

# Anwendungsorientierte Softwareentwicklung

## Laborblatt 2: Blinkende LED

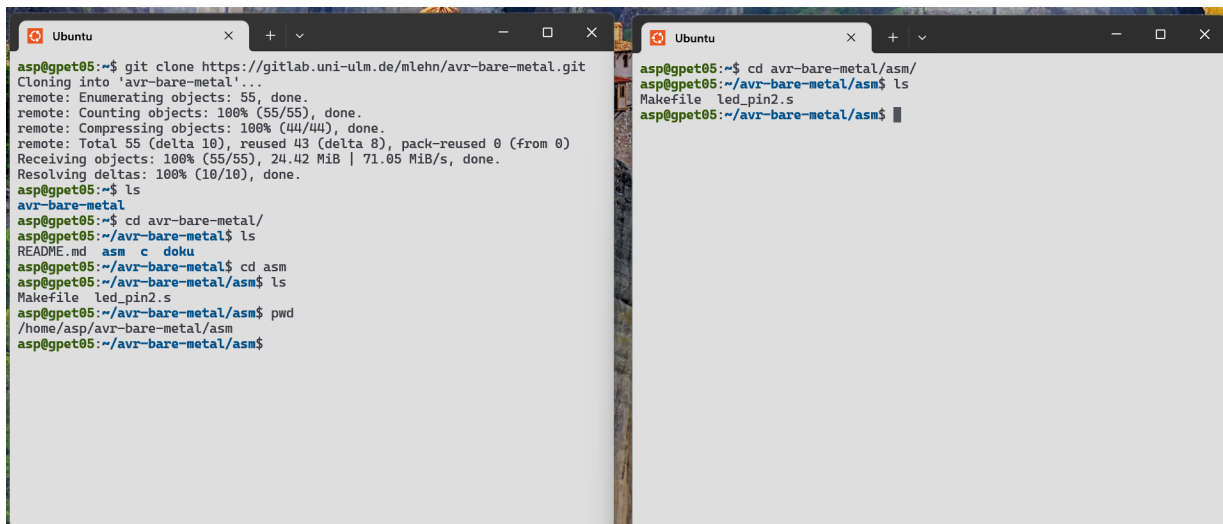
### Lernziele

Ihr habt im letzten Laborblatt Maschinencode von Hand erstellt. Heute zeigen wir, wie man aus einem Assemblerprogramm automatisch denselben Maschinencode erzeugt – und diesen dann auf den Mikrocontroller flasht. Außerdem übt ihr den Umgang mit dem Terminal.

### Schritt 1: Arbeiten mit dem Terminal

**Ziel:** Ihr sollt lernen, euch im Terminal zurechtzufinden, mit Git ein Repository zu klonen und im Dateisystem zu navigieren.

Wir starten mit zwei Terminals nebeneinander – eines zum Editieren, eines zum Ausführen von Kommandos.



```
asp@gpet05:~$ git clone https://gitlab.uni-ulm.de/mlehn/avr-bare-metal.git
Cloning into 'avr-bare-metal'...
remote: Enumerating objects: 55, done.
remote: Counting objects: 100% (55/55), done.
remote: Compressing objects: 100% (44/44), done.
remote: Total 55 (delta 10), reused 43 (delta 8), pack-reused 0 (from 0)
Receiving objects: 100% (55/55), 24.42 MiB | 71.05 MiB/s, done.
Resolving deltas: 100% (10/10), done.
asp@gpet05:~$ ls
avr-bare-metal
asp@gpet05:~$ cd avr-bare-metal/
asp@gpet05:~/avr-bare-metal$ ls
README.md  asm  c  doku
asp@gpet05:~/avr-bare-metal$ cd asm
asp@gpet05:~/avr-bare-metal/asm$ ls
Makefile  led_pin2.s
asp@gpet05:~/avr-bare-metal/asm$ pwd
/home/asp/avr-bare-metal/asm
asp@gpet05:~/avr-bare-metal/asm$
```

```
asp@gpet05:~$ cd avr-bare-metal/asm/
asp@gpet05:~/avr-bare-metal/asm$ ls
Makefile  led_pin2.s
asp@gpet05:~/avr-bare-metal/asm$
```

#### Linkes Terminal (Editieren):

- Verzeichnisinhalt anzeigen**  
Mit dem Kommando `ls` könnt ihr euch anzeigen lassen, welche Dateien und Unterverzeichnisse sich im aktuellen Verzeichnis befinden. Da es leer sein sollte, wird nichts angezeigt.
- Git-Repository klonen**  
Mit `git clone https://gitlab.uni-ulm.de/mlehn/avr-bare-metal.git` ladet ihr die Code-Basis herunter, die wir für dieses Laborblatt verwenden.
- Verzeichnisinhalt erneut anzeigen**  
Gebt erneut `ls` ein. Es sollte nun das Verzeichnis `avr-bare-metal` angezeigt werden – auf den Laborrechnern in blauer Schrift, weil es ein Verzeichnis ist. Normale Dateien erscheinen in Schwarz.
- Aktuelles Verzeichnis anzeigen**  
Mit `pwd` (= print working directory) zeigt ihr an, in welchem Verzeichnis ihr euch gerade befindet. Das sollte `/home/asp` sein – euer Heimatverzeichnis. Das Verzeichnis `asp` ist ein Unterverzeichnis von `home`, das wiederum direkt im Wurzelverzeichnis `/` liegt.
- Ins Unterverzeichnis wechseln**  
Mit `cd avr-bare-metal` wechselt ihr in das neue Verzeichnis. Kontrolliert mit `pwd`, dass ihr nun in `/home/asp/avr-bare-metal` seid, und lasst euch mit `ls` den Inhalt anzeigen. Ihr solltet ein Unterverzeichnis `asm` sehen.
- Noch ein Verzeichnis tiefer und Datei öffnen**  
Wechselt nun mit `cd asm` in das Unterverzeichnis und prüft wieder mit `pwd` und `ls`, dass ihr jetzt in `/home/asp/avr-bare-metal/asm` seid. Ihr solltet die Datei `led_pin2.s` sehen – das Assemblerprogramm. Öffnet sie im Editor mit `nano led_pin2.s`

**Rechtes Terminal (Kompilieren & Flashen)****A) Ins gleiche Verzeichnis wechseln**

Die Herausforderung: Ihr müsst im rechten Terminal manuell in das gleiche Verzeichnis wechseln wie im linken Terminal. Dazu könnt ihr mit `cd avr-bare-metal/asm` direkt zwei Verzeichnisse nach unten wechseln. Alternativ könnt ihr nacheinander `cd avr-bare-metal` und dann `cd asm` ausführen.

**B) Inhalt der Datei anzeigen**

Mit `cat led_pin2.s` könnt ihr den Inhalt der Datei anzeigen lassen – das ist Assemblercode, eine menschenlesbare Form von Maschinencode. Mit den richtigen Werkzeugen kann man daraus eine sogenannte Hex-Datei erzeugen, die sich auf den Mikrocontroller übertragen lässt.

```

GNU nano 7.2 led_pin2.s
1 sbi 0x0a, 2 ; Pin 2 als Ausgang konfigurieren
2
3 loop: sbi 0x0b, 2 ; Pin 2 auf HIGH setzen
4       nop       ; Nichts machen
5       nop       ; Nichts machen
6       cbi 0x0b, 2 ; Pin 2 auf LOW setzen
7       rjmp  loop ; Zurück zur Marke 'loop'
8
asp@pet05:~$ cd avr-bare-metal/asm/
asp@pet05:~/avr-bare-metal/asm$ ls
Makefile led_pin2.s
asp@pet05:~/avr-bare-metal/asm$ cat led_pin2.s
sbi 0x0a, 2 ; Pin 2 als Ausgang konfigurieren

loop: sbi 0x0b, 2 ; Pin 2 auf HIGH setzen
      nop       ; Nichts machen
      nop       ; Nichts machen
      cbi 0x0b, 2 ; Pin 2 auf LOW setzen
      rjmp  loop ; Zurück zur Marke 'loop'
asp@pet05:~/avr-bare-metal/asm$

```

**C) Makefile verstehen und kompilieren**

Auch das sogenannte Makefile befindet sich im aktuellen Verzeichnis. Es enthält Anweisungen, wie man aus dem Assemblerprogramm automatisch die HEX-Datei erzeugt. Lasst es euch anzeigen mit `cat Makefile`. Startet dann den Übersetzungsvorgang mit `make`

```

asp@pet05:~/avr-bare-metal/asm$ make
avr-as -o led_pin2.o led_pin2.s
avr-ld -mavr5 -o led_pin2.elf led_pin2.o
avr-objcopy -O ihex -R .eeprom led_pin2.elf led_pin2.hex
rm led_pin2.elf led_pin2.o
asp@pet05:~/avr-bare-metal/asm$

```

Wenn alles klappt, wird eine Datei `led_pin2.hex` erzeugt. Diese enthält den Maschinencode, den ihr im letzten Laborblatt manuell erzeugt habt. Schaut euch mit `cat led_pin2.hex` den Inhalt an, der sollte exakt so aussehen:

```

:0C0000000529A5A9A0000000005A98FBCF58
:000000001FF

```

Das ist exakt der Maschinencode von Laborblatt 1, der Pin 2 im 250 ns-Takt auf HIGH und LOW setzt.

**D) Mikrocontroller flashen**

1. Schließt den Mikrocontroller per USB an.
2. Führt `usb_check` aus. Dieses Kommando sorgt dafür, dass das WSL-Gastsystem den USB-Port sieht.
3. Startet den Upload mit `make upload HEX=led_pin2.hex`

**E) Messung mit dem Oszilloskop**

Verwendet ein Oszilloskop und messt das Signal an Pin 2 des Mikrocontrollers. Ihr solltet ein Rechtecksignal sehen mit 250 ns HIGH und 250 ns LOW.

## Befehlsübersicht zum Navigieren durchs Filesystem

Kommando		Erklärung
pwd	print working directory	Gibt wo man sich im Filesystem befindet
ls	list directory content	Liste aller Dateien und Unterverzeichnisse im aktuellen Verzeichnis
cd	change directory	Wechselt immer ins eigene Heimatverzeichnis
cd ..		Wechselt ein Verzeichnis nach oben.
cd foo		Wechselt ins Unterverzeichnis foo (falls es das gibt)
cd foo/bar		Wechselt ins Unter-Unterverzeichnis foo/bar (falls es das gibt)
cat foo.txt	concatenate files and print on the standard output	Falls es die Datei foo.txt im aktuellen Verzeichnis gibt, wird deren Inhalt ausgegeben. Geschickt um schnell zu schauen was in einer Datei drinsteht.

## Schritt 2: Ein zweiter Blick ins Assemblerprogramm

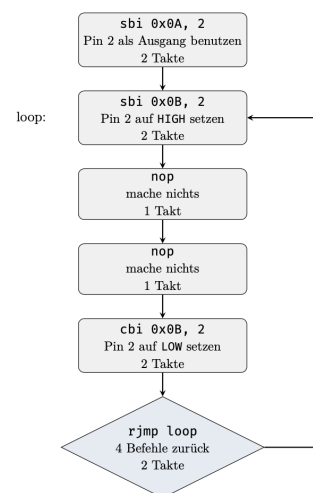
In dieser Aufgabe sollt ihr nachvollziehen, wie das Assemblerprogramm aufgebaut ist – und warum es das Gleiche tut wie das Maschinencode-Programm, das ihr im letzten Labor von Hand geschrieben habt.

```

1          sbi      0x0a, 2      ; 2 Takte
2
3 loop:    sbi      0x0b, 2      ; 2 Takte
4          nop                      ; 1 Takt
5          nop                      ; 1 Takt
6          cbi      0x0b, 2      ; 2 Takte
7          rjmp     loop         ; 2 Takte

```

Jede Zeile enthält einen Maschinenbefehl in symbolischer Form – das ist Assembler. Der Befehl steht ganz links, danach folgen ggf. Argumente. Alles nach `;` ist ein Kommentar, hier steht jeweils, wie viele Takte der Befehl benötigt.



Ablaufdiagramm

## Die Befehle im Einzelnen

Befehl	Bedeutung	Effekt im Programm
sbi	Set Bit in I/O-Register	sbi 0x0a, 2 Pin2 als Ausgang setzen sbi 0x0b, 2 Pin2 auf HIGH
cbi	Clear Bit in I/O-Register	cbi 0x0b, 2 Pin2 auf LOW setzen
nop	No Operation	Keine Wirkung, dient zur zeitlichen Streckung
rjmp loop	Relativer Sprung zur Marke	Springt zur Marke loop: zurück

Die Marken wie `loop:` dienen als Sprungziele. Der Befehl `rjmp loop` sorgt dafür, dass das Programm wieder bei Zeile 3 weitermacht – es entsteht eine Endlosschleife. Das Ablaufdiagramm zeigt den zeitlichen Ablauf grafisch. Es kann euch helfen zu verstehen, wie aus diesem Programm ein Rechtecksignal entsteht.

## Aufgabe

Ändert das Assemblerprogramm so, dass an Pin 2 ein Rechtecksignal mit 500 ns HIGH und 500 ns LOW erzeugt wird. Zur Erinnerung ein Takt benötigt 62.5 ns.

### Schritt 3: Eleganter blinken mit exklusivem Oder

Das folgende Programm sorgt – wie zuvor – dafür, dass an Pin 2 ein symmetrisches Rechtecksignal entsteht: 250 ns HIGH, 250 ns LOW im Wechsel.

Der Unterschied liegt im Code: Diesmal verwenden wir Register und eine neue Instruktion – das exklusive Oder (`eor`) in Zeile 5.

```

1      ldi      r19,    4           ; 1 Takt
2      out     0x0a,    r19        ; 1 Takt
3      ldi      r18,    0           ; 1 Takt
4 loop:
5      eor     r18,    r19          ; 1 Takt
6      out     0x0b,    r18        ; 1 Takt
7      rjmp    loop           ; 2 Takte

```

**Bevor wir analysieren, was der Code genau macht, testen wir zuerst, ob er auch wirklich das tut, was er verspricht.**

#### Aufgaben

- Speichert im Editor `nano` (linkes Terminal) eure aktuelle Datei mit **Strg+O** (oder **Strg+S**) und verlässt den Editor mit **Strg+X**.
- Erstellt eine neue Datei mit `nano led_pin2_xor.s`
- Tippt das folgende Programm ab und speichert es.
- Übersetzt und flasht das neue Programm mit `make upload HEX=led_pin2_xor.hex`
- Prüft mit dem Oszilloskop, ob Pin 2 wie behauptet zwischen HIGH und LOW wechselt.

#### Erläuterung des Programms (und noch eine Aufgabe)

Dieses Programm verwendet Register und eine XOR-Operation (`eor`), um den Zustand von Pin 2 umzuschalten. Register sind kleine Speicherzellen im Prozessor. Der ATmega328P verfügt über 32 Register (r0 bis r31), die jeweils 8 Bit breit sind. Im Gegensatz zum normalen RAM können sie direkt in Rechenoperationen verwendet werden. Neben Addition und Subtraktion ist auch ein exklusives Oder möglich. Im Folgenden ist beschrieben, was die einzelnen Zeilen tun.

**Zeile 1:** `ldi r19, 4`

Schreibt die Zahl 4 ins Register `r19`. Danach gilt `r19 = b00000100` also nur Bit 2 ist gesetzt.

**Zeile 2:** `out 0x0a, r19`

Konfiguriert die Pins 0 bis 7 des Mikrocontrollers als Ein- oder Ausgänge. Da nur Bit 2 in `r19` gesetzt ist, wird nur Pin 2 als Ausgang aktiviert.

**Zeile 3:** `ldi r18, 0`

Initialisiert `r18` mit 0, d. h. alle Bits sind gelöscht.

**Zeile 5:** `eor r18, r19`

Verknüpft `r18` mit `r19` über ein exklusives Oder. Damit wird Bit 2 in `r18` umgeschaltet man sagt auch „getoggelt“: War es vorher 0, wird es zu 1. War es vorher 1, wird es zu 0.

Der Inhalt von `r18` wechselt also zwischen `b00000000` und `b00000100`.

**Zeile 6:** `out 0x0b, r18`

Gibt den Inhalt von `r18` an die Pins 0 bis 7 des Mikrocontrollers aus.

Ist ein Bit in `r18` gesetzt und der entsprechende Pin als Ausgang konfiguriert, dann wird dieser Pin auf HIGH gesetzt. Ist das Bit nicht gesetzt, wird der Pin (sofern er als Ausgang konfiguriert ist) auf LOW gesetzt.

**Aufgabe:** Erklärt uns, warum durch das exklusive Oder in Zeile 5 Bit 2 „getoggelt“ wird.

### Schritt 4: Einfaches Delay mit einem Register

Im Vergleich zum vorherigen Programm sind in diesem Beispiel nur die Zeilen 7 bis 10 neu hinzugekommen. Ihr müsst also **nicht das gesamte Programm neu abtippen**. Wie ihr gleich nachmessen werdet, bewirken diese zusätzlichen Zeilen, dass Pin 2 langsamer zwischen HIGH und LOW wechselt.

```

1      ldi    r19,    4      ; 1 Takt
2      out    0x0a,    r19   ; 1 Takt
3      ldi    r18,    0      ; 1 Takt
4 loop:
5      eor    r18,    r19   ; 1 Takt
6      out    0x0b,    r18   ; 1 Takt
7      ldi    r24,    0x02   ; 1 Takt (Zähler für Countdown setzen)
8 delay:
9      subi    r24,    1      ; 1 Takt
10     brne    delay        ; 2 Takte bei Sprung, 1 Takt sonst
11
12     rjmp    loop          ; 2 Takte

```

#### Aufgaben

- Verlasst den Editor. Führt dann den Befehl `cp led_pin2_xor.s led_pin2_delay.s` aus. Damit habt ihr eine Kopie der Datei `led_pin2_xor.s` mit dem Namen `led_pin2_delay.s`.
- Ergänzt `led_pin2_delay.s` um die Zeilen 7 bis 10 aus dem obigen Listing.
- Die Abbildung rechts zeigt ein Ablaufdiagramm für das Programm. Man erkennt, dass nun zwei ineinander geschachtelte Schleifen vorliegen. Die innere Schleife ist hinzugekommen:
  - In Zeile 7 wird das Register `r24` mit dem Wert `0x02` (also 2) initialisiert.
  - Die Zeilen 8 bis 10 bilden die innere Schleife, in der `r24` heruntergezählt wird, bis es den Wert 0 erreicht hat. Erst dann wird die Schleife verlassen. Im Projektbegleiter wird hergeleitet:

Wird `r24` mit  $n$  initialisiert, dauert ein Durchlauf der äußeren Schleife genau  $3 \cdot n + 4$  Takte.

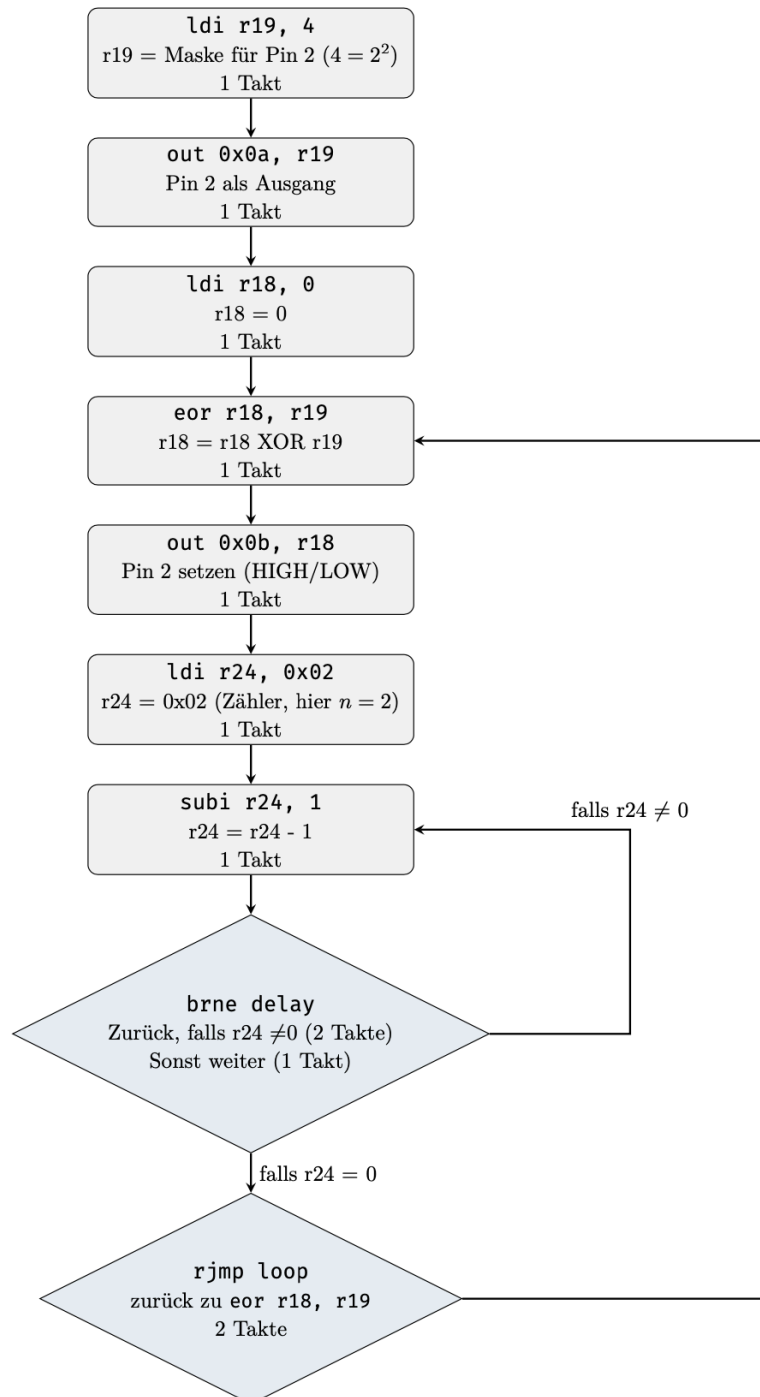
Das bedeutet: Pin 2 bleibt jeweils  $3 \cdot n + 4$  Takte lang auf HIGH und anschließend genauso viele Takte auf LOW, bevor der Zustand wieder umschaltet.

Diese Behauptung überprüft ihr die Aussage experimentell. Füllt dazu die Tabelle auf der nächsten Seite aus, in der ihr für verschiedene Werte von  $n$ :

- In Zeile 7 die Initialisierung von `r24` anpasst,
- das Programm neu auf den Mikrocontroller flasht,
- und anschließend mit dem Oszilloskop überprüft wie lange Pin 2 HIGH und LOW ist.

#### Tabelle für die Messungen (1 Takt dauert 62.5ns)

n	$3 \cdot n + 4$	Exakte Zeitdauer für $3 \cdot n + 4$ Takte	Messung Zeitdauer HIGH	Messung Zeitdauer LOW
1				
2				
3				
4				
255				
0				



## Schritt 5: 32 Bit Delay mit vier Register

Ein einzelnes Register unseres Mikroprozessors kann nur Werte zwischen `0x00` (also 0) und `0xFF` (also 255) speichern. Das reicht nicht aus, um die äußere Schleife so weit zu verzögern, dass man das Blinken einer an Pin2 angeschlossenen LED mit bloßem Auge wahrnehmen kann.

Das folgende Programm verwendet deshalb einen 32-Bit-Zähler, der über die vier Register `r24` bis `r27` realisiert wird. Diese vier Register bilden zusammen einen sogenannten Registerverbund, der wie ein einziges großes Register behandelt wird. Die Anordnung ist dabei:

`r27 : r26 : r25 : r24`

Dieser Verbund speichert einen 32-Bit-Wert im sogenannten Little-Endian-Format – das niederwertigste Byte steht also in `r24`, das höchstwertige in `r27`.

Beispiel: Wenn die Register wie folgt gesetzt sind mit

`r24 = 0x12, r25 = 0x34, r26 = 0x56, r27 = 0x78`

dann ergibt sich als gemeinsamer Wert:

`n = r27:r26:r25:r24 = 0x78563412`

Im untenstehenden Programm sind die Register wie folgt initialisiert:

`r24 = 0x00, r25 = 0x02, r26 = 0x00, r27 = 0x00`

Das entspricht:

`n = 0x00000200 = 512`

Das heißt: Die innere Schleife läuft 512-mal durch.

```

1      ldi      r19,    4          ; für Pin 2 (4 = 2^2)
2      out     0x0a,    r19
3      ldi      r18,    0
4 loop:
5      eor     r18,    r19          ; 1 Takt
6      out     0x0b,    r18          ; 1 Takt
7
8      ldi      r24,    0x00        ; 1 Takt
9      ldi      r25,    0x02        ; 1 Takt
10     ldi      r26,    0x00        ; 1 Takt
11     ldi      r27,    0x00        ; 1 Takt
12 delay:
13     subi     r24,    1          ; 1 Takt
14     sbci     r25,    0          ; 1 Takt
15     sbci     r26,    0          ; 1 Takt
16     sbci     r27,    0          ; 1 Takt
17     brne     delay             ; 2 Takte bei Sprung, 1 Takt sonst
18
19     rjmp     loop              ; 2 Takte

```

## Aufgaben

1. Ihr müsst nicht das gesamte Programm neu abtippen.  
Im Vergleich zu `led_pin2_delay.s` sind nur die Zeilen 9 bis 11 (Initialisierung der Register) und Zeilen 14 bis 16 (Dekrementieren des Registerverbands) neu hinzugekommen.  
→ Passt eure Datei `led_pin2_delay.s` entsprechend an.
2. Im Projektbegleiter wird im Detail erklärt, wie man einen Registerverbund korrekt dekrementiert. Hier sollt ihr die folgende Aussage experimentell überprüfen (gilt nur für  $n > 0$ ):

Wird `r24` mit  $n$  initialisiert, dauert ein Durchlauf der äußeren Schleife genau  $6 \cdot n + 7$  Takte.

Daraus folgt: Pin 2 bleibt jeweils  $6 \cdot n + 7$  Takte auf HIGH und dann genauso lange auf LOW.

Füllt diese Tabelle aus (1 Takt dauert 62.5ns):

n	$6 \cdot n + 7$	Exakte Zeitdauer für $6 \cdot n + 7$ Takte	Messung Zeitdauer HIGH	Messung Zeitdauer LOW
1				
2				

3. **Berechnet nun den Wert  $n$ , bei dem Pin2 im Takt von genau 1Hz blinkt.**

Damit das funktioniert, muss der Pin jeweils 500 ms HIGH und 500 ms LOW sein. Löst dazu die Gleichung:

$$(6 \cdot n + 7) \times 62.5 \text{ ns} = 500 \text{ ms} = 500000000 \text{ ns}$$

- Rechnet den passenden Wert für  $n$  aus.
- Falls  $n$  keine ganze Zahl ist, rundet auf eine ganze Zahl.
- Initialisiert den Zähler entsprechend, flasht das Programm und überprüft mit dem Oszilloskop, ob das Signal der Erwartung entspricht.
- Schließt zusätzlich eine LED mit Vorwiderstand an Pin 2 an und beobachtet das 1-Hz-Blinken – hoffentlich angenehm langsam und gleichmäßig.