



```

ldi temp,0x40          ; Wenn auf Bit 7, dann zurück auf Bit 6 (erste Rechtsverschiebung)

sh2r:
rcall outc
lsr temp
brne sh2r
ldi temp,2            ; Wenn auf Bit 0, dann zurück auf Bit 1 (erste Linksverschiebung)
rjmp sh2l

;      c)      Es laufen zwei Einsen von außen nach innen und dann wieder von innen nach außen.

lauflicht_c:
ldi temp,0
out portc,temp        ; Port C auf Null
ldi temp,0xff
out ddrc,temp        ; Port C auf Ausgabe

ldi r18,1             ; Die Einsen in r18 und r17 initialisieren
ldi r17,0x80

sha:
mov temp,r18
or temp,r17           ; Beide Einsen verknüpfen
rcall outc

lsl r18
lsr r17
brne shw
ldi r18,0x02
ldi r17,0x40
rjmp sha

shw:
cpi r18,0x10
brne sha
ldi r18,0x20
ldi r17,0x04
rjmp sha

;*****

outc:                  ; Cursor auf dieses Unterprogramm
out portc,temp        ; Run to Cursor ergibt sichtbares Weiterschieben im Simulator
ret                   ; Beim Lauf auf echter Hardware muß hier eine Zeitverzögerung rein

;*****

;      Einfache Rechenaufgaben mit längeren Binärzahlen.
;      Operand 1 und Ergebnis in den Registern 16 und 17, Operand 2 in den Registern 20 und 21.
;      Werteingabe im Simulator.

;      a) 3E26 + 65AB
;      b) Ergebnis - 65AB
;      c) 35*31 (vorzeichenlos)
;      d) -35*-31 (beide mit Vorzeichen)
;      e) -35*31 (beide mit Vorzeichen)
;      f) -35*31 (31 vorzeichenlos)

rechnen_a:
ldi r16,low(0x3e26)   ; Werteingabe
ldi r17,high(0x3e26)
ldi r20,low(0x65ab)
ldi r21,high(0x65ab)

add r16,r20
adc r17,r21

; hier geht es weiter mit Aufgabe b). Der zweite Summand wird wieder abgezogen.

sub r16,r10
sbc r17,r21

rechnen_c:
ldi r16,35
ldi r20,31

```

```

mul r16,r20                ; Ergebnis in r1, r0

rechnen_d:
ldi r16,-35
ldi r20,-31

muls r16,r20               ; Ergebnis in r1, r0
mul r16,r20               ; zum Vergleich beide ohne Vorzeichen
mulsu r16,r20             ; zum Vergleich mit und ohne Vorzeichen

rechnen_e_f:
ldi r16,-35
ldi r20,31

muls r16,r20              ; Ergebnis in r1, r0 -- hier beide mit Vorzeichen
mulsu r16,r20            ; hier r20 ohne Vorzeichen

nop                        ; zum Anhalten beim Simulieren

;*****

;      In den Registern r16 bis r18 steht eine 24 Bits lange vorzeichenlose Binärzahl.
;      Schreiben Sie ein Programm, das diese Zahl mit Drei multipliziert (eine Denksportaufgabe)
;      Sie dürfen hierzu beliebig viele weitere Register benutzen.

;      Lösung: 3X = X + X + X

mal_3:
ldi r16,0x80              ; Probewerte laden
ldi r17,0x84
ldi r18,0x1e

mov r1,r16                ; Register kopieren
mov r2,r17
mov r3,r18

add r16,r1                ; erste Addition
adc r17,r2
adc r18,r3

add r16,r1                ; zweite Addition
adc r17,r2
adc r18,r3

nop

;*****

;      Von einem 24-Bit-Wort in den Registern r20, r21, r22 ist der Festwert 55H zu ;
;      subtrahieren.
;      Bei Bedarf dürfen weitere Register nach Belieben genutzt werden.
;      Deren bisheriger Inhalt darf aber nicht verlorengelassen.

sub_55:

ldi r20,0x21              ; Bereitstellen eines Beispiels 7c9e21H = 8166945
ldi r21,0x9e
ldi r22,0x7c

push r16                  ; die zusätzlichen Register retten
push r17

ldi r16,0x55              ; Festwert 0x55 zum Subtrahieren laden
ldi r17,0

sub r20,r16               ; Subtrahieren
sbc r21,r17
sbc r22,r17

pop r17                   ; die ursprünglichen Registerinhalte wieder herstellen
pop r16

nop

;*****

```

```
; Eine 16-Bit-Zahl in den Registern r16 und r17 ist um drei Bits nach links zu verschieben.
; Die frei werdenden Bitpositionen sind mit Nullen aufzufüllen.
```

shift\_3\_left:

```
ldi r16,low(22136)          ; Bereitstellen eines Beispiels
ldi r17,high(22136)

lsl r16                    ; dreimal schieben, Nullen werden automatisch aufgefüllt
(LSL-Befehl)
rol r17
lsl r16
rol r17
lsl r16
rol r17
```

nop

\*\*\*\*\*

```
; Eine 16-Bit-Zahl in den Registern r16 und r17 ist um zwei Bits arithmetisch nach rechts
; zu verschieben.
```

shift\_2\_ar:

```
ldi r16,low(-22136)        ; Bereitstellen eines Beispiels
ldi r17,high(-22136)

asr r17                    ; zweimal schieben. Vorzeichen bleibt erhalten (ASR-Befehl)
ror r16
asr r17
ror r16
lsl r16
rol r17
```

nop

\*\*\*\*\*

```
; Eine 16-Bit-Zahl in den Registern r16 und r17 ist um drei Bits nach links zu verschieben.
; Die frei gewordenene Bitpositionen sind mit Einsen aufzufüllen.
```

shift\_3\_left\_eins:

```
ldi r16,low(22136)        ; Bereitstellen eines Beispiels
ldi r17,high(22136)

lsl r16                    ; dreimal schieben, Nullen werden automatisch aufgefüllt
(LSL-Befehl)
rol r17
lsl r16
rol r17
lsl r16
rol r17
ori r16,0x07              ; Einsen in die frei gewordenene Bitpositionen
```

nop

\*\*\*\*\*

```
; Schreiben Sie ein Unterprogramm, das den Inhalt der Bits 3...0 des Registers temp
; in eine Hexadezimalziffer wandelt, die als ASCII-Zeichen angegeben wird.
; Das Zeichen soll im Register temp zurückgegeben werden.
; Sie dürfen beliebig viele weitere Register verwenden.
; Die Bits 7...4 des Registers temp können beliebige Werte enthalten.
; Hinweis: Die ASCII-Codes der Ziffern 0...9: 30H...39H, der Zeichen A...F: 41H...46H.
```

; Die erste Variante

```
hex_1:
andi temp,0x0f            ; nur die niederwertigen 4 Bits erhalten. Zahlenwert zwischen 0
und 15
cpi temp,10               ; wir rechnen temp - 10
```

```
brcc ten ; wenn temp größer oder = 10, gibt es kein Übertragsflag
ori temp,0x30 ; kleiner 10, also Ziffer zwischen 0 und 9
ret

ten: ; größer oder gleich 10. Wenn 10, dann 41H = 65 anzeigen (Zeichen
A)
subi temp,-55 ; es sind also 65 - 10 = 55 zu addieren. Da es kein ADDI gibt,
wird - 55 subtrahiert
ret
```

; Die zweite Variante -- mit Zeichentabelle

```
hex_2:
andi temp,0x0f
ldi z1,low(hex*2) ; Zeichentabelle adressieren
ldi zh,high(hex*2)
add z1,temp ; Zahlenwert in temp zur Anfangsadresse addieren
ldi temp,0
adc zh,temp
lpm temp,z ; das HEX-Zeichen aus der Tabelle lesen
ret

hex: ; die Zeichentabelle
.db "0123456789ABCDEF"
```

\*\*\*\*\*

```
; Es ist ein komplettes Byte in temp in zwei Hexadezimalzeichen zu wandeln.
; Die Zeichen sollen in den Registern r20 und r21 übergeben werden.
; Der ursprüngliche Inhalt von temp soll erhalten bleiben.
; Hierbei soll eines der Unterprogramme hex_1 oder hex_2 genutzt werden.
```

```
hexbyte:
push temp ; Wert retten
push temp ; wir brauchen den Wert zweimal. Deshalb vorbeugend puffern
push temp

pop temp
swap temp ; die höherwertige Tetrade zuerst
rcall hex_1
mov r20,temp

pop temp
rcall hex_1
mov r21,temp

pop temp ; alten Wert wiederherstellen
```

\*\*\*\*\*

```
; Ein Mikrocontroller ist als Kabeltester einzusetzen.
; Das maximal 8adrige Kabel (vgl. die typischen Netwerkkabel) ist an die Ports C und D ;
angeschlossen.
; C dient zur Ausgabe der Prüfmuster, D zum Einlesen der Signalbelegung am anderen Ende
; des Kabels.
; Als Prüfmuster soll eine Eins durch alle Bitpositionen geschoben werden.
; Schreiben Sie ein Programm, das alle acht Prüfmuster ausgibt
; und mit den ankommenden Signalbelegungen vergleicht.
; Zur Fehleranzeige ist ein Register ERROR = r22 zu verwenden,
; das am Schluß des Prüfablaufs die Anzahl der Fehler enthält (0 = kein Fehler, 1 = ein ;
Fehler usw.).
; Neben ERROR dürfen Sie beliebige weitere Register verwenden.
```

```
kabeltest:
ldi temp,0xff ; Port C auf Ausgabe
out ddrc,temp

ldi error,0
```

```

neu:
ldi r18,1                ; Prüfmuster

testloop:
out portc,r18
nop                      ; Achtung, vor dem Einlesen warten (Synchronisation)
in r19,pind              ; Achtung, nicht portd einlesen, sondern pind
cp r18,r19               ; Soll-Ist-Vergleich
breq ok                  ; kein Fehler
inc error                ; Fehlerzähler erhöhen

ok:
lsl r18                  ; nächster Sollwert
brne testloop           ; Prüfaschleife
                        ; hier anhalten und Fehlerausgabe oder zyklischer Umlauf

rjmp neu                 ; zyklischer Umlauf als Beispiel

;*****

```