1.4 Atmega-Programmierung in ASM/Port-Bit-Befehle

1.4.1 Halb-helle LED

Das Programm soll nun erweitert werden. Die LED1 soll jetzt hell leuchten, LED2 nur halb-hell.

Die halbe Helligkeit kann man in der Digitaltechnik nur bekommen, indem man den Ausgang periodisch ein- und wieder ausschaltet. In dieser Ausgabe wäre es außerdem gut, wenn man die Bits des Ausgangsports *getrennt voneinander* schalten könnte, damit die eine LED immer an sein kann und die andere hin- und hergeschaltet wird.

Abbildung 1 zeigt ein Flussdiagramm zu dieser Aufgabe. Und das folgende Diagramm zeigt

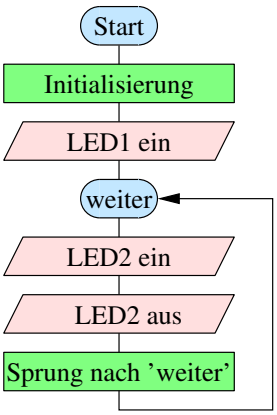

Abbildung 1: Flussdiagramm

einen ersten (unfertigen) Entwurf für den Quellcode des Programms (`led2_halb_v1.asm`):

```
1  .include "/usr/share/avra/m32def.inc"
2          ldi r16, 0b00000110 ; LED2 und LED1 als Ausgang
3          out DDRC, r16
4          ldi r16, 0b11111101 ; LED1 an, LED2 aus
5          out PORTC, r16
6  weiter:
7          ldi r16, 0b11111001 ; beide LEDs an
8          out PORTC, r16
9
10         ldi r16, 0b11111101 ; LED1 an, LED2 aus
11         out PORTC, r16
12         rjmp weiter
```

1.4.2 Port-Bitbefehle

Unschön wird es dann, wenn man jetzt die LED2 an PC2 immer wieder ein- und ausschalten möchte, ohne dass LED1 an PC1 sowie der Rest des Ports C beeinflusst werden. In diesem Beispiel macht es sicher nichts aus, immer wieder PC1 auf eins zu setzen, aber es gibt andere Zusammenhänge, in denen ein unbeabsichtigter Schreibzugriff auf ein Port-Bit große Folgen haben kann.[1] Also: Wie kann man bestimmte Bits in einem Port setzen/löschen, ohne andere zu beeinflussen?

Im folgenden Beispiel sollen Bit 7 von DDRC auf eins und Bit 7 von PORTC auf null gesetzt werden: Bisher haben wir allgemeine Transport-Befehle benutzt:

[1] Bei ungünstigem Hardware-Design kann sogar schon ein Lesezugriff ein Bit verändern! Das kommt aber selten vor.

```
1 ldi r16, 0x80  ; setze r16-Bit Nr. 7 auf eins (Ausgabe)
2                ; und alle anderen auf null(Eingabe)
3 out DDRC, r16  ; gib r16-Inhalt nach Port DDRC
4 ldi r16, 0x80  ; setze r16-Bit Nr. 7 auf eins (Ausgabe)
5                ; und alle anderen auf null(Eingabe)
```

Jetzt gibt es aber auch spezielle Port-Bit-Befehle:

```
1 sbi DDRC, 7  ; setze Bit Nr. 7 von DDRC auf eins (Ausgabe)
2 cbi PORTC, 7 ; setze Bit Nr. 7 von PORTC auf null (low)
```

Diese Port-Bit-Befehle funktionieren leider nur für diejenigen SFR-Register, deren Nummern zwischen 0 und 31 liegen. `sbi` bedeutet *set bit immediate*.[2] `sbi` setzt genau ein Port-Bit auf eins, ohne irgendein anderes Bit zu beeinflussen. `cbi` bedeutet *clear bit immediate*. `cbi` setzt genau ein Port-Bit auf null, ebenfalls, ohne irgendein anderes Bit zu beeinflussen.

Damit das Programm portabel ist, kann die Bit-Nr. noch durch den symbolischen Namen des Bits ersetzt werden:

```
sbi DDRC, DDC7 ; setze Bit DDC7 auf eins (Ausgabe)
cbi PORTC, PC7 ; setze Bit PC7 auf null (low)
```

Diese Ersetzung ist zwar überall üblich, aber nicht notwendig.

1.4.3 Ergebnis mit Port-Bit-Befehlen

Im folgenden Listing sieht man das fertige Programm mit mehrfacher Anwendung der Port-Bit-Befehle:

```
1 .include "/usr/share/avra/m32def.inc"
2         sbi DDRC, PC1  ; LED1 als Ausgang
3         sbi DDRC, PC2  ; LED2 als Ausgang
4         cbi PORTC, PC1 ; LED1 an
5 weiter:
6         cbi PORTC, PC2 ; LED2 an
7         nop
8         sbi PORTC, PC2 ; LED2 aus
9         rjmp weiter
```

In Zeile 8 fällt noch der Befehl `nop` auf. Er steht für *no operation*, und er bewirkt, dass der Controller nichts tut. Er bewirkt nur eine kleine Zeitverzögerung. Mit diesem Kniff kann man dafür sorgen, dass die LED etwa genauso lange ein- wie ausgeschaltet ist: Zwei Befehle (sbi und nop) lang ein- und zwei Befehle (cbi und rjmp) lang ausgeschaltet.[3] Tabelle 1 listet noch einmal alle neuen Befehle auf.

Befehl	a	b
`cbi a, b`	SFR-Nummer (nur 0–31)	Bit-Nr. 0–7
`sbi a, b`	SFR-Nummer (nur 0–31)	Bit-Nr. 0–7
`nop`	—	—

Tabelle 1: Port-Bit-Befehle und NOP

[2] *immediate* ist der Name der Adressierung: Das gemeinte (= adressierte) Bit ist unmittelbar im Befehl angegeben.

[3] Dazu müsste man eigentlich noch nachsehen, wie lange die einzelnen Befehle dauern (und wann genau die Ausgänge geschaltet werden). Angaben dazu finden sich in der Beschreibung des Herstellers.