# B2 Arbeiten mit Tabellen

Viele Programme dienen der Verarbeitung größerer Datenmengen. Diese werden meist in Tabellen abgelegt. Um mit Tabellen zu Arbeiten benötigt man die **indirekte Adressierung**. In diesem Kapitel soll die indirekte Adressierung wiederholt und vertieft werden.

Bei der indirekten Adressierung ist die Adresse des Operanden nicht direkt im Befehl enthalten sondern in einem der drei Registerpaare **X**, **Y** oder **Z**, welche als Adresszeiger (Indexregister, Pointer) dienen. Vor jeder indirekten Adressierung muss der Adresszeiger initialisiert werden!

## *Tabellen im Datenspeicher*

"Speichern" (*store*, **st**) und "Laden" (*load*, **ld**) sind die beiden Befehle die der Verarbeitung von Tabellen im Datenspeicher dienen. Besonders praktisch sind auch die Befehle zur indirekte Adressierung mit automatischem Erhöhen bzw. Erniedrigen des Adresszeigers.

Am flexibelsten ist es den Speicher mit Hilfe von Labeln zu organisieren. Dadurch ist man nicht an absolute Adressen gebunden und der Speicher kann optimal genutzt werden.

Im Kapitel zur Adressierung wurde ein Assemblerprogramm erstellt, dessen Aufgabe es war den SRAM-Speicher ab der Adresse **0x0100** mit den Dezimalzahlen 0 bis 255 aufzufüllen (**A404_sram_indirect_1.asm**). Dieses Programm soll jetzt ohne absolute Adresse programmiert werden. Eine mögliche Lösung könnte folgendermaßen aussehen:

```
;------------------------------------------------------------------------------
;       Organisation des Datenspeichers (SRAM)
;------------------------------------------------------------------------------
.DSEG                           ;was ab hier folgt kommt in den SRAM-Speicher
TAB1:  .BYTE  256               ;256 Byte grosse Tabelle im Datensegment

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;       Programmspeicher (FLASH)     Programmstart nach RESET ab Adr. 0x0000
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.CSEG                            ;was ab hier folgt kommt in den FLASH-Speicher
.ORG    0x0000                   ;Programm beginnt an der FLASH-Adresse 0x0000
RESET:  rjmp    INIT             ;springe nach INIT (ueberspringe ISR Vektoren)

;------------------------------------------------------------------------------
;       Initialisierungen und eigene Definitionen
;------------------------------------------------------------------------------
.ORG    INT_VECTORS_SIZE         ;Platz fuer ISR Vektoren lassen
INIT:
.DEF    Cnt1 = r18               ;Register 18 dient als erster Zaehler
        clr     Cnt1             ;Zaehler (Zahl) mit 0 initialisieren
        ldi     XL,LOW(TAB1)     ;Adresszeiger mit Tabellenangfang initialisieren
        ldi     XH,HIGH(TAB1)    ;

;------------------------------------------------------------------------------
;       Hauptprogramm
;------------------------------------------------------------------------------
MAIN:   st      X,Cnt1           ;Speichere den Inhalt des Zaehlers in
                                 ;den Datenspeicher. Die Adresse befindet sich
                                 ;im Doppelregister X
        adiw    XL,1             ;Inkrementiere den 16-Bit-Adresszeiger X
```

```
        inc     Cnt1                ;Inkrementiere den Zaehler (Zahl)
        brne    MAIN                ;Bleib solange in der Schleife bis 256
                                    ;(Ueberlauf, Register wieder auf null)

        ;Ende des Hauptprogramms (falls keine Endlosschleife im Hauptprogramm)
END:    rjmp    END                 ;Endlosschleife

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.EXIT                               ;Ende des Quelltextes
```

✎ **B200** Das obige Programm (und Flussdiagramm!) soll erweitert werden.

a) Zusätzlich zum Erstellen der Zahlentabelle sollen noch die ersten 128 Byte der erstellten Tabelle in eine zweite Tabelle "**Tab2**" kopiert werden. Die Zahlen in der zweiten Tabelle sollen als **ASCII**-Zeichen interpretiert werden (siehe **ASCII**-Tabelle im Anhang). Die Kopie soll dann nur noch die Zeichen "**A-Z**" enthalten. Die restlichen Zeichen sollen mit dem Nullbyte (**0x00**) aufgefüllt werden.
Gib dem Programm den Namen "**B200_sram_indirect_2.asm**".

b) Teste das Programm im Studio 4 mit dem Debugger. Lass das Programm durchlaufen (Autostep ("Alt+F5")) und beobachte das Resultat im Speicherfenster ("View/Memory" oder "Alt+4").

c) Für Fleißige: Ändere das Programm so um, dass zusätzlich "**0-9**", "**a-z**" gültig sind. Nenne das Programm "**B200_sram_indirect_3.asm**" und teste es ebenfalls.

## Tipps zur Programmieraufgabe:

Beim Kopieren von Tabellen wird mit zwei verschiedenen Adresszeigern (z.B. **X** und **Y**) gearbeitet. Zum Filtern von **ASCII**-Zeichen kann man die Befehle "**brsh**" (Verzweige falls gleich oder größer) und "**brlo**" (Verzweige falls kleiner) verwenden.

**brsh k**

**Bedingter relativer Sprung falls (vorzeichenlos) größer oder gleich (*branch if same or higher*).**

| 1 1 1 1 | 0 1 k k | k k k k | k 0 0 0 |
|---|---|---|---|

Der Befehl wird meist gleich nach einem Vergleichsbefehl (**cp**, **cpi**) eingesetzt. **Der Sprung erfolgt falls kein Übertrag (Carry) aufgetreten ist** (**C**-Flag = 0). Beim Vergleich vorzeichenloser Zahlen erfolgt der Sprung wenn der Zieloperand (Rd) ⩾ dem Quelloperand (Rr, K). Der Befehl entspricht dem Befehl brcc.
**Beeinflusste Flags: keine**
**Taktzyklen: 1 (kein Sprung), 2 (Sprung)**

**brlo k**

**Bedingter relativer Sprung falls (vorzeichenlos) kleiner (*branch if lower*).**(k = Adresskonstante)

| 1 1 1 1 | 0 0 k k | k k k k | k 0 0 0 |
|---|---|---|---|

Der Befehl wird meist gleich nach einem Vergleichsbefehl (**cp**, **cpi**) eingesetzt. **Der Sprung erfolgt falls ein Übertrag (Carry) aufgetreten ist** (**C**-Flag = 1). Beim Vergleich vorzeichenloser Zahlen erfolgt der Sprung wenn der Zieloperand (Rd) < dem Quelloperand (Rr, K). Der Befehl entspricht dem Befehl brcs.

**Beeinflusste Flags: keine**
**Taktzyklen: 1 (kein Sprung), 2 (Sprung)**

# *Tabellen im Programmspeicher*

Öfter benötigt man kleine Tabellen mit festen Werten. Da die Programmierung des EEPROM recht aufwändig ist, bietet es sich an diese Tabellen im Programmspeicher mit abzulegen.

"Lade Programmspeicher" (*load program memory*, **lpm**) ermöglicht es ein beliebiges Datenbyte aus dem Programmspeicher (Flash) in ein Arbeitsregister zu laden. Die indirekte Adresse muss sich dazu im **Z**-Pointer (Adresszeiger) befinden (siehe Kapitel "Befehle und Adressierung).

Mit der Direktive "**.ORG**" kann der Beginn der Tabelle festgelegt werden. Ohne diese Anweisung wird die Tabelle gleich hinter dem Programm abgespeichert[4]. Am sichersten ist es die "**.ORG**" Direktive nicht einzusetzen sondern nur mit Labeln (symbolischen Adressen) zu arbeiten. Der Programmieraufwand ist dadurch eventuell größer, jedoch kann dann der Speicher optimal genutzt werden und man läuft nicht Gefahr den Speicher bei großen Projekten mit eingebundenen Bibliotheken mehrfach zu belegen.

Die Direktive "**.DB**" (define Byte) ermöglicht es die Tabelle (Liste mit Bytekonstanten) zu definieren. Da der Programmspeicher in Worten (2 Byte) organisiert ist, ist es nötig die Wortadresse mit dem Faktor zwei zu multiplizieren, da der Adresszeiger nur byteweise adressieren kann. Dies kann einfach im Assembler mit Multiplikationszeichen geschehen.

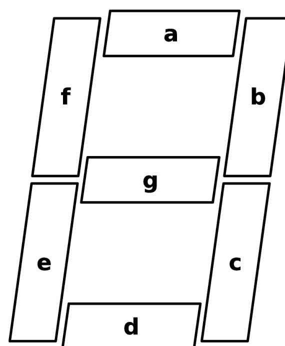
4 Eine Tabelle sollte sich immer zum Schluss der Assemblerdatei befinden, da sonst Teile des Programms oder eines Unterprogramms hinter der Tabelle landen und nicht mehr richtig angesprochen werden!

# *Ansteuerung eines Sieben-Segment-Displays*

## Ansteuerung einer Displaystelle

✎ **B201** Das Beispielprogramm "**A408_flash_indirect_textstring.asm**" kann als Vorlage dienen (Initialisierung des Stapels nicht vergessen!).

a) Das untere Nibble (4 Bit) eines Registers (**r18**) soll in hexadezimaler Schreibweise auf einer Siebensegment_Anzeige ausgegeben werden. Die sieben Segmente (**a-f**) werden durch die unteren 7 Bit des Port D angesteuert. Das hochwertigste Bit aktiviert die Anzeige. Erstelle die Dekodiertabelle!




| Bit: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| Seg: | | g | f | e | d | c | b | a | Byte: |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0xBF |
| 1 | 1 | | | | | | | | |
| 2 | 1 | | | | | | | | |
| 3 | 1 | | | | | | | | |
| 4 | 1 | | | | | | | | |
| 5 | 1 | | | | | | | | |
| 6 | 1 | | | | | | | | |
| 7 | 1 | | | | | | | | |
| 8 | 1 | | | | | | | | |
| 9 | 1 | | | | | | | | |
| A | 1 | | | | | | | | |
| b | 1 | | | | | | | | |
| C | 1 | | | | | | | | |
| d | 1 | | | | | | | | |
| E | 1 | | | | | | | | |
| F | 1 | | | | | | | | |

b) Die Dekodiertabelle soll sich hinter dem Programmcode befinden. Die Addition des aus-maskierten unteren Nibbles des Registers **r18** zur Anfangsadresse der Tabelle dient als Adresszeiger!
Achtung! Es muss eine 16-Bit-Addition durchgeführt werden!
Zeichne das Flussdiagramm. Schreibe den Assemblercode und speichere ihn als "**B201_flash_indirect_numdisplay_1.asm**".
Lade im Initialisierungsteil das Register **r18** mit unterschiedlichen Werten und teste so dein Programm.

c) Erweitere das Programm (und Flussdiagramm) so, dass im Sekunden rythmus abwechselnd das untere und das obere Nibble des Registers auf der Siebensegmentanzeige dargestellt werden. Speichere das erweiterte Programm als "**B201_flash_indirect_numdisplay_2.asm**".

## Tipps zur Programmieraufgabe:

Für eine 16-Bit Addition benötigt man die Befehle "**add**" und "**adc**". Im ersten Schritt werden mit "**add**" die beiden **niederwertigen Register** addiert. Der eventuell auftretende Übertrag (*carry*) wird mit dem "**adc**"-Befehl berücksichtigt. Mit diesem werden die **hochwertigen Register** dann addiert.
Wird ein 8-Bit-Register zu einem 16-Bit-Register addiert (wie in der obigen Aufgabe), so wird bei der Addition der hochwertigen Register Null addiert. Da der "**adc**" Befehl nicht als unmittelbarer Befehl zur Verfügung steht, wird ein gelöschtes Hilfsregister benötigt.

```
        clr     Tmp1            ;Hilfsregister = 0
        add     ZL, Counter     ;Addition der niederwertigen Register
        adc     ZH,Tmp1         ;eventuelles Carry berücksichtigen
```

**adc Rd,Rr**

**Addition des Arbeitsregisters Rd mit dem Arbeitsregister Rr mit Berücksichtigung des Übertrags (*add with carry*).**

| 0 0 0 1 | 1 1 r d | d d d d | r r r r |
|---|---|---|---|

Resultat in Rd. Carry-Flag wird zur Summe hinzuaddiert.
**Beeinflusste Flags: H, S, V, N, Z, C     Taktzyklen: 1**

Mit dem "**swap**"-Befehl können die beiden Nibble (4 Bit-Gruppe, eine hexadezimale Stelle) eines Byte einfach vertauscht werden (Unterpunkt c))!

**swap Rd**

**Niederwertiges und hochwertiges Nibble (4 Bit) eines Registers vertauschen (*swap nibbles*).**

| 1 0 0 1 | 0 1 0 d | d d d d | 0 0 1 0 |
|---|---|---|---|

**Beeinflusste Flags: keine    Taktzyklen: 1**

## Ansteuerung von vier Displaystellen

Es besteht das folgende Unterprogramm "**SR_NUMDISPLAY.asm**", das die Darstellung des Registerinhalts des Doppelregisters **Y** während einer Sekunde auf einem 7-Segment-Display ermöglicht.

```
;****************************************************************************
;*
;*      Titel:  Unterprogramm zur Darstellung eines 16-Bit-Registers auf einer
;*              7-Segmentanzeige (SR_NUMDISPLAY)
;*
;*      Datum:  15/10/08        Version:        0.2
;*      Autor:  WEIGU
;*
;*      Informationen zur Beschaltung:
;*      Prozessor:      ATmega32        Quarzfrequenz:  16MHz
;*      Eingaenge:
;*      Ausgaenge:      Pin 0-6 (a-g) am SegPort
;*                      Pin 0-3 (digit 0-3) am DigPort
;*
;*      !! alle 8 Bit von SegPort werden angesprochen. Bit 7 SegPort ist !!
;*      !! also nicht frei verfuegbar                                    !!
;*
;*      Informationen zur Funktionsweise:
;*
;*      Der Inhalt des Doppelregisters Y wird waehrend einer Sekunde am Display
;*      ausgegeben.
;*      Im Hauptprogramm muessen beide Ports (DigPort, SegPort) initialisiert
;*      werden und die Zeitschleifenbibliothek muss eingebunden sein.
;*      Die Dekodiertabelle befindet sich im Programmspeicher!
;*      Die Hexstelle addiert zur Anfangsadresse dient als Adresszeiger.
;*
;****************************************************************************

;----------------------------------------------------------------------------
;       Unterprogramm zur Darstellung des Doppelreg. Y auf dem 7-Seg.-Display
;----------------------------------------------------------------------------
;Verwendete Register:   r16,r17 ;Reg. 16 u. 17 dienen als Zwischenspeicher
;                       r18     ;Register 18 wird als Zaehler verwendet


NDISP:  push    r16             ;alle verwendeten Register (inkl. SREG) retten
        in      r16,SREG
        push    r16
        push    r17
        push    r18
        push    YL
        push    YH
        push    ZL
        push    ZH

        ;Zaehler initialisieren
        ldi     r18,0xFA         ;Zaehler mit 250 initialisieren (250*4*1ms=1s)

NDISP1: ;erste Stelle anzeigen
        ldi     ZL,LOW(NDISP2*2)  ;Adresszeiger mit der Adresse der Tab.*2
```

```
        ldi     ZH,HIGH(NDISP2*2) ;initialisieren (Worte statt Bytes)
        mov     r16,YL          ;Adresszeiger errechnen
        andi    r16,0x0F        ;Maskieren des untersten Nibble von YL
        clr     r17             ;Zwischenspeicher = 0
        add     ZL,r16          ;Addiere Zaehlerstand zur Basisadresse
        adc     ZH,r17          ;ZH erhoehen falls Carry!
        lpm     r16,Z           ;Speichere das Zeichen der Speicherzeile deren
                                ;Adresse im Z-Pointer steht in das Arbeits-
                                ;register r16.
        sbi     DigPort,0       ;Erste Stelle einschalten
        out     SegPort,r16     ;Zeichen am SegPort ausgeben
        rcall   W1ms            ;Warte 1ms
        cbi     DigPort,0       ;Erste Stelle ausschalten

        ;zweite Stelle anzeigen
        ldi     ZL,LOW(NDISP2*2)  ;Adresszeiger mit der Adresse der Tab.*2
        ldi     ZH,HIGH(NDISP2*2) ;initialisieren (Worte statt Bytes)
        mov     r16,YL          ;Adresszeiger errechnen
        andi    r16,0xF0        ;Maskieren des obersten Nibble von YL
        swap    r16             ;Vertausche beide Nibble
        clr     r17             ;Zwischenspeicher = 0
        add     ZL,r16          ;Addiere Zaehlerstand zur Basisadresse
        adc     ZH,r17          ;ZH erhoehen falls Carry!
        lpm     r16,Z           ;Speichere das Zeichen der Speicherzeile deren
                                ;Adresse im Z-Pointer steht in das Arbeits-
                                ;register r16.
        sbi     DigPort,1       ;Zweite Stelle einschalten
        out     SegPort,r16     ;Zeichen am SegPort ausgeben
        rcall   W1ms            ;Warte 1ms
        cbi     DigPort,1       ;Zweite Stelle ausschalten


        ;dritte Stelle anzeigen
        ldi     ZL,LOW(NDISP2*2)  ;Adresszeiger mit der Adresse der Tab.*2
        ldi     ZH,HIGH(NDISP2*2) ;initialisieren (Worte statt Bytes)
        mov     r16,YH          ;Adresszeiger errechnen
        andi    r16,0x0F        ;Maskieren des untersten Nibble von YH
        clr     r17             ;Zwischenspeicher = 0
        add     ZL,r16          ;Addiere Zaehlerstand zur Basisadresse
        adc     ZH,r17          ;ZH erhoehen falls Carry!
        lpm     r16,Z           ;Speichere das Zeichen der Speicherzeile deren
                                ;Adresse im Z-Pointer steht in das Arbeits-
                                ;register r16.
        sbi     DigPort,2       ;Dritte Stelle einschalten
        out     SegPort,r16     ;Zeichen am SegPort ausgeben
        rcall   W1ms            ;Warte 1ms
        cbi     DigPort,2       ;Dritte Stelle ausschalten

        ;vierte Stelle anzeigen
        ldi     ZL,LOW(NDISP2*2)  ;Adresszeiger mit der Adresse der Tab.*2
        ldi     ZH,HIGH(NDISP2*2) ;initialisieren (Worte statt Bytes)
        mov     r16,YH          ;Adresszeiger errechnen
        andi    r16,0xF0        ;Maskieren des obersten Nibble von YH
        swap    r16             ;Vertausche beide Nibble
        clr     r17             ;Zwischenspeicher = 0
        add     ZL,r16          ;Addiere Zaehlerstand zur Basisadresse
        adc     ZH,r17          ;ZH erhoehen falls Carry!
        lpm     r16,Z           ;Speichere das Zeichen der Speicherzeile deren
                                ;Adresse im Z-Pointer steht in das Arbeits-
                                ;register r16.
        sbi     DigPort,3       ;Vierte Stelle einschalten
        out     SegPort,r16     ;Zeichen am SegPort ausgeben
        rcall   W1ms            ;Warte 1ms
        cbi     DigPort,3       ;Vierte Stelle ausschalten

        ;Zaehler ueberpruefen
        dec     r18             ;Zaehler dekrementieren bis Schleife 250-mal
        brne    NDISP1          ;durchlaufen

        pop     ZH
        pop     ZL
```

```
        pop     YH
        pop     YL
        pop     r18
        pop     r17
        pop     r16
        out     SREG,r16
        pop     r16
        ret                             ;Zurueck ins Hauptprogramm

;-------------------------------------------------------------------------------
;       Tabelle         (7-Segment-Dekodiertabelle (hoechstwertigstes Bit = 0))
;-------------------------------------------------------------------------------
NDISP2:
.DB     0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07     ;Dekodiertabelle fuer
.DB     0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71     ;alle Hex-Zahlen
```

✎ **B202** a) Erstelle ein Flussdiagramm des Unterprogramms!
b) Schreibe ein Assemblerprogramm, das im Sekundentakt alle möglichen Zustände des **Y**-Registers durchläuft und teste es auf mit einem vierstelligen LED-Display (z.B. MICES-Board, siehe Anhang). Das Programm soll als "**B202_numdisplay_4.asm**" abgespeichert werden.
c) Wie lange benötigt das Programm um alle Zustände zu durchlaufen?

## Lauflicht aus einer Tabelle:

✎ **B203** Ein einfaches Lauflicht über 16 LEDs (2 Ports) soll mit Hilfe einer Flash-Tabelle realisiert werden.
Schreibe das Assemblerprogramm und nenne es "**B203_christmas_2.asm**".

# *Blinkmuster aus einer Tabelle:*

Zur Festigung des Gelernten kann zusätzlich folgende Aufgabe gelöst werden:

✎ **B204** Ein Programm soll 32 verschiedene Blinkmuster ausgeben können. Die Blinkmuster befinden sich in einer Tabelle im Programmspeicher. Am Anfang des Programms soll die ganze Tabelle ins SRAM kopiert werden.
Über 5 Schalter soll das Bitmuster aus der SRAM-Tabelle geladen werden. Die Schalterstellung + Basisadresse soll hierbei gleich die Adresse des Bitmusters ergeben.
Ein Unterprogramm holt das Bitmuster aus der Tabelle ab, invertiert es, speichert es in die Tabelle zurück und gibt das Bitmuster an den LEDs aus.
Die Blinkzeit kann über weitere drei Schalter als Vielfaches von 100ms eingestellt werden.

a) Wieso ist dieses Umkopieren notwendig?
b) Erstelle das Flussdiagramm und die Bitmustertabelle!
c) Schreibe das Assemblerprogramm und speichere es als "**B204_christmas_3.asm**".

### Beispiel für eine Bitmustertabelle mit 16 Blinkmustern:

| Adresse | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | Wert |
|---|---|---|---|---|---|---|---|---|---|
| Label+0x00 | ★ | | ★ | | ★ | | ★ | | 0xAA |
| Label+0x01 | ★ | ★ | | | ★ | ★ | | | 0xCC |
| Label+0x02 | ★ | ★ | ★ | ★ | | | | | 0xF0 |
| Label+0x03 | ★ | | | ★ | ★ | | | ★ | 0x99 |
| Label+0x04 | ★ | | | | | | | ★ | 0x81 |
| Label+0x05 | ★ | ★ | | | | | ★ | ★ | 0xC3 |
| Label+0x06 | ★ | ★ | ★ | | | ★ | ★ | ★ | 0xE7 |
| Label+0x07 | ★ | ★ | ★ | ★ | ★ | ★ | ★ | ★ | 0xFF |
| Label+0x08 | | | | ★ | | | | ★ | 0x11 |
| Label+0x09 | | | ★ | | | | ★ | | 0x22 |
| Label+0x0A | | ★ | | | | ★ | | | 0x44 |
| Label+0x0B | ★ | | | | ★ | | | | 0x88 |
| Label+0x0C | | ★ | ★ | ★ | | ★ | ★ | ★ | 0x77 |
| Label+0x0D | ★ | ★ | ★ | | ★ | ★ | ★ | | 0xEE |
| Label+0x0E | | ★ | ★ | | | ★ | ★ | | 0x66 |
| Label+0x0F | ★ | | ★ | ★ | ★ | ★ | | ★ | 0xBD |