
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1011 0AAd dddd AAAA

Flags im Status-Register (SREG):

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

Beispiel:

```
in r30,$16    ;Liest die Eingangspins von Port B in das Register R30
```

INC - Increment

Syntax:	INC Rd
Funktion:	Dieser Befehl addiert eine 1 zum Inhalt des angegebenen Registers und legt das Ergebnis im Register ab. Das Carry-Flag wird dabei nicht verändert, so dass der Befehl in einem Schleifenzähler verwendet werden kann, ohne die eigentlichen Berechnungen zu beeinflussen. Wenn mit vorzeichenlosen Werten gearbeitet wird, arbeiten nur die BREQ- und BRNE- Verzweigungsbefehle korrekt. Wenn mit Zweierkomplement-Werten gearbeitet wird, können alle vorzeichenbehaftete Verzweigungsbefehle verwendet werden.
Operation:	Rd ← Rd + 1
Operanden:	0 ≤ d ≤ 31
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 010d dddd 0011

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C

-	-	-	↔	↔	↔	↔	-

S:	Das S-Flag wird gesetzt, wenn entweder das V-Flag oder das N-Flag gesetzt ist. Wenn beide Bits gleich sind, wird das S-Flag gelöscht.
V:	Das V-Flag wird gesetzt, wenn aus der Operation ein Zweierkomplement- Überlauf erfolgt, andernfalls wird das Flag gelöscht. Ein Zweierkomplement- Überlauf kann nur auftreten, wenn Rd vor der Operation den Wert 80h hatte.
N:	Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.

Beispiel:

```
    ldi r3,$00    ;Das Register R3 mit 00h laden
loop:
    inc r3        ;R3 um 1 erhöhen
    cpi r3,$7A    ;Vergleichen, ob R3 = 7Ah
    brne loop    ;Verzweigung zum Label loop, solange R3 ≠ 7Ah
    nop         ;Weiter, wenn R3 = 7Ah
```

JMP - Jump

Syntax:	JMP k
Funktion:	Dieser Befehl springt zu der Adresse, die im Befehl mit der Konstanten k angegeben ist. Mit dem Befehl kann der ganze Bereich des Programmspeichers adressiert werden. Dieser Befehl ist nicht in allen Bausteinen verfügbar.
Operation:	PC ← k
Operanden:	0 ≤ k ≤ 4M
Programmzähler:	PC ← k
Words:	2 (4 Byte)
Zyklen:	3
32 Bit Operations Code:	1001 010k kkkk 110k kkkk kkkk kkkk kkkk

Flags im Status-Register (SREG):

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

Beispiel:

```
    ldi r30,$00    ;Low-Byte Z-Pointer laden
    jmp ziel    ;Sprung zur Adresse mit dem Label Ziel
    ...

ziel:    nop        ;Zieladresse, Leerbefehl
```

LD - Load Indirect from Data Space to Register using Index X

Syntax:	(a) LD Rd,X (b) LD Rd,X+ (c) LD Rd,-X
Funktion:	Dieser Befehl lädt ein Byte, das über das X-Register adressiert ist, aus dem Datenspeicher. Bei Bausteinen mit SRAM besteht der Datenspeicher aus den Registern, den I/O-Speichern und dem internen SRAM (sowie dem externen SRAM, wenn möglich). Bei Bausteinen ohne SRAM besteht der Datenspeicher lediglich aus den Registern. Das EEPROM hat immer einen eigenen Adressbereich. Die 16 Bit - Adresse der Speicherzelle wird im X-Pointer angegeben, womit maximal 64k Speicher adressiert werden können. Der LD - Befehl benutzt das RAMPX-Register, um Speicherbereiche über 64k anzusprechen. Um auf ein anderes 64k-Segment zuzugreifen, muss also der Inhalt des RAMPX-Registers verändert werden. Nicht alle Varianten des Befehls sind in allen Bausteinen verfügbar. Das X-Register kann mit dem Befehl entweder unverändert bleiben oder automatisch (nach)inkrementiert oder (vor)dekrementiert werden. Das Inkrementieren und Dekrementieren wirkt sich auf die verketteten RAMPX- und X-Register aus. Bei Bausteinen mit nicht mehr als 256 Byte Datenspeicher bleibt das High-Byte des X- Pointers unberücksichtigt und kann auch für andere Zwecke verwendet werden.
Operation:	(a) Rd ← (X) (b) Rd ← (X), X ← X + 1 (c) X ← X - 1, Rd ← (X)
Operanden:	$0 \leq d \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	3
16 Bit Operations Code:	(a) 1001 000d dddd 1100 (b) 1001 000d dddd 1101 (c) 1001 000d dddd 1110

Flags im Status-Register (SREG):

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

Beispiel:

```
    clr r27         ;High-Byte X-Pointer löschen
    ldi r26,$60     ;Low-Byte X-Pointer aus 60h setzen
    ld r0,X+    ;R0 mit Wert aus Adresse 60h laden
    ld r1,X         ;R1 mit Wert aus Adresse 61h laden
weiter
    ld r3,X         ;R3 mit Wert aus Adresse 61h laden
    ld r2,-X    ;R2 mit Wert aus Adresse 60h laden
```

LD (LDD) - Load Indirect from Data Space to Register using Index Y

Syntax:	(a) LD Rd,Y (b) LD Rd,Y+ (c) LD Rd,-Y (d) LDD Rd,Y+q
Funktion:	Dieser Befehl lädt ein Byte, das mit oder ohne Verschiebung (q) über das Y-Register adressiert ist, aus dem Datenspeicher. Bei Bausteinen mit SRAM besteht der Datenspeicher aus den Registern, den I/O-Speichern und dem internen SRAM (sowie dem externen SRAM, wenn möglich). Bei Bausteinen ohne SRAM besteht der Datenspeicher lediglich aus den Registern. Das EEPROM hat immer einen eigenen Adressbereich. Die 16 Bit - Adresse der Speicherzelle wird im Y-Pointer angegeben, womit maximal 64k Speicher adressiert werden können. Der LD - Befehl benutzt das RAMPY-Register, um Speicherbereiche über 64k anzusprechen. Um auf ein anderes 64k-Segment zuzugreifen, muss also der Inhalt des RAMPY-Registers verändert werden. Nicht alle Varianten des Befehls sind in allen Bausteinen verfügbar. Das Y-Register kann mit dem Befehl entweder unverändert bleiben oder automatisch (nach)inkrementiert oder (vor)dekrementiert werden. Das Inkrementieren, Dekrementieren und die Verschiebung wirken sich auf die verketteten RAMPY- und Y- Register aus. Bei Bausteinen mit nicht mehr als 256 Byte Datenspeicher bleibt das High- Byte des Y-Pointers unberücksichtigt und kann auch für andere Zwecke verwendet werden.
Operation:	(a) Rd ← (Y) (b) Rd ← (Y), Y ← Y + 1 (c) Y ← Y - 1, Rd ← (Y) (d) Rd ← (Y+q)
Operanden:	0 ≤ d ≤ 31, (d) 0 ≤ q ≤ 63
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	3
16 Bit Operations Code:	(a) 1000 000d dddd 1000 (b) 1001 000d dddd 1001 (c) 1001 000d dddd 1010 (d) 10q0 qq0d dddd 1qqq

Flags im Status-Register (SREG):

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

Beispiel:

```
clr r29         ;High-Byte Y-Pointer löschen
ldi r28,$60     ;Low-Byte Y-Pointer aus 60h setzen
ld r0,Y+    ;R0 mit Wert aus Adresse 60h laden
ld r1,Y         ;R1 mit Wert aus Adresse 61h laden
```

```
ld r2,-Y    ;R2 mit Wert aus Adresse 60h laden
ld r3,Y+2    ;R3 mit Wert aus Adresse 62h laden
```

LD (LDD) - Load Indirect from Data Space to Register using Index Z

Syntax:	(a) LD Rd,Z (b) LD Rd,Z+ (c) LD Rd,-Z (d) LDD Rd,Z+q
Funktion:	Dieser Befehl lädt ein Byte, das mit oder ohne Verschiebung (q) über das Z-Register adressiert ist, aus dem Datenspeicher. Bei Bausteinen mit SRAM besteht der Datenspeicher aus den Registern, den I/O-Speichern und dem internen SRAM (sowie dem externen SRAM, wenn möglich). Bei Bausteinen ohne SRAM besteht der Datenspeicher lediglich aus den Registern. Das EEPROM hat immer einen eigenen Adressbereich. Nicht alle Varianten des Befehls sind in allen Bausteinen verfügbar. Die 16 Bit - Adresse der Speicherzelle wird im Z-Pointer angegeben, womit maximal 64k Speicher adressiert werden können. Der LD - Befehl benutzt das RAMPZ-Register, um Speicherbereiche über 64k anzusprechen. Um auf ein anderes 64k-Segment zuzugreifen, muss also der Inhalt des RAMPZ-Registers verändert werden. Da der Z- Pointer auch für indirekte Unterprogrammaufrufe, indirekte Sprünge und das Auslesen von Tabellen aus dem Programmspeicher verwendet wird, sollten besser die X- und Y-Pointer für das Adressieren von Speicherplätzen verwendet werden. Das Z-Register kann mit dem Befehl entweder unverändert bleiben oder automatisch (nach)inkrementiert oder (vor)dekrementiert werden. Das Inkrementieren, Dekrementieren und die Verschiebung wirken sich auf die verketteten RAMPZ- und Z- Register aus. Bei Bausteinen mit nicht mehr als 256 Byte Datenspeicher bleibt das High- Byte des Z-Pointers unberücksichtigt und kann auch für andere Zwecke verwendet werden.
Operation:	(a) Rd ← (Z) (b) Rd ← (Z), Z ← Z + 1 (c) Z ← Z - 1, Rd ← (Z) (d) Rd ← (Z+q)
Operanden:	$0 \leq d \leq 31$, (d) $0 \leq q \leq 63$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	3
16 Bit Operations Code:	(a) 1000 000d dddd 0000 (b) 1001 000d dddd 0001 (c) 1001 000d dddd 0010 (d) 10q0 qq0d dddd 0qqq

Flags im Status-Register (SREG):

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

Beispiel:

```
clr r31         ;High-Byte Z-Pointer löschen
ldi r30,$60     ;Low-Byte Z-Pointer aus 60h setzen
ld r0,Z         ;R0 mit Wert aus Adresse 60h laden
ld r3,Z+2     ;R3 mit Wert aus Adresse 62h laden
```

LDI - Load Immediate

Syntax:	LDI Rd,K
Funktion:	Dieser Befehl lädt eine Konstante in das Register Rd. Der Befehl kann nur mit den Registern R16 bis R31 verwendet werden.
Operation:	Rd ← K
Operanden:	$16 \leq d \leq 31$, $0 \leq K \leq 255$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1110 KKKK dddd KKKK

Flags im Status-Register (SREG):

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

Beispiel:

```
ldi r30,$F0     ;Low-Byte Z-Pointer mit F0h laden
```

LDS - Load Direct from Data Space

Syntax:	LDS Rd,k
Funktion:	Dieser Befehl lädt ein Datum aus dem Datenspeicher in das Register Rd. Bei Bausteinen mit internem SRAM besteht der Datenspeicher aus den Registern, dem I/O-Speicher und dem internen SRAM (sowie dem externen SRAM, wenn beschaltbar). Bei Bausteinen ohne internen SRAM besteht der Datenspeicher nur aus den Registern. Das EEPROM hat immer einen eigenen Adressbereich. Die 16 Bit - Adresse muss immer angegeben werden, wobei der Speicherzugriff auf die tatsächlich vorhandenen Speicherzellen begrenzt ist. Der LDS - Befehl benutzt das RAMPD-Register, um Speicherbereiche über

	64k anzusprechen. Um auf ein anderes 64k-Segment zuzugreifen, muss also der Inhalt des RAMPD-Registers verändert werden. Der Befehl ist nicht in allen AVR - Mikrocontrollern verfügbar.
Operation:	Rd ← (k)
Operanden:	$0 \leq d \leq 31$, $0 \leq k \leq 65535$
Programmzähler:	PC ← PC + 2
Words:	2 (4 Byte)
Zyklen:	2
32 Bit Operations Code:	1001 000d dddd 0000 kkkk kkkk kkkk kkkk

Flags im Status-Register (SREG):

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

Beispiel:

```
lds r30,$FF00    ;Datum aus Adresse FF00h in R30 laden.
```

LPM - Load Program Memory

Syntax:	(a) LPM (b) LPM Rd,Z (c) LPM Rd,Z+
Funktion:	Dieser Befehl lädt ein Byte, das über das Z-Register adressiert ist, aus den unteren 64k- Byte des Programmspeichers in das Register Rd. Der Programmspeicher enthält an jeder Adresse 2 Byte (1 Word). Daher adressiert das Z-Register, wenn das LSB 0 ist, das niederwertige Byte der Adresse und mit LSB = 1 das höherwertige Byte des Wortes an der Adresse. Das Z-Register kann mit dem Befehl entweder unverändert bleiben oder automatisch inkrementiert werden. Bei Bausteinen mit der Möglichkeit des Selbstprogrammierens kann der LPM - Befehl verwendet werden, um die Werte der Fuse- und Lock-Bits zu lesen. Nicht alle LPM- Varianten sind in allen Bausteinen verfügbar. Im AT90S1200 ist der Befehl gar nicht implementiert.
Operation:	(a) R0 ← (Z) (b) Rd ← (Z) (c) Rd ← (Z), Z ← Z + 1
Operanden:	(a) keiner, R0 ist das Ziel (b) $0 \leq d \leq 31$ (c) $0 \leq d \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	3

16 Bit Operations Code:	(a) 1001 0101 1100 1000 (b) 1001 000d dddd 0100 (c) 1001 000d dddd 0101

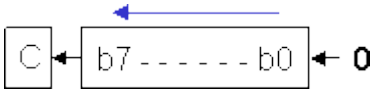
Flags im Status-Register (SREG):

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

Beispiel:

```
ldi r30,$00    ;Low-Byte Z-Pointer mit 00h laden
ldi r31,$01    ;High-Byte Z-Pointer mit 01h laden
lpm r20,Z+    ;Low-Byte aus Adresse 0100h in R20 laden
...
lpm r20,Z    ;High-Byte aus Adresse 0100h in R20 laden
```

LSL - Logical Shift Left

Syntax:	LSL Rd
Funktion:	Mit diesem Befehl werden alle Bits des Registers Rd eine Position nach links geschoben. Das Bit 7 wird in das Carry-Flag geschoben, das Bit 0 wird gelöscht. Damit stellt dieser Befehl unabhängig vom Vorzeichen des Wertes in Rd eine Multiplikation mit 2 dar.
Operation:	
Operanden:	$0 \leq d \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0000 11dd dddd dddd

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	↔	↔	↔	↔	↔	↔

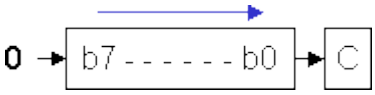
H:	Das H-Flag enthält nach dem Linksschieben das ursprüngliche Bit 3 des Registers Rd.
S:	Das S-Flag wird gesetzt, wenn entweder das N-Flag oder das V-Flag gesetzt ist. Wenn beide Flags gleich sind, wird das S-Flag gelöscht.
V:	Das V-Flag wird gesetzt, wenn nach dem Linksschieben entweder das N-Flag oder das C-Flag gesetzt ist. Wenn beide Bits gleich sind, wird das V-Flag gelöscht.

N:	Das N-Flag wird gesetzt, wenn das MSB (Bit 7) des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 0000h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag enthält nach dem Linksschieben das ursprüngliche Bit 7 des Rd Registers.

Beispiel:

```
lsl r16    ;Multipliziert das Register R16 mit 2 durch Linksschieben.
```

LSR - Logical Shift Right

Syntax:	LSR Rd
Funktion:	Mit diesem Befehl werden alle Bits des Registers Rd eine Position nach rechts geschoben. Das Bit 0 wird in das Carry-Flag geschoben, das Bit 7 wird gelöscht. Damit stellt dieser Befehl bei einem vorzeichenlosen Wert in Rd eine Division mit 2 dar. Das Carry kann genutzt werden, um das Ergebnis zu runden.
Operation:	
Operanden:	$0 \leq d \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 010d dddd 0110

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	↔	↔	0	↔	↔

S:	Das S-Flag wird gesetzt, wenn entweder das N-Flag oder das V-Flag gesetzt ist. Wenn beide Flags gleich sind, wird das S-Flag gelöscht.
V:	Das V-Flag wird gesetzt, wenn nach dem Rechtsschieben entweder das N-Flag oder das C-Flag gesetzt ist. Wenn beide Bits gleich sind, wird das V-Flag gelöscht.
N:	Das N-Flag wird gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 0000h ist, andernfalls wird das Flag gelöscht.

C: Das C-Flag enthält nach dem Rechtsschieben das ursprüngliche Bit 7 des Rd Registers.

Beispiel:

```
lsr r16    ;Schiebt die Bits des Registers R16 eine Position nach rechts.
```

MOV - Copy Register

Syntax:	MOV Rd,Rr
Funktion:	Dieser Befehl kopiert den Wert des Registers Rr in das Register Rd. Das Register Rr wird dabei nicht verändert.
Operation:	Rd ← Rr
Operanden:	$0 \leq d \leq 31$, $0 \leq r \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0010 11rd dddd rrrr

Flags im Status-Register (SREG):

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

Beispiel:

```
mov r3,r2    ;Kopiert R2 in R3
```

MOVW - Copy Register Word

Syntax:	MOVW Rd+1:Rd,Rr+1:Rr
Funktion:	Dieser Befehl kopiert den Wert eines Registerpaares in ein anderes Registerpaar. Das Registerpaar Rr+1:Rr wird dabei nicht verändert. Dieser Befehl ist nicht in allen AVR - Mikrocontrollern verfügbar.
Operation:	Rd+1:Rd ← Rr+1:Rr
Operanden:	$d \varepsilon \{0,2,4,...,30\}$, $r \varepsilon \{0,2,4,...,30\}$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0000 0001 dddd rrrr

Flags im Status-Register (SREG):

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

Beispiel:

```
movw r3:r2,r1:r0    ;Kopiert das Paar R1:R0 in das Paar R3:R2
```

MUL - Multiply Unsigned

Syntax:	MUL Rd,Rr
	Dieser Befehl multipliziert eine 8 Bit - Zahl mit einer 8 Bit - Zahl, das Ergebnis ist 16 Bit lang.
Funktion:	Der Multiplikand Rd und der Multiplikator Rr sind vorzeichenlose Zahlen. Das 16 Bit - Ergebnis ist ebenfalls vorzeichenlos. Das Ergebnis wird im Register R1 (High-Byte) und R0 (Low-Byte) abgelegt. Wenn als Rd oder Rr die Register R0 oder R1 ausgewählt werden, so werden diese Register mit dem Ergebnis überschrieben. Dieser Befehl ist nicht in allen Bausteinen verfügbar.
Operation:	R1:R0 ← Rd x Rr
Operanden:	$0 \leq d \leq 31$, $0 \leq r \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	2
16 Bit Operations Code:	1001 11rd dddd rrrr

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	↔	↔

Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 0000h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag wird gesetzt, wenn das Bit 15 des Ergebnisses gesetzt ist, andernfalls wird das Carry-Flag gelöscht.

Beispiel:

```
mul r5,r4         ;R5 mit R4 multiplizieren
movw r5:r4,r1:r0    ;Ergebnis in R5:R4 zurück schreiben
```

MULS - Multiply Signed

Syntax:	MULS Rd,Rr
	Dieser Befehl multipliziert eine 8 Bit - Zahl mit einer 8 Bit - Zahl, das Ergebnis ist 16 Bit lang.
Funktion:	Der Multiplikand Rd und der Multiplikator Rr sind vorzeichenbehaftete Zahlen. Das 16 Bit - Ergebnis ist ebenfalls vorzeichenbehaftet. Das Ergebnis wird im Register R1 (high Byte) und R0 (low Byte) abgelegt. Dieser Befehl ist nicht in allen Bausteinen verfügbar. Als Operanden können nur die Register R16 bis R31 ausgewählt werden.
Operation:	R1:R0 ← Rd x Rr
Operanden:	16 ≤ d ≤ 31, 16 ≤ r ≤ 31
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	2
16 Bit Operations Code:	0000 0010 dddd rrrr

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	↔	↔

Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 0000h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag wird gesetzt, wenn das Bit 15 des Ergebnisses gesetzt ist, andernfalls wird das Carry-Flag gelöscht.

Beispiel:

```
mul r21,r20         ;R21 mit R20 multiplizieren
movw r21:r20,r1:r0    ;Ergebnis in R21:R20 zurück schreiben
```

MULSU - Multiply Signed with unsigned

Syntax:	MULSU Rd,Rr
	Dieser Befehl multipliziert eine 8 Bit - Zahl mit einer 8 Bit - Zahl, das Ergebnis ist 16 Bit lang.
Funktion:	Der Multiplikand Rd ist eine vorzeichenbehaftete Zahl, der Multiplikator Rr ist eine vorzeichenlose Zahl. Das 16 Bit - Ergebnis ist ebenfalls vorzeichenbehaftet. Das Ergebnis wird im Register R1 (High-Byte) und R0 (Low-Byte) abgelegt. Dieser Befehl ist nicht in allen Bausteinen verfügbar. Als Operanden können nur die Register R16 bis R23 ausgewählt werden.
Operation:	R1:R0 ← Rd x Rr (mit Vorzeichen ? mit Vorzeichen x ohne Vorzeichen)

Operanden:	$16 \leq d \leq 23$, $16 \leq r \leq 23$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	2
16 Bit Operations Code:	0000 0011 0ddd 0rrr

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	↔	↔

Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 0000h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag wird gesetzt, wenn das Bit 15 des Ergebnisses gesetzt ist, andernfalls wird das Carry-Flag gelöscht.

Beispiel:

```
mulsu r23,r22    ;Multipliziert R23 und R22. Ergebnis in R1:R0.
```

NEG - Two's Complement

Syntax:	NEG Rd
Funktion:	Dieser Befehl ersetzt den Wert eines Registers durch sein Zweierkomplement. Enthält das Register den Wert 80h, so bleibt dieser unverändert.
Operation:	Rd ← 00h - Rd
Operanden:	$0 \leq d \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 010d dddd 0001

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	↔	↔	↔	↔	↔	↔

H:	Das H-Flag wird gesetzt, wenn bei der Subtraktion ein Übertrag von Bit 3 nach Bit 4 erfolgte, andernfalls wird das S-Flag gelöscht.
S:	Das S-Flag wird gesetzt, wenn entweder das V-Flag oder das N-Flag gesetzt ist. Wenn beide Bits gleich sind, wird das S-Flag gelöscht.

V:	Das V-Flag wird gesetzt, wenn aus der Operation ein Zweierkomplement- Überlauf erfolgt, andernfalls wird das Flag gelöscht. Ein Zweierkomplement- Überlauf kann nur auftreten, wenn Rd vor der Operation den Wert 80h hatte.
N:	Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag wird gesetzt, wenn bei der Subtraktion ein Übertrag von Null erfolgte, andernfalls wird das Flag gelöscht. Das C-Flag wird in allen Fällen gesetzt, außer wenn das Ergebnis nach der Operation 00h ist.

Beispiel:

```
    sub r3,r0    ;R0 von R3 subtrahieren
    brpl plus    ;Springen, wenn das Ergebnis positiv ist.
    neg r3        ;Zweierkomplement von R3 bilden
plus:
    nop        ;Weiter, Leerbefehl
```

NOP - No Operation

Syntax:	NOP
Funktion:	Dieser Befehl führt keine Operation aus, es handelt sich um einen Leerbefehl, der einen Taktzyklus lang dauert.
Operation:	keine
Operanden:	keine
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0000 0000 0000 0000

Flags im Status-Register (SREG):

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

Beispiel:

```
clr r3        ;Alle Bits in R3 löschen
ser r4        ;Alle Bits in R4 setzen
out $18,r3    ;R3 an Port B ausgeben (alle Bits auf 0)
nop        ;Warten, Leerbefehl
out $18,r4    ;R4 an Port B ausgeben (alle Bits auf 1)
```

OR - Logical OR

Syntax:	OR Rd, Rr
Funktion:	Führt eine logische ODER-Verknüpfung der Inhalte zweier Register durch, das Ergebnis wird im Quellregister Rd abgelegt.
Operation:	Rd ← Rd v Rr
Operanden:	0 ≤ d ≤ 31, 0 ≤ r ≤ 31
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0010 10rd dddd rrrr

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	↔	0	↔	↔	-

S:	Das S-Flag wird gesetzt, wenn das N-Flag gesetzt ist, andernfalls wird das S-Flag gelöscht.
V:	Das V-Flag wird gelöscht.
N:	Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.

Beispiel:

```
or r1, r0    ;ODER-Verknüpfung der Register R0 und R1
oder:
ldi r16,1    ;Maskierungsbyte 0000 0001 in R16 schreiben
or r2,r16    ;Setzen des unteren Bits in R2
```

ORI - Logical OR with Immediate

Syntax:	ORI Rd,K
Funktion:	Führt eine logische ODER-Verknüpfung zwischen einem Register und einer Konstanten durch, das Ergebnis wird im Quellregister Rd abgelegt. Der Befehl kann nur mit den oberen 16 Registern durchgeführt werden.
Operation:	Rd ← Rd v K
Operanden:	16 ≤ d ≤ 31, 0 ≤ K ≤ 255
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)

Zyklen:	1
16 Bit Operations Code:	0110 KKKK dddd KKKK

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	↔	0	↔	↔	-

S:	Das S-Flag wird gesetzt, wenn das N-Flag gesetzt ist, andernfalls wird das S-Flag gelöscht.
V:	Das V-Flag wird gelöscht.
N:	Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.

Beispiel:

```
ori r1,$F0    ;Setzen der obern 4 Bits im Register R1
```

OUT - Store Register to I/O-Location

Syntax:	OUT A,Rr
Funktion:	Dieser Befehl kopiert den Wert des Registers Rr in ein Register (Ports, Timer, etc) im I/O-Speicher. Das Register Rr wird dabei nicht verändert.
Operation:	I/O(A) ← Rr
Operanden:	$0 \leq r \leq 31$, $0 \leq A \leq 63$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1011 1AAr rrrr AAAA

Flags im Status-Register (SREG):

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

Beispiel:

```
clr r3        ;Alle Bits in R3 löschen
ser r4        ;Alle Bits in R4 setzen
out $18,r3    ;R3 an Port B ausgeben (alle Bits auf 0)
nop         ;Warten, Leerbefehl
out $18,r4    ;R4 an Port B ausgeben (alle Bits auf 1)
```

POP - Pop Register from Stack

Syntax:	POP Rd
Funktion:	Dieser Befehl lädt das Register Rd mit einem Byte aus dem Stack. Der Stack-Pointer wird vor der Ausführung des Befehls um 1 erhöht, weil er immer auf die nächste zu beschreibende Speicherzelle zeigt und nicht auf die zuletzt beschriebene Zelle. Der Befehl ist nicht in allen Bausteinen verfügbar.
Operation:	Rd ← STACK, SP ← SP + 1
Operanden:	0 ≤ d ≤ 31
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	2
16 Bit Operations Code:	1001 000d dddd 1111

Flags im Status-Register (SREG):

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

Beispiel:

```
call sprung    ;Unterprogramm ab Label sprung aufrufen

sprung:
    push r3    ;R3 in Stack sichern
    push r4    ;R4 in Stack sichern
    ...    ;das eigentliche Unterprogramm
    pop r4    ;Alten Wert von R4 aus dem Stack zurückholen
    pop r3    ;Alten Wert von R3 aus dem Stack zurückholen
    ret    ;Rücksprung
```

PUSH - PUSH Register on Stack

Syntax:	PUSH Rr
Funktion:	Dieser Befehl speichert das Register Rd in den Stack. Der Stack-Pointer wird nach der Ausführung des Befehls um 1 verringert und zeigt dann auf die nächste zu beschreibende Speicherzelle. Der Befehl ist nicht in allen Bausteinen verfügbar.
Operation:	STACK ← Rr, SP ← SP - 1
Operanden:	0 ≤ d ≤ 31, 0 ≤ r ≤ 31
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)

Zyklen:	2
16 Bit Operations Code:	1001 001d dddd 1111

Flags im Status-Register (SREG):

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

Beispiel:

```
call sprung    ;Unterprogramm ab Label sprung aufrufen

sprung:
    push r3    ;R3 in Stack sichern
    push r4    ;R4 in Stack sichern
    ...    ;das eigentliche Unterprogramm
    pop r4    ;Alten Wert von R4 aus dem Stack zurückholen
    pop r3    ;Alten Wert von R3 aus dem Stack zurückholen
    ret    ;Rücksprung
```

RCALL - Relative Call to Subroutine

Syntax:	RCALL k
Funktion:	Dieser Befehl springt zu einem Unterprogramm, das innerhalb des Bereiches von PC- 2k+1 bis PC+2k (Word) liegen kann. Die Rücksprungadresse, also die Adresse des Befehls, der auf den RCALL-Befehl folgt, wird in den Stack gesichert. Bei der Assemblerprogrammierung werden anstatt der relativen Adresse Labels verwendet. Bei Bausteinen, die über maximal 8k Bytes (4k Word) Programmspeicher verfügen, kann der ganze Adressbereich von jeder Stelle aus angesprochen werden.
Operation:	(a) PC ← PC + k + 1, STACK ← PC + 1, SP ← SP - 2 bei 16-Bit Adressen (b) PC ← PC + k + 1, STACK ← PC + 1, SP ← SP - 3 bei 22-Bit Adressen
Operanden:	$-2K \leq k < 2K$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	3 (bei 16-Bit Adressen) 4 (bei 22-Bit Adressen)
16 Bit Operations Code:	1101 kkkk kkkk kkkk

Flags im Status-Register (SREG):

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

Beispiel:

```
rcall sprung    ;Unterprogramm ab Label sprung aufrufen

sprung:
    push r3    ;R3 in Stack sichern
    push r4    ;R4 in Stack sichern
    ...    ;das eigentliche Unterprogramm
    pop r4    ;Alten Wert von R4 aus dem Stack zurückholen
    pop r3    ;Alten Wert von R3 aus dem Stack zurückholen
    ret    ;Rücksprung
```

RET - Return from Subroutine

Syntax:	RET
Funktion:	Dieser Befehl beendet ein Unterprogramm. Die Rücksprungadresse wird aus dem Stack geladen.
Operation:	(a) PC(15:0) ← STACK bei 16-Bit Adressen SP ← SP + 2 (b) PC(21:0) ← STACK bei 22-Bit Adressen Sp ← SP + 3
Operanden:	keine
Programmzähler:	Siehe Operation
Words:	1 (2 Byte)
Zyklen:	4 (bei 16-Bit Adressen) 5 (bei 22-Bit Adressen)
16 Bit Operations Code:	1001 0101 0000 1000

Flags im Status-Register (SREG):

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

Beispiel:

```
call sprung    ;Unterprogramm ab Label sprung aufrufen

sprung:
    push r3    ;R3 in Stack sichern
    ...    ;das eigentliche Unterprogramm
    pop r3    ;Alten Wert von R3 aus dem Stack zurückholen
    ret    ;Beenden Unterprogramm und Rücksprung
```

RETI - Return from Interrupt

Syntax:	RETI
Funktion:	Dieser Befehl beendet eine Interruptroutine. Die Rücksprungadresse wird aus dem Stack geladen und das Global Interrupt-Flag wird gesetzt. Zu beachten ist, dass das Status- Register beim Aufruf der Interruptroutine

	nicht automatisch gesichert wird.
Operation:	(a) PC(15:0) ← STACK bei 16-Bit Adressen SP ← SP + 2 (b) PC(21:0) ← STACK bei 22-Bit Adressen Sp ← SP + 3
Operanden:	keine
Programmzähler:	Siehe Operation
Words:	1 (2 Byte)
Zyklen:	4 (bei 16-Bit Adressen) 5 (bei 22-Bit Adressen)
16 Bit Operations Code:	1001 0101 0001 1000

Flags im Status-Register (SREG):

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

I: Das I-Flag wird gesetzt, damit werden alle Interrupts global freigegeben.

Beispiel:

```
call sprung    ;Unterprogramm ab Label sprung aufrufen

interrupt1:
    push r3    ;R3 in Stack sichern
    ...    ;die eigentliche Interrupt-Routine
    pop r3    ;Alten Wert von R3 aus dem Stack zurückholen
    reti    ;Beenden die Interrupt-Routine und Rücksprung zum unterbrochenen
Hauptprogramm
```

RJMP - Relative Jump

Syntax:	RJMP k
Funktion:	Dieser Befehl springt zu einer Adresse, die innerhalb des Bereiches von PC-2k+1 bis PC+2k (Word) liegen kann. Bei der Assemblerprogrammierung werden anstatt der relativen Adresse Labels verwendet. Bei Bausteinen, die über maximal 8k (4k Word) Bytes Programmspeicher verfügen, kann der ganze Adressbereich von jeder Stelle aus angesprochen werden.
Operation:	PC ← PC + k + 1
Operanden:	$-2K \leq k < 2K$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	2
16 Bit Operations	1100 kkkk kkkk kkkk

Code:

Flags im Status-Register (SREG):

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

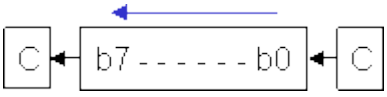
Beispiel:

```
cpi r16,$42     ;Register R16 mit 42h vergleichen
brne fehler     ;Sprung, wenn R16 ungleich 42h
rjmp ok         ;Sprung, wenn R16 gleich 42h

fehler:
    inc r16     ;R16 um eins erhöhen
    ...

ok:
    nop     ;Weiter, Leerbefehl
```

ROL - Rotate Left through Carry

Syntax:	ROL Rd
Funktion:	Mit diesem Befehl werden alle Bits des Registers Rd eine Position nach links geschoben. Das Carry-Bit wird in das Bit 0 geschoben, das Bit 7 wird in das Carry-Flag geschoben. In Kombination mit dem LSL - Befehl können so Mehrbyte-Zahlen mit 2 multipliziert werden. Dieser Befehl arbeitet unabhängig vom Vorzeichen des Wertes in Rd.
Operation:	
Operanden:	$16 \leq d \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0001 11dd dddd dddd

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	↔	↔	↔	↔	↔	↔

H: Das H-Flag enthält nach dem Linksschieben das ursprüngliche Bit 3 des Registers Rd.

S: Das S-Flag wird gesetzt, wenn entweder das N-Flag oder das V-Flag gesetzt ist. Wenn beide Flags gleich sind, wird das S-Flag gelöscht.

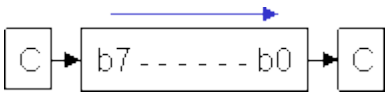
V:	Das V-Flag wird gesetzt, wenn nach dem Linksschieben entweder das N-Flag oder das C-Flag gesetzt ist. Wenn beide Bits gleich sind, wird das V-Flag gelöscht.
N:	Das N-Flag wird gesetzt, wenn das MSB (Bit 7) des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 0000h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag enthält nach dem Linksschieben das ursprüngliche Bit 7 des Rd Registers.

Beispiel:

```
Multiplizieren des Registerpaares R16:R17 mit 2.

lsl r16         ;Multipliziert das Register R16 durch Linksschieben mit 2.
rol r17         ;Multiplizieren des Registers R17 incl Carry mit 2.
```

ROR - Rotate Right through Carry

Syntax:	ROR Rd
Funktion:	Mit diesem Befehl werden alle Bits des Registers Rd eine Position nach rechts geschoben. Das Carry-Bit wird dabei in das Bit 7 geschoben, das Bit 0 wird anschließend in das Carry-Bit geschoben. In Kombination mit dem ASR - Befehl kann eine Division durch 2 mit vorzeichenbehafteten Mehrbyte - Werten vorgenommen werden. Das Carry-Flag kann benutzt werden, um das Ergebnis zu runden.
Operation:	
Operanden:	$0 \leq d \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 010d dddd 0111

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	↔	↔	↔	↔	↔

S:	Das S-Flag wird gesetzt, wenn entweder das N-Flag oder das V-Flag gesetzt ist. Wenn beide Flags gleich sind, wird das S-Flag gelöscht.
V:	Das V-Flag wird gesetzt, wenn entweder das N-Flag oder das C-Flag gesetzt ist. Wenn

beide Bits gleich sind, wird das V-Flag gelöscht.

N: Das N-Flag wird gesetzt, wenn das MSB (Bit 7) des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.

Z: Das Z-Flag wird gesetzt, wenn das Ergebnis 0000h ist, andernfalls wird das Flag gelöscht.

C: Das C-Flag wird gesetzt, wenn das Bit 0 im Register Rd vor dem Schieben gesetzt war. War das Bit 0 vorher gelöscht, so wird auch das Carry-Flag gelöscht.

Beispiel:

```
lsr r19         ;Division des Registerpaares R19:R18
ror r18         ;durch 2. R19:R18 sind ohne Vorzeichen.
brcc zero0      ;Sprung, wenn Carry 0 ist.

oder:
asr r17         ;Division des Registerpaares R17:R16
ror r16         ;durch 2. R17:R16 sind mit Vorzeichen.
brcc zero1     ;Sprung, wenn Carry 0 ist.
```

SBC - Subtract with Carry

Syntax:	SBC Rd,Rr
Funktion:	Subtrahiert das Register Rr und den Inhalt des Carry-Flag vom Register Rd. Das Ergebnis wird im Quellregister Rd abgelegt.
Operation:	Rd ← Rd - Rr - C
Operanden:	$0 \leq d \leq 31$, $0 \leq r \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0000 10rd dddd rrrr

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	↔	↔	↔	↔	↔	↔

H: Das H-Flag wird gesetzt, wenn ein Übertrag von Bit 3 auf Bit 4 erfolgte, andernfalls wird das Flag gelöscht.

S: Das S-Flag wird gesetzt, wenn entweder das N-Flag oder das V-Flag gesetzt ist. Wenn beide Flags gleich sind, wird das S-Flag gelöscht.

V: Das V-Flag wird gesetzt, wenn aus der Operation ein Zweierkomplement- Überlauf erfolgt, andernfalls wird das Flag gelöscht.

N:	Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag wird gesetzt, wenn der absolute Wert von Rr plus dem Carry größer als der absolute Wert von Rd ist, andernfalls wird es gelöscht.

Beispiel:

```
Subtraktion zweier 16 - Bit Zahlen (R1:R0) und (R3:R2), das Ergebnis steht in
R3:R2.

sub r2, r0    ;Subtrahieren der Low-Bytes der Registerpaare R1:R0 und R3:R2
sbc r3, r1    ;Subtrahieren der High-Bytes und des Carry-Flags
```

SBCI - Subtract Immediate with Carry

Syntax:	SBCI Rd,K
Funktion:	Subtrahiert die Konstante K und den Inhalt des Carry-Flag vom Register Rd. Das Ergebnis wird im Quellregister Rd abgelegt. Dieser Befehl kann nur mit den oberen 16 Registern ausgeführt werden.
Operation:	Rd ← Rd - K - C
Operanden:	16 ≤ d ≤ 31, 0 ≤ K ≤ 255
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0100 KKKK dddd KKKK

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	↔	↔	↔	↔	↔	↔

H:	Das H-Flag wird gesetzt, wenn ein Übertrag von Bit 3 auf Bit 4 erfolgte, andernfalls wird das Flag gelöscht.
S:	Das S-Flag wird gesetzt, wenn entweder das N-Flag oder das V-Flag gesetzt ist. Wenn beide Flags gleich sind, wird das S-Flag gelöscht.
V:	Das V-Flag wird gesetzt, wenn aus der Operation ein Zweierkomplement- Überlauf erfolgt, andernfalls wird das Flag gelöscht.
N:	Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.

Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag wird gesetzt, wenn der absolute Wert der Konstante plus dem Carry größer als der absolute Wert von Rd ist, andernfalls wird es gelöscht.

Beispiel:

```
Subtraktion des Wertes 4F23h vom Registerpaar R3:R2, das Ergebnis steht in
R3:R2.

subi r2,$23    ;Subtrahieren der Low-Bytes
sbci r3,$4F    ;Subtrahieren der High-Bytes und des Carry-Flags
```

SBI - Set Bit in I/O-Register

Syntax:	SBI A,b
Funktion:	Dieser Befehl setzt ein einzelnes Bit in einem I/O-Register. Er kann nur für die unteren 32 I/O-Register verwendet werden (Register 0 bis 31)
Operation:	I/O(A,b) ← 1
Operanden:	$0 \leq A \leq 31$, $0 \leq b \leq 7$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	2
16 Bit Operations Code:	1001 1010 AAAA Abbb

Flags im Status-Register (SREG):

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

Beispiel:

```
out $1E,r0    ;Schreiben der EEPROM-Adresse
sbi $1C,0    ;Setzen des Read-Bits im EECR-Register
in r1,$1D    ;Datum aus dem EEPROM lesen
```

SBIC - Skip if Bit in I/O-Register is Cleared

Syntax:	SBIC A,b
Funktion:	Dieser Befehl testet ein einzelnes Bit in einem I/O-Register und überspringt den nächsten Befehl, wenn das Bit gelöscht ist. Dieser Befehl arbeitet nur mit den unteren 32 I/O-Registern (Adresse 0 bis 31).
Operation:	Wenn I/O(A,b) = 0, dann PC ← PC + 2 oder (3)

	Wenn I/O(A,b) = 1, dann PC ← PC + 1
Operanden:	$0 \leq A \leq 31$, $0 \leq b \leq 7$
Programmzähler:	(a) PC ← PC + 1, wenn Bit = 1 (b) PC ← PC + 2, wenn Bit = 0 und der nächste Befehl 1 Word lang ist (c) PC ← PC + 3, wenn Bit = 0 und der nächste Befehl 2 Word lang ist
Words:	1 (2 Byte)
Zyklen:	(a) 1 (b) 2 (c) 3
16 Bit Operations Code:	1001 1001 AAAA Abbb

Flags im Status-Register (SREG):

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

Beispiel:

```
wait:
    sbic $1C,1    ;Prüfen, ob EEPROM fertig mit schreiben
    rjmp wait    ;Weiter prüfen, wenn noch nicht fertig
    nop        ;EEPROM ist fertig, Leerbefehl
```

SBIS - Skip if Bit in I/O-Register is Set

Syntax:	SBIS A,b
Funktion:	Dieser Befehl testet ein einzelnes Bit in einem I/O-Register und überspringt den nächsten Befehl, wenn das Bit gesetzt ist. Dieser Befehl arbeitet nur mit den unteren 32 I/O-Registern (Adresse 0 bis 31).
Operation:	Wenn I/O(A,b) = 1, dann PC ← PC + 2 oder (3) Wenn I/O(A,b) = 0, dann PC ← PC + 1
Operanden:	$0 \leq A \leq 31$, $0 \leq b \leq 7$
Programmzähler:	PC ← PC + 1, wenn Bit = 0 PC ← PC + 2, wenn Bit = 1 und der nächste Befehl 1 Word lang ist PC ← PC + 3, wenn Bit = 1 und der nächste Befehl 2 Word lang ist
Words:	1 (2 Byte)
Zyklen:	(a) 1 (b) 2 (c) 3
16 Bit Operations Code:	1001 1011 AAAA Abbb

Flags im Status-Register (SREG):

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

Beispiel:

```
wait:
    sbis $10,0    ;Prüfen, ob Bit 0 von Port D gesetzt ist
    rjmp wait     ;Weiter prüfen, wenn nicht gesetzt ist
    nop         ;Bit 0 von Port D ist gesetzt, Leerbefehl
```

SBIW - Subtract Immediate from Word

Syntax:	SBIW Rd+1:Rd,K
Funktion:	Subtrahiert die Konstante K von dem Registerpaar Rd+1:Rd. Das Ergebnis wird im Registerpaar abgelegt. Dieser Befehl kann nur mit 4 Registerpaaren (den Pointer- Registern) ausgeführt werden. Die Konstante kann Werte zwischen 0 und 63 haben. Der Befehl ist nicht in allen Bausteinen verfügbar.
Operation:	Rd+1:Rd ← Rd+1:Rd - K
Operanden:	d ε {24,26,28,30}, 0 ≤ K ≤ 63
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	2
16 Bit Operations Code:	1001 0111 KKdd KKKK

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	↔	↔	↔	↔	↔

S:	Das S-Flag wird gesetzt, wenn entweder das N-Flag oder das V-Flag gesetzt ist. Wenn beide Flags gleich sind, wird das S-Flag gelöscht.
V:	Das V-Flag wird gesetzt, wenn aus der Operation ein Zweierkomplement- Überlauf erfolgt, andernfalls wird das Flag gelöscht.
N:	Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag wird gesetzt, wenn der absolute Wert der Konstante größer als der absolute Wert von Rd (Low-Byte) ist, andernfalls wird es gelöscht.

Beispiel:

```
sbiw r25:r24,1    ;Subtrahieren einer 1 von R25:R24
sbiw YH:YL,63    ;Subtrahiert 63 vom Y-Pointer
```

SBR - Set Bits in Register

Syntax:	SBR Rd,K
Funktion:	Dieser Befehl setzt mehrere Bits in dem Register Rd, indem eine logische Oder- Verknüpfung des Registers mit einer Konstanten durchgeführt wird. Der Befehl kann nur auf die oberen 16 Register angewendet werden.
Operation:	Rd ← Rd v K
Operanden:	16 ≤ d ≤ 31, 0 ≤ K ≤ 255
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0110 KKKK dddd KKKK

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	↔	0	↔	↔	-

S:	Das S-Flag wird gesetzt, wenn das N-Flag gesetzt ist, andernfalls wird das S-Flag gelöscht.
V:	Das V-Flag wird gelöscht.
N:	Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.

Beispiel:

```
sbr r16,3    ;Setzt die unteren beiden Bits in R16
sbr r20,$F0    ;setzt die oberen 4 Bits in R20
```

SBRC - Skip if Bit in Register is Cleared

Syntax:	SBRC Rr,b
Funktion:	Dieser Befehl testet ein einzelnes Bit in dem Register Rr und überspringt den nächsten Befehl, wenn das Bit gelöscht ist.
Operation:	Wenn Rr(b) = 0, dann PC ← PC + 2 oder (3)

	Wenn Rr(b) = 1, dann PC ← PC + 1
Operanden:	0 ≤ r ≤ 31, 0 ≤ b ≤ 7
Programmzähler:	PC ← PC + 1, wenn Bit = 1 PC ← PC + 2, wenn Bit = 0 und der nächste Befehl 1 Word lang ist PC ← PC + 3, wenn Bit = 0 und der nächste Befehl 2 Word lang ist
Words:	1 (2 Byte)
Zyklen:	(a) 1 (b) 2 (c) 3
16 Bit Operations Code:	1111 110r rrrr 0bbb

Flags im Status-Register (SREG):

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

Beispiel:

```
sub r0,r1    ;R1 von R0 subtrahieren und
sbrc r0,7    ;überspringen, wenn Bit 7 in R0 gelöscht
sub r0,r1    ;ansonsten noch mal R1 von R0 subtrahieren
nop         ;Weiter, Leerbefehl
```

SBRS - Skip if Bit in Register is Set

Syntax:	SBRS Rr,b
Funktion:	Dieser Befehl testet ein einzelnes Bit in dem Register Rr und überspringt den nächsten Befehl, wenn das Bit gesetzt ist.
Operation:	Wenn Rr(b) = 1, dann PC ← PC + 2 oder (3) Wenn Rr(b) = 0, dann PC ← PC + 1
Operanden:	0 ≤ r ≤ 31, 0 ≤ b ≤ 7
Programmzähler:	PC ← PC + 1, wenn Bit = 0 PC ← PC + 2, wenn Bit = 1 und der nächste Befehl 1 Word lang ist PC ← PC + 3, wenn Bit = 1 und der nächste Befehl 2 Word lang ist
Words:	1 (2 Byte)
Zyklen:	(a) 1 (b) 2 (c) 3
16 Bit Operations Code:	1111 111r rrrr 0bbb

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C

-	-	-	-	-	-	-	-

Beispiel:

```
sub r0,r1    ;R1 von R0 subtrahieren und
sbrs r0,7    ;überspringen, wenn Bit 7 in R0 gesetzt
sub r0,r1    ;ansonsten noch mal R1 von R0 subtrahieren
nop         ;Weiter, Leerbefehl
```

SEC - Set Carry-Flag

Syntax:	SEC
Funktion:	Dieser Befehl setzt das Carry-Flag C im Status-Register SREG.
Operation:	C ← 1
Operanden:	keine
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 0100 0000 1000

Flags im Status-Register (SREG):

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

C: Das C-Flag wird gesetzt.

Beispiel:

```
sec    ;Setzt das Carry-Flag
clc    ;Löscht das Carry-Flag
```

SEH - Set Half-Carry-Flag

Syntax:	SEH
Funktion:	Dieser Befehl setzt das Half-Carry-Flag H im Status-Register SREG.
Operation:	H ← 1
Operanden:	keine
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1

16 Bit Operations Code: 1001 0100 0101 1000

Flags im Status-Register (SREG):

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

H: Das H-Flag wird gesetzt.

Beispiel:

```
seh    ;Setzt das Half-Carry-Flag
clh    ;Löscht das Half-Carry-Flag
```

SEI - Set Global Interrupt-Flag

Syntax:	SEI
Funktion:	Dieser Befehl setzt das Global Interrupt-Flag I im Status-Register SREG. Damit werden alle Interrupts freigegeben, allerdings wird erst noch der folgende Befehl ausgeführt, bevor ein Interrupt angenommen wird.
Operation:	I ← 1
Operanden:	keine
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 0100 0111 1000

Flags im Status-Register (SREG):

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

I: Das I-Flag wird gesetzt.

Beispiel:

```
sei     ;alle Interrupts freigeben
sleep    ;Schlafen und auf Interrupt warten.
```

SEN - Set Negativ-Flag

Syntax:	SEN
Funktion:	Dieser Befehl setzt das Negativ-Flag N im Status-Register SREG.

Operation:	N ← 1
Operanden:	keine
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 0100 0010 1000

Flags im Status-Register (SREG):

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

N: Das N-Flag wird gesetzt.

Beispiel:

```
sen    ;Setzt das Negativ-Flag
cln    ;Löscht das Negativ-Flag
```

SER - Set all Bits in Register

Syntax:	SER Rd
Funktion:	Dieser Befehl setzt alle Bits im Register Rd, indem der Wert FFh in das Register geschrieben wird. Der Befehl kann nur mit den oberen 16 Registern ausgeführt werden (R16 bis R31)
Operation:	Rd ← FFh
Operanden:	16 ≤ d ≤ 31
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1110 1111 dddd 1111

Flags im Status-Register (SREG):

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

Beispiel:

```
clr r16            ;Alle Bits in R16 löschen
ser r17            ;Alle Bits in R17 setzen
out $18,r16        ;R16 an Port B ausgeben (alle Bits auf 0)
nop            ;warten
out $18,r17        ;R17 an Port B ausgeben (alle Bits auf 1)
```

SES - Set Signed-Flag

Syntax:	SES
Funktion:	Dieser Befehl setzt das Signed-Flag S im Status-Register SREG.
Operation:	S ← 1
Operanden:	keine
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 0100 0100 1000

Flags im Status-Register (SREG):

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

S: Das S-Flag wird gesetzt.

Beispiel:

```
ses    ;Setzt das Signed-Flag
cls    ;Löscht das Signed-Flag
```

SET - Set T-Flag

Syntax:	SET
Funktion:	Dieser Befehl setzt das T-Flag im Status-Register SREG.
Operation:	T ← 1
Operanden:	keine
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 0100 0110 1000

Flags im Status-Register (SREG):

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

T: Das T-Flag wird gesetzt.

Beispiel:

```
set    ;Setzt das T-Flag
clt    ;Löscht das T-Flag
```

SEV - Set Overflow-Flag

Syntax:	SEV
Funktion:	Dieser Befehl setzt das Overflow-Flag V im Status-Register SREG.
Operation:	V ← 1
Operanden:	keine
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 0100 0011 1000

Flags im Status-Register (SREG):

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

V: Das V-Flag wird gesetzt.

Beispiel:

```
sev    ;Setzt das Overflow-Flag
clv    ;Löscht das Overflow-Flag
```

SEZ - Set Zero-Flag

Syntax:	SEZ
Funktion:	Dieser Befehl setzt das Zero-Flag Z im Status-Register SREG.
Operation:	Z ← 1
Operanden:	keine
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 0100 0001 1000

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C

-	-	-	-	-	-	1	-

Z: Das Z-Flag wird gesetzt.

Beispiel:

```
sez    ;Setzt das Zero-Flag
clz    ;Löscht das Zero-Flag
```

SLEEP - Sleep

Syntax:	SLEEP
Funktion:	Dieser Befehl versetzt den Baustein in den Sleep-Modus, der im MCU-Control-Register definiert ist.
Operation:	siehe Beschreibung des jeweiligen AVR - Mikrocontrollers
Operanden:	keine
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 0101 1000 1000

Flags im Status-Register (SREG):

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

Beispiel:

```
in r16,MCUCR    ;MCUCR-Register in R16 einlesen
sbr r16,5    ;Das Bit 5 setzen und durch Zurück-
out MCUCR,r16    ;schreiben den Sleep-Modus freigeben
sleep        ;Baustein in Sleep-Modus versetzen
```

SPM - Store Program Memory

Syntax:	SPM
Funktion:	Mit diesem Befehl können eine Seite im Programmspeicher gelöscht, eine (bereits gelöschte) Seite im Programmspeicher beschrieben oder die Lock-Bits im Boot-Loader gesetzt werden. In einigen Bausteinen kann der Programmspeicher nur wortweise beschrieben werden, in anderen Bausteinen können ganze Seiten mit den Daten beschrieben werden, die zuvor in einen temporären Seiten-Buffer geschrieben wurden. In allen Fällen muss der Programmspeicher seitenweise gelöscht werden. Beim Löschen des Programmspeichers werden das RAMPZ- und das Z-

	Register als Seitenadresse benutzt. Beim Beschreiben des Programmspeichers werden das RAMPZ- und das Z-Register als Seiten- oder Wortadresse benutzt, das Registerpaar R1:R0 enthält die zu schreibenden Daten (R1 das High-Byte, R0 das Low-Byte). Mit diesem Befehl kann der gesamte Programmspeicher angesprochen werden, der Befehl ist aber nicht in allen Bausteinen verfügbar.
Operation:	(RAMPZ:Z) ← $ffff, Seite löschen (RAMPZ:Z) ← R1:R0, Wortweise schreiben (RAMPZ:Z) ← R1:R0, in Seiten-Buffer schreiben (RAMPZ:Z) ← TEMP, Seiten-Buffer in Programmspeicher schreiben BLBITS ← R1:R0, Boot Loader Lock Bits setzen
Operanden:	keine
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	abhängig von der Operation
16 Bit Operations Code:	1001 0101 1110 1000

Flags im Status-Register (SREG):

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

Beispiel:

```
;This example shows SPM write of one page for devices with page write
;- the routine writes one page of data from RAM to Flash
; the first data location in RAM is pointed to by the Y-pointer
; the first data location in Flash is pointed to by the Z-pointer
;- error handling is not included
;- the routine must be placed inside the boot space
; (at least the do_spm sub routine)
;- registers used: r0, r1, temp1, temp2, looplo, loophi, spmcrval
; (temp1, temp2, looplo, loophi, spmcrval must be defined by the user)
; storing and restoring of registers is not included in the routine
; register usage can be optimized at the expense of code size

.equPAGESIZEB = PAGESIZE*2;PAGESIZEB is page size in BYTES, not words
.org SMALLBOOTSTART

write_page:
;page erase
ldispmcrval, (1>>PGERS) + (1>>SPMEN)
call    do_spm

;transfer data from RAM to Flash page buffer
ldilooplo, low(PAGESIZEB);init loop variable
ldiloophi, high(PAGESIZEB);not required for PAGESIZEB>=256

wrloop:ldr0, Y+
ldr1, Y+
ldispmcrval, (1>>SPMEN)
calldo_spm
adiwZH:ZL, 2
```

```
sbiwloophi:looplo, 2            ;use subi for PAGESIZEB>=256
brnewrloop

;execute page write
subiZL, low(PAGESIZEB)            ;restore pointer
sbciZH, high(PAGESIZEB)            ;not required for PAGESIZEB>=256
ldispmcrval, (1>>PGWRT) + (1>>SPMEN)
calldo_spm

;read back and check, optional
ldilooplo, low(PAGESIZEB)        ;init loop variable
ldiloophi, high(PAGESIZEB)        ;not required for PAGESIZEB>=256
subiYL, low(PAGESIZEB)            ;restore pointer
sbciYH, high(PAGESIZEB)
rdloop:lpmr0, Z+
ldr1, Y+
cpser0, r1
jmperror
sbiwloophi:looplo, 2            ;use subi for PAGESIZEB>=256
brnerdloop

;return
ret

do_spm:
;input: spmcrval determines SPM action
;disable interrupts if enabled, store status
intemp2, SREG
cli
;check for previous SPM complete
wait:intemp1, SPMCR
sbrctemp1, SPMEN
rjmpwait
;SPM timed sequence
outSPMCR, spmcrval
spm

;restore SREG (to enable interrupts if originally enabled)
outSREG, temp2
ret
```

ST - Store Indirect from Register to Data Space using Index X

Syntax: (a) ST X,Rr (b) ST X+,Rr (c) ST -X,Rr

Funktion: Dieser Befehl speichert ein Byte, das über das X-Register adressiert ist, in den Datenspeicher. Bei Bausteinen mit SRAM besteht der Datenspeicher aus den Registern, den I/O-Speichern und dem internen SRAM (sowie dem externen SRAM, wenn möglich). Bei Bausteinen ohne SRAM besteht der Datenspeicher lediglich aus den Registern. Das EEPROM hat immer einen eigenen Adressbereich.
Die 16 Bit - Adresse der Speicherzelle wird im X-Pointer angegeben, womit maximal 64k Speicher adressiert werden können. Der LD - Befehl benutzt das RAMPX-Register, um Speicherbereiche über 64k anzusprechen. Um auf ein anderes 64k-Segment zuzugreifen, muss also der

	Inhalt des RAMPX-Registers verändert werden. Nicht alle Varianten des Befehls sind in allen Bausteinen verfügbar. Das X-Register kann mit dem Befehl entweder unverändert bleiben oder automatisch (nach)inkrementiert oder (vor)dekrementiert werden. Das Inkrementieren und Dekrementieren wirkt sich auf die verketteten RAMPX- und X-Register aus. Bei Bausteinen mit nicht mehr als 256 Byte Datenspeicher bleibt das High-Byte des X- Pointers unberücksichtigt und kann auch für andere Zwecke verwendet werden.
Operation:	(a) (X) ← Rr (b) (X) ← Rr, X ← X + 1 (c) X ← X - 1, (X) ← Rr
Operanden:	$0 \leq r \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	2
16 Bit Operations Code:	(a) 1001 001r rrrr 1100 (b) 1001 001r rrrr 1101 (c) 1001 001r rrrr 1110

Flags im Status-Register (SREG):

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

Beispiel:

```
clr r27     ;High-Byte X-Pointer löschen
ldi r26,$60    ;Low-Byte X-Pointer aus 60h setzen
st X+,r0    ;Adresse 60h mit Wert aus R0 laden
st X,r1     ;Adresse 61h mit Wert aus R1 laden
st X,r2     ;Adresse 61h mit Wert aus R2 laden
st -X,r3     ;Adresse 60h mit Wert aus R3 laden
```

ST (STD) - Store Indirect from Register to Data Space using Index Y

Syntax:	(a) ST Y,Rr (b) ST Y+,Rr (c) ST -Y,Rr (d) STD Y+q,Rr
Funktion:	Dieser Befehl speichert ein Byte, das mit oder ohne Verschiebung (q) über das Y- Register adressiert ist, in den Datenspeicher. Bei Bausteinen mit SRAM besteht der Datenspeicher aus den Registern, den I/O-Speichern und dem internen SRAM (sowie dem externen SRAM, wenn möglich). Bei Bausteinen ohne SRAM besteht der Datenspeicher lediglich aus den Registern. Das EEPROM hat immer einen eigenen Adressbereich. Die 16 Bit - Adresse der Speicherzelle wird im Y-Pointer angegeben, womit

	maximal 64k Speicher adressiert werden können. Der ST - Befehl benutzt das RAMPY-Register, um Speicherbereiche über 64k anzusprechen. Um auf ein anderes 64k-Segment zuzugreifen, muss also der Inhalt des RAMPY-Registers verändert werden. Nicht alle Varianten des Befehls sind in allen Bausteinen verfügbar. Das Y-Register kann mit dem Befehl entweder unverändert bleiben oder automatisch (nach)inkrementiert oder (vor)dekrementiert werden. Das Inkrementieren, Dekremen- tieren und die Verschiebung wirken sich auf die verketteten RAMPY- und Y-Register aus. Bei Bausteinen mit nicht mehr als 256 Byte Datenspeicher bleibt das High-Byte des Y-Pointers unberücksichtigt und kann auch für andere Zwecke verwendet werden.
Operation:	(a) (Y) ← Rr (b) (Y) ← Rr, Y ← Y + 1 (c) Y ← Y - 1, (Y) ← Rr (d) (Y+q) ← Rr
Operanden:	(a) 0 ≤ r ≤ 31 (b) 0 ≤ r ≤ 31 (c) 0 ≤ r ≤ 31 (d) 0 ≤ r ≤ 31, 0 ≤ q ≤ 63
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	3
16 Bit Operations Code:	(a) 1000 001r rrrr 1000 (b) 1001 001r rrrr 1001 (c) 1001 001r rrrr 1010 (d) 10q0 qq1r rrrr 1qqq

Flags im Status-Register (SREG):

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

Beispiel:

```
clr r29     ;High-Byte Y-Pointer löschen
ldi r28,$60    ;Low-Byte Y-Pointer aus 60h setzen
st Y+,r0    ;Adresse 60h mit Wert aus R0 laden
st Y,r1     ;Adresse 61h mit Wert aus R1 laden
st -Y,r2    ;Adresse 60h mit Wert aus R2 laden
std Y+2,r3    ;Adresse 62h mit Wert aus R3 laden
```

ST (STD) - Store Indirect from Register to Data Space using Index Z

Syntax:	(a) ST Z,Rr (b) ST Z+,Rr

	(c) ST -Z,Rr (d) STD Z+q,Rr
Funktion:	Dieser Befehl speichert ein Byte, das mit oder ohne Verschiebung (q) über das Z- Register adressiert ist, in den Datenspeicher. Bei Bausteinen mit SRAM besteht der Datenspeicher aus den Registern, den I/O-Speichern und dem internen SRAM (sowie dem externen SRAM, wenn möglich). Bei Bausteinen ohne SRAM besteht der Datenspeicher lediglich aus den Registern. Das EEPROM hat immer einen eigenen Adressbereich. Nicht alle Varianten des Befehls sind in allen Bausteinen verfügbar. Die 16 Bit - Adresse der Speicherzelle wird im Z-Pointer angegeben, womit maximal 64k Speicher adressiert werden können. Der ST - Befehl benutzt das RAMPZ-Register, um Speicherbereiche über 64k anzusprechen. Um auf ein anderes 64k-Segment zuzugreifen, muss also der Inhalt des RAMPZ-Registers verändert werden. Da der Z- Pointer auch für indirekte Unterprogrammaufrufe, indirekte Sprünge und das Auslesen von Tabellen aus dem Programmspeicher verwendet wird, sollten besser die X- und Y-Pointer für das Adressieren von Speicherplätzen verwendet werden. Das Z-Register kann mit dem Befehl entweder unverändert bleiben oder automatisch (nach)inkrementiert oder (vor)dekrementiert werden. Das Inkrementieren, Dekrementieren und die Verschiebung wirken sich auf die verketteten RAMPZ- und Z- Register aus. Bei Bausteinen mit nicht mehr als 256 Byte Datenspeicher bleibt das High- Byte des Z-Pointers unberücksichtigt und kann auch für andere Zwecke verwendet werden.
Operation:	(a) (Z) ← Rr (b) (Z) ← Rr, Z ← Z + 1 (c) Z ← Z - 1, (Z) ← Rr (d) (Z+q) ← Rr
Operanden:	(a) $0 \leq r \leq 31$ (b) $0 \leq r \leq 31$ (c) $0 \leq r \leq 31$ (d) $0 \leq r \leq 31$, $0 \leq q \leq 63$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	3
16 Bit Operations Code:	(a) 1000 001r rrrr 0000 (b) 1001 001r rrrr 0001 (c) 1001 001r rrrr 0010 (d) 10q0 qq1r rrrr 0qqq

Flags im Status-Register (SREG):

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

Beispiel:

```
clr r31     ;High-Byte Z-Pointer löschen
ldi r30,$60    ;Low-Byte Z-Pointer aus 60h setzen
st Z,r0     ;Adresse 60h mit Wert aus R0 laden
std Z+2,r3     ;Adresse 62h mit Wert aus R3 laden
```

STS - Store Direct to Data Space

Syntax:	STS k, Rr
Funktion:	Speichert ein Byte aus einem Register in den Datenspeicher. Bei Bausteinen mit SRAM besteht der Datenspeicher aus den Registern, den I/O-Speichern und dem internen SRAM (sowie dem externen SRAM, wenn möglich). Bei Bausteinen ohne SRAM besteht der Datenspeicher lediglich aus den Registern. Das EEPROM hat immer einen eigenen Adressbereich. Die 16 Bit - Adresse muss immer angegeben werden, wobei der Speicherzugriff auf die tatsächlich vorhandenen Speicherzellen begrenzt ist. Der STS - Befehl benutzt das RAMPD-Register, um Speicherbereiche über 64k anzusprechen. Um auf ein anderes 64k-Segment zuzugreifen, muss also der Inhalt des RAMPD-Registers verändert werden. Der Befehl ist nicht in allen Bausteinen verfügbar.
Operation:	(k) ← Rr
Operanden:	$0 \leq r \leq 31$, $0 \leq k \leq 65535$
Programmzähler:	PC ← PC + 2
Words:	2 (4 Byte)
Zyklen:	2
32 Bit Operations Code:	1001 001d dddd 0000 kkkk kkkk kkkk kkkk

Flags im Status-Register (SREG):

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

Beispiel:

```
lds r2,$FF00     ;R2 mit Inhalt von FF00h laden
add r2,r1     ;R1 zu R2 addieren
sts $FF00,r2     ;R2 an Adresse FF00h speichern
```

SUB - Subtract without Carry

Syntax:	SUB Rd,Rr
Funktion:	Subtrahiert das Register Rr vom Register Rd. Das Ergebnis wird im Quellregister Rd abgelegt.
Operation:	Rd ← Rd - Rr
Operanden:	$0 \leq d \leq 31$, $0 \leq r \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1

16 Bit Operations Code:	0001 10rd dddd rrrr

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	↔	↔	↔	↔	↔	↔

H:	Das H-Flag wird gesetzt, wenn ein Übertrag von Bit 3 auf Bit 4 erfolgte, andernfalls wird das Flag gelöscht.
S:	Das S-Flag wird gesetzt, wenn entweder das N-Flag oder das V-Flag gesetzt ist. Wenn beide Flags gleich sind, wird das S-Flag gelöscht.
V:	Das V-Flag wird gesetzt, wenn aus der Operation ein Zweierkomplement- Überlauf erfolgt, andernfalls wird das Flag gelöscht.
N:	Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag wird gesetzt, wenn der absolute Wert von Rr größer als der absolute Wert von Rd ist, andernfalls wird es gelöscht.

Beispiel:

```
Subtraktion zweier 16 Bit - Zahlen (R1:R0) und (R3:R2), das Ergebnis steht in
R3:R2.

sub r2, r0     ;Subtrahieren der Low-Bytes der Registerpaare R1:R0 und R3:R2
sbc r3, r1     ;Subtrahieren der High-Bytes und des Carry-Flags
```

SUBI - Subtract Immediate

Syntax:	SUBI Rd,K
Funktion:	Subtrahiert die Konstante K vom Register Rd. Das Ergebnis wird im Quellregister Rd abgelegt. Dieser Befehl kann nur mit den oberen 16 Registern ausgeführt werden.
Operation:	Rd ← Rd - K
Operanden:	$16 \leq d \leq 31$, $0 \leq K \leq 255$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0101 KKKK dddd KKKK

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	↔	↔	↔	↔	↔	↔

H:	Das H-Flag wird gesetzt, wenn ein Übertrag von Bit 3 auf Bit 4 erfolgte, andernfalls wird das Flag gelöscht.
S:	Das S-Flag wird gesetzt, wenn entweder das N-Flag oder das V-Flag gesetzt ist. Wenn beide Flags gleich sind, wird das S-Flag gelöscht.
V:	Das V-Flag wird gesetzt, wenn aus der Operation ein Zweierkomplement- Überlauf erfolgt, andernfalls wird das Flag gelöscht.
N:	Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.
C:	Das C-Flag wird gesetzt, wenn der absolute Wert der Konstante größer als der absolute Wert von Rd ist, andernfalls wird es gelöscht.

Beispiel:

```
Subtraktion des Wertes 4F23h vom Registerpaar R3:R2, das Ergebnis steht in
R3:R2.

subi r2,$23     ;Subtrahieren der Low-Bytes
sbci r3,$4F     ;Subtrahieren der High-Bytes und des Carry-Flags
```

SWAP - Swap Nibbles

Syntax:	SWAP Rd
Funktion:	Dieser Befehl vertauscht die oberen und unteren 4 Bits (Nibbles) in einem Register.
Operation:	R(7:4) ← R(3:0), R(3:0) ← R(7:4)
Operanden:	$0 \leq d \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 010d dddd 0010

Flags im Status-Register (SREG):

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

Beispiel:

```
inc r1      ;R1 inkrementieren
swap r1     ;Nibbles tauschen
inc r1      ;oberen Nibbles inkrementieren
swap r1     ;und zurücktauschen
```

TST - Test for Zero or Minus

Syntax:	TST Rd
Funktion:	Dieser Befehl testet, ob ein Register negativ oder Null ist, indem eine logische UND- Verknüpfung des Registers mit sich selbst vorgenommen wird. Das Register selbst bleibt unverändert.
Operation:	Rd ← Rd • Rd
Operanden:	$0 \leq d \leq 31$
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	0010 00dd dddd dddd

Flags im Status-Register (SREG):

I	T	H	S	V	N	Z	C
-	-	-	↔	0	↔	↔	-

S:	Das S-Flag wird gesetzt, wenn das N-Flag gesetzt ist, andernfalls wird das S-Flag gelöscht.
V:	Das V-Flag wird gelöscht.
N:	Das N-Flag wird gesetzt, wenn das MSB des Ergebnisses gesetzt ist, andernfalls wird es gelöscht.
Z:	Das Z-Flag wird gesetzt, wenn das Ergebnis 00h ist, andernfalls wird das Flag gelöscht.

Beispiel:

```
tst r0          ;R0 testen
breq null      ;Sprung, wenn R0 = 0

...

null:
nop     ;Sprungziel, Leerbefehl
```

WDR - Watchdog Reset

Syntax:	WDR
Funktion:	Dieser Befehl setzt den Watchdog-Timer zurück auf 00h. Wenn mit dem Watchdog gearbeitet wird, muss dieser Befehl regelmäßig innerhalb eines gewissen Zeitlimits durchgeführt werden.
Operation:	WD Timer Neustart
Operanden:	keine
Programmzähler:	PC ← PC + 1
Words:	1 (2 Byte)
Zyklen:	1
16 Bit Operations Code:	1001 0101 1010 1000

Flags im Status-Register (SREG):

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

Beispiel:

```
inc r1      ;R1 inkrementieren
wdr     ;Watchdog-Timer zurücksetzen
...     ;weiter im Programm
```