2.4 Atmega-Peripherie/Konstanten im Flash-EPROM

2.4.1 Speichern eines Datenmusters für ein Lauflicht

Die Leuchtdioden sollen ein durchlaufendes Lichtmuster bekommen. Dieses Lichtmuster soll einfach zu ändern sein, etwa in der Form:

```
1 uint8_t muster[10]={0x81, 0x42, 0x24, 0x18};
```

In C wird dazu `muster` initialisiert durch eine unbenannte Konstante im Konstanten-Speicherbereich. Das Muster wird dazu im Quelltext gespeichert; nach dem Compilieren befindet es sich in der Programmdatei. Auf dem μC muss es im Flash-EPROM oder im EEPROM gespeichert werden. In diesem Fall soll zunächst das Flash-EPROM benutzt werden, weil es einfacher zu handhaben ist.

2.4.2 Benutzung des Flash-EPROMs

Zunächst sagt man das dem Assemblierer, dass man an einer Speicherstelle eine Konstante anlegen möchte.

```
1 .cseg ; default
2 meinekonst:
3         .db 0x81, 0x42, 0x24, 0x18, 0x24, 0x42, 0x81, 0x0
4         .db 'A', "lfa", 0x10, 0x0
5         .db 0b01010101 ; leider nur fuer ein Byte, danach 0x0
6         .dw 1234
```

Nun muss man noch an die Speicherstelle herankommen. Dazu nutzt man aus, dass ihr symbolischer Name bekannt ist (hier: `meinekonst`). Für die Transport des Inhalts aus dem Flash-EPROM in ein Register gibt es einen besonderen Befehl, nämlich `lpm` (*load from programm memory*):

```
1         ld ZH, 2*high(meinekonst)       ; ZH ist r31
2         ld ZL, 2*low(meinekonst)        ; ZL ist r30
3         lpm r16, [Z]                ; Z ist r31:r30 (nur Z erlaubt)
4 ;       lpm                         ; Default: r0=[Z]
```

`lpm` lädt den Inhalt des Bytes, dessen Inhalt in der Registerkombination `r30/r31` steht (hier als `Z`-Register bezeichnet, in das angegebene Universalregister.

Man braucht zwei Register, weil die Flash-EPROM-Adresse größer als 255 sein kann.

Warum muss der Faktor 2 hier eingesetzt werden? Das Flash-EPROM ist 16 Bit breit. Der Befehl `lpm` holt aber nur 8 Bit heraus. Mit dem Bit 0 der Adresse wird unterschieden zwischen den unteren und den oberen 8 Bit eines 16-Bit-Wortes im EPROM.

Ein Beispiel: `meinekonst` liegt an Adresse 100. Dann muss nach `Z` der Wert 200 geladen werden, wenn man das Low-Byte von `meinekonst` holen will. Und es muss der Wert 201 geladen werden, wenn man das High-Byte von `meinekonst` braucht:

```
1         ld ZH, 0          ;
2         ld ZL, 200        ;   Z=200
3         lpm r16, Z        ;
4         adiw Z, 1         ;   ++Z
5         lpm r17, Z        ;
```

Die Zeilen 3 und 4 können auch zusammengefasst werden mit dem Befehl lpm r16, Z+. Dabei wird zuerst das Byte aus dem Flash-EPROM geladen und anschließend das Register `Z` um eins erhöht.

2.4.3 Indirekte Adressierung

Beim `lpm`-Befehl enthält das Register `Z` also *die Adresse* des Speicherwortes (mal zwei gerechnet), das ins Register geladen werden soll. In C-Schreibweise sähe das so aus:

```
1   r16 = *Z;
```

Diese Vorgehensweise wird bei anderen Controllern und CPUs oft *indirekte Adressierung* genannt. Um diese Adressierungsart auszudrücken, schreibt man dort in der Assemblersprache etwa:

```
1       mov r16, [Z]
```

Damit unterscheidet man sie von der sonst üblichen *direkten Adressierung*, bei der einfach der Inhalt eines Registers in den Inhalt des anderen Registers übetragen wird:

```
1       mov r16, r17
```

Mit der indirekten Adressierung kann man so umgehen wie mit Zeigern und Arrays in C: Man kann in einer Schleife das Z-Register schrittweise erhöhen und dann im Schleifenrumpf jeweils ein Byte abholen. Die folgende Sequenz liest aus dem Bereich ab Adresse 100 Zeichen für Zeichen bis zu einem Null-Byte und gibt es aus:

```
1       ld ZH, 0        ;
2       ld ZL, 200      ;  Z=200;
3 loop:
4       lpm r16, Z+     ;  r16=Z++;
5       cpi r16, 0      ;
6       breq ende       ;  while (r16!='\0')
7       call putchar    ;  putchar(r16)
8       rjmp ende
9 ende:
```

2.4.4 Beispielprogramm

Hier ist ein vollständiges Beispiel zum Lesen eines Bytes aus dem Flash-EPROM (`lpm_beispiel.asm`):

```
1  .include "/usr/share/avra/m32def.inc"
2  ;
3  ; gibt ein Muster auf PORTC aus
4  ;
5          ldi r16, 0xff
6          out DDRC, r16           ; PORTC = Ausgabe
7  main:
8          ldi ZL, low(2*muster)   ; ZL laden mit ((2*Adresse)&&0xff)
9          ldi ZH, high(2*muster)  ; ZH laden mit ((2*Adresse)>>16)
10
11         lpm r16, Z              ;  r0=[Z]
12         out PORTC, r16          ;  Ausgabe
13 ende:
14         rjmp ende
15 muster:
16         .db 0x55                ; Muster: jede zweite LED an
```

2.4.5 Wortregister

`r31` und `r30` bilden zusammen das Wortregister `Z`. Die AVR-Controller kennen noch drei weitere Wortregister:

- `r25` und `r24` (unbenannt)
- `r27` und `r26` (Wortregister `X`)
- `r29` und `r28` (Wortregister `Y`)

Für Wortregister gibt es besondere Befehle:

- `adiw` – addiere eine Zahl zum Wortregister, z.B. `adiw ZL, 20`
- `sbiw` – subtrahiere eine Zahl vom Wortregister, z.B. `sbiw YL, 30`
- `movw` – kopiere eine Zahl zwischen Wortregisten, z.B. `movw ZL, YL`