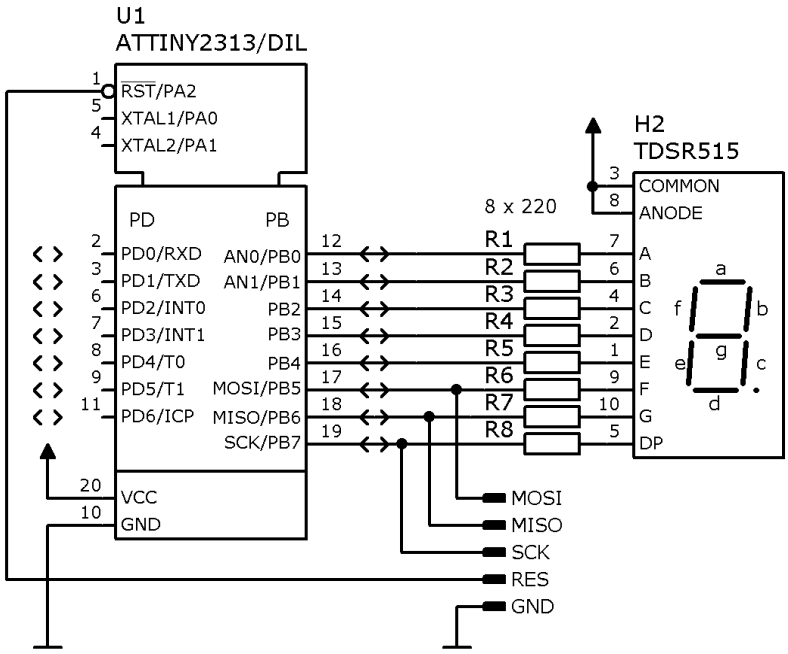
Mikrocontroller 2

Aufgabe 3

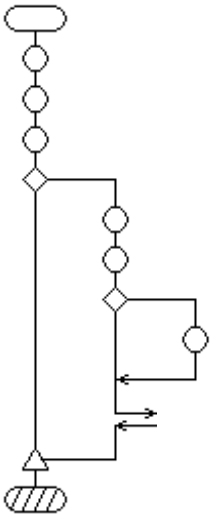
Zählen auf einer Sieben- Segment- Anzeige

Schaltplan Seg7-1



Ablaufdiagramm Seg7-1

Für diesen Ablauf muss das Array Segment[], welches die Bitkombinationen der Zahlen auf der 7-Segmentanzeige enthält, vorher definiert sein. Außerdem wird die Interrupt Service Routine und die Funktion Warte aus Blinker 2.0 gebraucht.



```
Seg7_1
Timer initialiseren
Variablen initialiseren
Anzeigewert = 0
solange wahr
    Port = Segment[Anzeigewert]
    Anzeigewert incrementieren
    wenn Anzeigewert > 9
        Anzeigewert = 0

    Warte
wiederhole
Ende
```

Schreiben Sie das Programm Seg7_1 und lassen Sie es auf der Hardware laufen.

Mikrocontroller 2

Sieben- Segment- Anzeige

Die Anzeige besteht aus 8 Leds, 7 Segmentleds (a...g) und der Punktled.
Damit eine Led leuchtet muss der korrespondierende Anschlusspin (A...G, DP) einen Strom gegen GND abfließen lassen. Um das zu erreichen wird der entsprechende Portpin des Controllers auf low (logisch '0') geschaltet. Ist ein Portpin high (logisch '1') wird die zugehörige Led dunkel sein.
Der Befehl PORTB = 255; schaltet die Anzeige also aus, da alle Portbits auf '1' gehen und damit keine Led ihren Strom bekommt.
Um eine Eins auf der Anzeige zu sehen, sind die Bits für die Segmente b und c auf '0' zu schalten. Die Bitfolge am Port ist demnach 11111001, oder Dezimal 249. Der Punkt ist hier auch ausgeschaltet. Für eine Null sind die Segmente a,b,c,d,e und f eingeschaltet, also '0'. Bitfolge 11000000, Dezimal 192.
Auf diese Weise sind für alle Ziffern Dezimalwerte zu ermitteln. Mit diesen Werten wird das Array Segment[] gefüllt.
Als Erweiterung können auch die Buchstaben A bis F definiert werden, um hexadezimale Zahlen anzuzeigen.

Deklaration eines Arrays mit 10 Einträgen:

```
Type der Inhalte, Name, [ Länge ] ;
char Segment [10] ;
```

Ein Array kann bei der Deklaration gleich initialisiert werden:

```
Type der Inhalte, Name, [] = { Wert0, Wert1, … , Wert9 } ;
```

Dabei kann die Länge entfallen. Der Compiler zählt selber nach.

```
char Segment [] = { 192, 249, … } ;
```

In unserem Fall werden sich die Inhalte des Array während das Programm läuft nicht ändern. Es handelt sich also um ein konstantes Array:

```
const char Segment [] = { 192, 249, … } ;
```

Ein so erzeugtes Array liegt im Hauptspeicher des Mikrocontrollers. Dieses RAM ist allerdings relativ klein (128 Byte). Üblicherweise werden Konstanten deshalb im Programmspeicher des Mikrocontrollers (2 kByte) abgelegt. Um Daten, nicht Programm, in den Programmspeicher zu schreiben braucht der Compiler eine weitere Header- Datei (#include <avr/pgmspace.h>) und ein entsprechendes Schlüsselwort.

```
const char Segment [] PROGMEM = { 192, 249, … } ;
```

Zum lesen von Daten aus dem Programmspeicher wird nun noch eine spezielle Funktion gebraucht. Sie nennt sich pgm_read_byte(). Dieser Funktion wird die Adresse des zu lesenden Bytes mitgegeben. Um die 249 aus dem Programmspeicher zu bekommen lautet der Befehl:

```
pgm_read_byte( &Segment[1]);
```