

## Aufgabe 1: Ansteuerung der LEDs

Importieren Sie zunächst die Projektdatei `vorlage.zip` in das Code Composer Studio.

Das funktionsfähige Projekt bringt einige der LEDs zum Leuchten. Dafür ist im Wesentlichen der Befehl

**`mov.b #52, &33`**

verantwortlich, der den Wert 52, als Zahlenwert gekennzeichnet durch #, an die Speicheradresse 33, als Adresse gekennzeichnet durch &, ablegt. An dieser Speicheradresse „lauscht“ der Output-Controller, welcher die LEDs schaltet.

Aufgabe ist nun, herauszufinden, nach welchem System die LEDs zu den Zahlenwerten leuchten.

### Aufgabe 1.1

Testen Sie zuerst einige weitere ein- bis zweistellige Zahlenwerte und beobachten Sie, was passiert. Versuchen Sie es insbesondere auch mit dem Wert #0.

### Aufgabe 1.2

Versuchen Sie anschließend gezielt, die LEDs jeweils einzeln zum Leuchten zu bringen. Schreiben Sie sich die jeweils zu den LED gehörenden Werte auf.

### Aufgabe 1.3

Versuchen Sie nun zuerst vorherzusagen, was bei folgenden Werten passiert. Testen Sie danach, ob ihre Vorhersage stimmt:

1. #32
2. #33
3. #3
4. #6
5. #7
6. #128
7. #127
8. #255

### Aufgabe 1.4

Sie sollten nun in der Lage sein folgende LED-Muster zu erzeugen: (X an, 0 aus)

1. 00000X0X
2. 0000X00X
3. X00XX00X
4. XXXX0000
5. XX000000
6. 000X0XXX
7. 00X0X0X0

### Aufgabe 1.5

*Zusatzaufgabe: Erstellen Sie ein Lauflicht, dass Sie durch schnelle Einzelschrittausführung ablaufen lassen können.*

## Aufgabe 2: Speicherzugriffe

Importieren Sie zunächst die Projektdatei `vorlage.zip` in das Code Composer Studio.

### Aufgabe 2.1

Benutzen Sie den Befehl **mov.b** um einige Werte aus dem Hauptspeicher (ab Adresse &512) über ein Akkumulatorregister (R15) an den LED-Controller (&33) zu kopieren, um die Werte aus dem Hauptspeicher anzuzeigen.

### Aufgabe 2.2

Eingaben sind in unseren Mikrocontrollern mittels der vier Taster möglich. Der durch Tastenkombinationen erzeugte Eingabewert kann über Speicheradresse &40 ausgelesen werden. Schreiben Sie ein kurzes Programm, welches den Wert liest und (über den Akkumulator!) an den Ausgabespeicher des LED-Controllers kopiert. Wenn sie dieses Codefragment in eine Schleife legen, sollten Sie die Möglichkeit haben, direkt mit den Tasten die LEDs zu steuern.

### Aufgabe 2.3

*Zusatzaufgabe: Erstellen Sie ein Lauflicht oder andere Animationen auf den LEDs.*

## Beispiellösungen:

### Aufgabe 2.1

```
; Beginn des Programms
    mov.b    &512, R15
    mov.b    R15, &33
    hold
; Ende des Programms
```

### Aufgabe 2.2

```
; Beginn des Programms
begin
    mov.b    &40, R15
    mov.b    R15, &33
    jmp      begin
; Ende des Programms
```

### Aufgabe 3: Arithmetische Operationen

Importieren Sie zunächst die Projektdatei `vorlage.zip` in das Code Composer Studio.

#### Aufgabe 3.1

Der Befehl `inc R15` erhöht den Wert in einem Akkumulatorregister um 1. Schreiben Sie ein Programm, das aus dem Hauptspeicher [&512] einen Wert in den Akkumulator lädt, um eins erhöht und wieder an der ursprünglichen Speicheradresse ablegt.

#### Aufgabe 3.2

Was macht folgendes Programm?

```
; Beginn des Programms
    mov.b    #0,&33
start
    mov.b    &33, R15
    inc      R15
    mov.b    R15, &33
    sleep
    jmp      start
; Ende des Programms
```

- Versuchen Sie zuerst, das Programm zu verstehen und probieren sie es danach aus!

#### Aufgabe 3.3

Der Befehl `'add'` bzw. `'add.b'` erlaubt es, beliebige Zahlen oder auch Speicherinhalte zu einem Akkumulatorregister zu addieren.

- Schreiben Sie ein Programm, dass die Zahlen #34 und #19 addiert. Beobachten Sie den Programmablauf Schritt für Schritt im Register. Wo ist das Ergebnis?
- Ändern sie das Programm ab, damit es die Inhalte der ersten beiden Speicherzellen des Arbeitsspeichers (&512 und &513) addiert und das Ergebnis an der Speicheradresse &514 ablegt. Beobachten Sie den Programmablauf Schritt für Schritt in Speicher und im Register.

#### Aufgabe 3.4

Der Befehl `'sub'` bzw. `'sub.b'` subtrahiert aus dem angegebenen Akkumulatorregister einen Wert oder Speicherinhalt.

- Entwickeln Sie ein Programm, welches von #100 eine beliebige Zahl kleiner 100 abzieht. Beobachten sie das Vorgehen genau im Akkumulatorregister.
- Ziehen Sie jetzt eine Zahl > 100 ab. Wie interpretieren Sie das Ergebnis? Beobachten Sie auch die anderen Register genau bzgl. Veränderungen.

#### Aufgabe 3.5

Versuchen Sie nun, einen einfachen Taschenrechner zu bauen. Dieser soll nacheinander 2 mal Binärwerte über die 4 Knöpfe einlesen und zusammenaddieren. Das Ergebnis soll dann auf den LED dargestellt werden.

Das Programm kann jedoch leider nicht warten, bis Knöpfe gedrückt werden, daher ist es nur durch Einzelschritte (Step Over) möglich auszuführen!

### Beispiellösungen:

### Aufgabe 3.1

```
; Beginn des Programms
```

```
mov.b    &512, R15
inc.b    &512
mov.b    R15, &512
hold
```

```
; Ende des Programms
```

### Aufgabe 3.2

Das angegebene Programm durchläuft auf den LEDs die Binärdarstellung der Zahlen 1 bis 255. Danach beginnt es von vorne, da ein Überlauf  $255 \rightarrow 1$  stattfindet.

### Aufgabe 3.3

```
; Beginn des Programms
```

```
mov.b    #34, R15
add.b    #19
hold
```

```
; Ende des Programms
```

Das Ergebnis steht im Akkumulatorregister (R15).

```
; Beginn des Programms
```

```
mov.b    &512, R15
add.b    &513
mov.b    R15, &514
hold
```

```
; Ende des Programms
```

### Aufgabe 3.4

```
; Beginn des Programms
```

```
mov.b    #100, R15
sub.b    #23, R15
hold
```

```
; Ende des Programms
```

Wird eine Zahl  $x > 100$  abgezogen, so findet ein Überlauf (bzw. ein Unterlauf) statt. Dabei wird auch das Statusregister (SR) entsprechend geändert und der Negative-Flag gesetzt.

### Aufgabe 3.5

```
; Beginn des Programms
```

```
mov.b    &40, R15
add.b    &40, R15
mov.b    R15, &33
```

hold

```
; Ende des Programms
```

## Aufgabe 4: Bedingungen

Importieren Sie zunächst die Projektdatei `vorlage.zip` in das Code Composer Studio.

### Aufgabe 4.1: Zweiseitige Bedingung

Nutzen Sie den Befehl `testBtn1` um zu erkennen, wenn der Knopf 1 gedrückt wird. Ist der Knopf gedrückt, wird der Befehl das Zero-Bit setzen. Konstruieren Sie damit eine zweiseitige Abfrage um bei Knopfdruck alle LEDs einzuschalten. Wird der Knopf losgelassen, sollten alle LEDs wieder ausgehen.

### Aufgabe 4.2: Einseitige Bedingung

Verändern Sie ihr Programm aus 4.1 wie folgt: Zu Programmstart sollen alle LED aus sein. Wird der Knopf 1 gedrückt, sollen alle LEDs an gehen und auch eingeschaltet bleiben, wenn der Knopf losgelassen wird.

Können Sie noch einbauen, dass Knopf 2 die LEDs wieder ausschaltet?

### Aufgabe 4.3 Schleife mit Eingangsbedingung

Was macht folgendes Programm?

```
begin      mov.b    #4, R15
           dec.b    R15
           jz       end
           mov.b    #255, &33
           sleep
           mov.b    #0, &33
           sleep
           jmp      begin
end        mov.b    #0, &33
           hold
```

Versuchen Sie es zuerst zu verstehen. Führen Sie es dann aus um Ihre Vermutung zu prüfen.

Modifizieren Sie dann das Programm wie folgt: Anstatt herunterzuzählen (`dec`) können Sie, ohne die offensichtliche Funktion des Programms zu verändern, auch beginnend mit `#0` hinaufzählen (`inc.b`).

## Kompliziertere Aufgaben

Die folgenden Aufgaben erfordern viele Befehle, dabei müssen Sie oftmals Rechenzwischenschritte für spätere Verwendung zwischenspeichern. Benutzen Sie dazu die Speicherzellen des Hauptspeichers (`&512` ff.). Gehen Sie bei der Programmierung kleinschrittig vor und testen Sie ihre Zwischenlösungen mittels Einzelschrittausführung.

Die Aufgaben sind nicht aufeinander aufgebaut. Suchen Sie sich einfach eine heraus, die Sie lösen möchten!

### Aufgabe 4.4: Multiplikation

Der MSP430 hat keinen eingebauten Befehl für die Multiplikation. Jedoch ist die Multiplikation mit  $n$  nichts anderes als eine  $n$ -Fache Summe über einen Wert (z.B.  $3 \times 5 = 5+5+5$ ). Schreiben sie ein Programm, dass die Werte der Speicheradressen `&512` und `&513` multipliziert. Legen Sie dazu ihre Zwischensumme immer in `&514` ab.

### Aufgabe 4.5: Umständliche Tastatur

Zu Beginn sollen alle LEDs aus sein (Wert 0). Durch Drücken des ersten Knopfes soll der Wert um 1 erhöht werden. Durch Drücken des zweiten Knopfes soll der Wert wieder um 1 gesenkt werden. Der vierte Knopf soll den Speicher wieder zurücksetzen.

*Für Experten: Erweitern Sie ihr Programm zu einem einfachen Taschenrechner: Der dritte Knopf soll den momentan eingegebenen Wert zu den bereits vorher eingegebenen addieren (legen Sie die Zwischensummen in Speicher &512 ab). Der vierte Knopf soll nun nicht zurücksetzen, sondern die bisherige Lösung anzeigen.*

#### **Aufgabe 4.6: Verschachtelte Schleifen**

Lassen Sie die LEDs blinken: Zuerst die erste LED ein mal, dann die zweite LED zwei mal, dann die LED-Kombination für den Wert drei dreimal u.s.w.

## Beispiellösungen:

### Aufgabe 4.1: Zweiseitige Bedingung

; Beginn des Programms

```
start      testBtn1
           jz on
           jmp off
on          mov.b #255,&33
           jmp start
off         mov.b #0, &33
           jmp start
; Ende des Programms
```

### Aufgabe 4.2: Einseitige Bedingung

Ohne Button2:

; Beginn des Programms

```
start      mov.b #0, &33
           testBtn1
           jz on
           jmp start
on          mov.b #255,&33
           hold
; Ende des Programms
```

Mit Button2:

; Beginn des Programms

```
start      mov.b #0, &33
           testBtn1
           jz on
           testBtn2
           jz off
           jmp start
on          mov.b #255,&33
           jmp start
off         mov.b #0, &33
           jmp start
; Ende des Programms
```

### Aufgabe 4.3 Schleife mit Eingangsbedingung

Das angegebene Programm lässt die LEDs 3 mal aufblinken.

```
           mov.b #4, &512
           mov.b #0, R15
begin      inc.b R15
           cmp.b &512
           jz end
           mov.b #255, &33
           sleep
           mov.b #0, &33
           sleep
           jmp begin
end        hold
```

Dieses Dokument wurde von Christoph Krichenbauer (schule@krichenbauer.de) verfasst und darf unter den Bedingungen der folgenden Lizenz weitergegeben und bearbeitet werden:

[Creative Commons Namensnennung - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz](#).