1.6 Atmega-Programmierung in ASM/Warteschleife

1.6.1 Aufgabe

Der Summer soll zum Summen gebracht werden. Dazu braucht er eine Wechselspannung. Man muss den Summer also abwechselnd ein- und ausschalten (Abbildung 1). Einen ersten Lösungsversuch

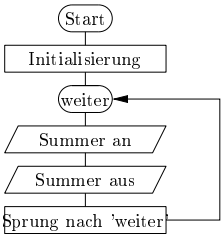

Abbildung 1: Flussdiagramm von `ton_v0.asm`

zeigt `ton_v0.asm`:

```
1 .include "/usr/share/avra/m32def.inc"
2         sbi DDRD, PD7   ; Summer als Ausgang
3 start:
4         sbi PORTD, PD7  ; 2 CPU-Takte
5         cbi PORTD, PD7  ; 2 CPU-Takte
6         rjmp start      ; 2 CPU-Takte
```

Dieses Programm funktioniert jedoch nicht. Beim Nachmessen mit dem Oszilloskop stellt sich heraus, dass die erzeugte Frequenz im Megahertz-Bereich liegt: Ein Durchlauf benötigt hier ca. $0,375\ \mu s$. Dies entspricht einer Frequenz $f = 2,67\ \text{MHz}$[1]. Für eine hörbare Frequenz unter 10 kHz braucht man eine Zykluszeit von mindestes 100 µs.

1.6.2 Befehl zum Warten

Das Flussdiagramm muss also ergänzt werden um zwei große Blöcke mit dem Namen `WARTEN` (Abbildung 2). Auf Programmebene braucht man dazu einen Befehl, der nur Zeit kostet und

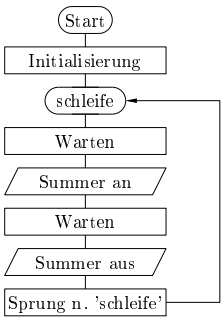

Abbildung 2: Flussdiagramm von `ton_v1.asm`

sonst nichts tut. Er heißt `nop` (*no operation*).[2] Leider wird der Programmcode dadurch sehr lang (`ton_v0.asm`, hier stark verkürzt dargestellt):

[1] Wenn man den internen Oszillator des AVR benutzt, bekommt man eine Frequenz $f = 167\ \text{kHz}$

[2] Für C-Programmierer: Es gibt auch einen Befehl mit dem Namen `sleep`. Er hält den Controller an und ist deshalb hier unbrauchbar.

```
1  .include "/usr/share/avra/m32def.inc"
2          sbi DDRD, PD7   ; Summer als Ausgang
3  start:
4          sbi PORTD, PD7
5          nop
6          nop
7          cbi PORTD, PD7
8          nop
9          nop
10         rjmp start
```

Daher kann man sich überlegen, ob man die vielen `nop`-Befehle durch eine Schleife ersetzen kann:

```
1    for(r16=0; r16<255; ++r16)
2    {
3        nop;
4    }
```

Bei dieser C-Variante zählt ein Zähler hoch. Bei jedem Durchlauf gibt es einen Vergleich, ob ein bestimmter Zähler-Stand erreicht wurde. Falls nicht, springt das Programm zum Anfang.

1.6.3 Befehle für Schleifen

Zunächst braucht man Befehle zum Zählen.

- Zähler hochzählen (inkrementieren): `inc r16` (alle Allzweckregister)
- Zähler herunterzählen (dekrementieren): `dec r16` (alle Allzweckregister)

Für den Vergleich kann man die Befehle `sbrc` und `sbrs` verwenden. Das ist jedoch etwas unpraktisch, weil man mit ihnen nur nach unten springen kann. Ein vollständiges Beispiel findet sich in `ton_v2.asm`. Stattdessen empfehlen sich die Befehle

- `breq` (*branch if equal*)
- `brne` (*branch if not equal*)

Sie bewirken einen Sprung in Abhängigkeit vom Erfolg der vorherigen Operation. Diesen Erfolg (der vorherigen Operation) kann man in einem speziellen Register ablesen, dem Statusregister `SREG` (auch Zustandsregister oder Flags genannt, siehe Tabelle 1). Zunächst sind nur die Bits `Z`

Name									Adresse
	Bit	7	6	5	4	3	2	1	0	
SREG		I	T	H	S	V	N	Z	C	SFR 63
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	Init.	0	0	0	0	0	0	0	0	

Tabelle 1: Statusregister

und `C` wichtig[3].

a) Das `Z`-Bit (*zero*) ist eins, wenn alle Ergebnis-Bits einer Operation null sind.

b) Das `C`-Bit (*carry*) ist eins, wenn das Ergebnis den Wert 255 überschritten (Überlauf) oder 0 unterschritten (Unterlauf) hat.

[3] `I` hat mit Interrupts zu tun, `T` mit nur zwei Spezialbefehlen, H braucht man nur bei BCD-Arithmetik. S, V, N braucht man bei negativen Zahlen im Zweierkomplement.

Damit ist ab jetzt bei vielen Befehlen interessant, welche `SREG`-Bits durch diesen Befehl beeinflusst werden.

- Die Befehle `dec` und `inc` beeinflussen Z ([Schmitt], S. 180/183) sowie S,V und N.
- Der Befehl `breq ziel` springt bei $Z = 1$ zur Marke `ziel` ([Schmitt], S. 174).
- Der Befehl `brne ziel` springt bei $Z = 0$ nach `ziel`.

In der offiziellen AVR-Befehlsliste ist nur aufgelistet, ob eine Beeinflussung stattfindet. Bei [Schmitt] ist zusätzlich angemerkt, wie die Beeinflussung aussieht:

- Minuszeichen: keine Beeinflussung
- 0: Bit wird immer auf null gesetzt
- 1: Bit wird immer auf eins gesetzt
- <>: Bit wird je nach Ergebnis auf null oder eins gesetzt

Befehl	a	b	I	T	H	S	V	N	Z	C
`nop`	—	—	-	-	-	-	-	-	-	-
`inc a`	Univ.-Register	—	-	-	-	<>	<>	<>	<>	-
`dec a`	Univ.-Register	—	-	-	-	<>	<>	<>	<>	-
`breq a`	Sprungmarke	—	-	-	-	-	-	-	-	-
`brne a`	Sprungmarke	—	-	-	-	-	-	-	-	-

Tabelle 2: Befehle für Warteschleifen

1.6.4 Lösungshinweis

Mit diesen Befehlen kann man zunächst ein Flussdiagramm anfertigen (Abbildung 3).

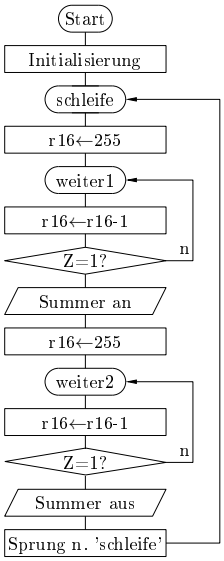

Abbildung 3: Flussdiagramm Zählschleife