Befehle für Bit- und Logikoperationen

In Assemblerprogrammen sind häufig Manipulationen von einzelnen Bits oder auch von Bytes erforderlich.

Manipulation einzelner Bits im I/O Space

Im I/O-Adressbereich **`0x00`** bis **`0x1F`** (0 bis 31) können einzelne Bits von I/O-Registern direkt manipuliert werden. Hierzu gibt es die beiden Befehle:

- **`CBI`** (Clear Bit in I/O Register)
  ```
  cbi    PORTE,    PORTE0   ; setzt das Bit 0 in PORT E auf 0
  ```
- **`SBI`** (Set Bit in I/O Register)
  ```
  Sbi    PORTE,    PORTE0   ; setzt das Bit 0 in PORT E auf 1
  ```

Müssen einzelne Bits in I/O-Registern mit höheren Adressen manipuliert werden, kann dies nur über den „Umweg“ mittels Allzweck-Register durchgeführt werden. Dazu später mehr.

Testen einzelner Bits im I/O Space

Ebenfalls nur gültig für den I/O Adressbereich 0x00 bis 0x1F (0 bis 31) sind die beiden folgenden Befehle zum Testen eines einzelnen Bits in einem I/O Register:

- SBIC (Skip if Bit in I/O Register is Cleared)
  ```
  sbic   PORTE,    PORTE0   ; überspringe den nächsten Befehl, falls
                            ; Bit 0 in PORT E den Wert 0 hat
  ```
- SBIS (Skip if Bit in I/O Register is Set)
  ```
  sbis   PORTE,    PORTE0   ; überspringe den nächsten Befehl, falls
                            ; Bit 0 in PORT E den Wert 1 hat
  ```

Mit diesen beiden Befehlen lässt sich relativ einfach eine Art **`if...else`**-Struktur aufgrund eines Bit-Wertes implementieren.

Betrachten Sie hierzu das dargestellte Assemblerprogramm.

Abgesehen von einer Zeitverzögerung mittels Warteschleife, toggelt der Wert von Bit 0 in Port E und damit die angeschlossene LED.

```
;=====================================================================
; LED-Blinkprogramm
; LED an PORTE Bit0
;=======Start des Assembler-Programms=================================
.org        0x0000
;=======Initialisierung (Anfang)======================================
INIT:
; Befehle zur Initialisierung
    sbi         DDRE,    PORTE0        ; Bit 0 von Port E als Ausgang
    sbi         PORTE,   PORTE0        ; Port E Bit 0 auf 1 setzen
;=======Initialisierung (Ende)========================================

;===Haupt-Programm in Endlosschleife (Anfang)==========================
MAIN:
    sbis    PORTE,    PORTE0    ; Ist PortE Bit 0 = 1 (LED ist an)?
    rjmp    LED_ON              ; bei false wird der Befehl ausgeführt
                                ; bei true wird der Befehl übersprungen

    cbi     PORTE,  PORTE0      ; Setze PortE Bit 0 = 0 (LED aus)
    rjmp    MAIN

LED_ON:
    sbi     PORTE,    PORTE0      ; Setze PortE Bit 0 = 1 (LED an)
    rjmp    MAIN
;===Haupt-Programm in Endlosschleife (Ende)============================

;=======Ende des Assembler-Programms==================================
```

Logikoperationen

Mikrocontroller verfügen regulär über diverse bitorientierte Logikoperationen wie z.B. ein bitweises UND, ein bitweises ODER und ein bitweises NICHT:

- **`AND`**; bitweises UND von zwei Registern
- **`ANDI`**; bitweises UND von einem Register und einer 8 Bit-Konstanten
- **`OR`**; bitweises ODER von zwei Registern
- **`ORI`**; bitweises ODER von einem Register und einer 8 Bit-Konstanten
- **`COM`**; Einer-Komplement eines Registers (Negation aller Bits)

Aber in der Regel gibt es einige weitere Logikoperationen, so auch beim ATmega 2560:

- **`EOR`**; bitweises Exklusiv-ODER von zwei Registern
- **`NEG`**; Zweier-Komplement eines Registers (Negation aller Bits und Addition von 1)
- **`TST`**; Testet, ob der Wert eines Register 0 oder negativ ist
  (bei negativen Zahlen ist das MSB = 1). Das angegebene Register wird hierfür mit sich selbst UND-verknüpft und ändert seinen Wert daher nicht!
- **`CLR`**; Löscht alle Bits in einem Register, alle Bits haben den Wert 0
- **`SER`**; Setzt alle Bits in einem Register, alle Bits haben den Wert 1

Alle diese Logikoperationen haben Einfluss auf das Prozessorstatus-Register **`SREG`** (**`0x3F`** im I/O-Space bzw. **`0x5F`** im Dataspace).

Das Prozessorstatus-Register besteht aus 8 Bit. Jedes einzelne Bit hat eine bestimmte Bedeutung. Verschiedene Bits werden durch verschiedene Assembler-Befehle beeinflusst. Welche Bits von welchem Befehl wie beeinflusst werden, ist in der Befehlsreferenz anhand der Abkürzungen für die einzelnen Bits ersichtlich.

Aber sehen wir uns erst mal das Prozessorstatus-Register an:

The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

`C` (Bit 0): Dies ist das „Carry-Flag". Das Carry-Flag meldet einen Übertrag bei arithmetischen oder logischen Operationen.

`Z` (Bit 1): Das „Z" steht für das „Zero-Flag". Das Zero-Flag meldet, wenn das Ergebnis einer arithmetischen oder logischen Operation 0 ist.

`N` (Bit 2): Dies ist das „Negative-Flag". Das Negative-Flag meldet, wenn das Ergebnis einer arithmetischen oder logischen Operation negativ ist.

`V` (Bit 3): Dies ist das „Two's Complement Overflow-Flag". Diese Flagge unterstützt die Zweier-Komplement-Arithmetik.

S (Bit 4): Dies ist das „Sign-Flag". Hiermit wird das Vorzeichen eines Zahlenwerts angezeigt. Laut Dokumentation wird das Flag durch eine Exklusiv-ODER Verknüpfung der beiden Flags N und V gebildet: S = N ⊕ V.

H (Bit 5): Dies ist das „Half-Carry-Flag". Bei einigen arithmetischen Operationen zeigt diese Flagge einen Übertrag vom unteren Halb-Byte. Nützlich bei der BCD-Arithmetik.

T (Bit 6): Dieses etwas spezielle Bit wird durch die Befehle **BLD** (**B**it **L**oa**d**) und **BST** (**B**it **St**ore) beeinflusst. Mit diesen Befehlen kann ein bestimmtes Bit in einem Register in das „T-Flag" kopiert oder umgekehrt vom T-Flag an eine bestimmte Bitposition in einem Register kopiert werden. Dieses Bit kann also als eine Art „1-Bit-Speicher" verwendet werden.

I (Bit 7): Dies ist das „Global Interrupt Enable-Flag". Hiermit können zentral alle Interrupts gesperrt oder freigegeben werden.
Nachdem ein Interrupt ausgelöst wurde, wird dieses Bit automatisch durch eine Hardware gelöscht. Damit werden alle nachfolgenden Interrupts erst einmal gesperrt. Durch den Befehl **reti** (Return from Interrupt) wird das Bit dann wieder gesetzt.
Außerdem kann dieses Bit per Assembler-Befehl beeinflusst werden. Dies ermöglichen die beiden Befehle **CLI** (Clear Global Interrupt Flag) und **SEI** (Set Global Interrupt Flag). CLI setzt diese Flagge auf 0 und SEI setzt die Flagge auf 1.

Wofür sind die Bits im Prozessorstatus-Register dann nützlich?

Für Verzweigungen im Programmablauf!

Betrachten Sie die verschiedenen bedingten Sprungbefehle in der Befehlsreferenz. Es gibt für jeden denkbaren Anwendungszweck einen bedingten Sprungbefehl, der aufgrund des Zustands eines Bits im Prozessorstatus-Register arbeitet.

Gezielte Bitmanipulationen

Mit den Logik-Operationen können mehrere Bits in einem Register oder in einem I/O-Register gezielt beeinflusst werden, ohne die anderen Bits in diesem Register anzutasten.

Setzen von einzelnen Bits

Mit dem Befehl **ORI** können ein oder mehrere Bits in einem Register gezielt gesetzt werden, unabhängig von ihrem momentanen Wert. Sehen wir uns dies an einem Beispiel an:

Angenommen, im Register **R16** sollen die beiden Bits **2** und **5** gesetzt werden und die restlichen Bits sollen bleiben, wie sie gerade sind.

Bitnr.	7	6	5	4	3	2	1	0
R16	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**
Maske	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**
ODER.	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**

Dann lässt sich das mittels einer Ver-ODER-ung von **R16** mit einer Maske als konstantem Wert erreichen:

```
ORI    R16,   0b00100100
```

In der Maske werden die gewünschten, zu beeinflussenden Bits auf „1" gesetzt. Die ODER-Funktion setzt dann den Wert an der gewünschten Position auf „1" und die anderen Bits bleiben unangetastet.

Löschen von einzelnen Bits

Mit der UND-Verknüpfung können bestimmte Bits in einem Register auf den Wert „0" gesetzt (gelöscht) werden.

Angenommen, im Register **R16** sollen die Bits **0**, **1**, **5** und **6** gelöscht werden und die restlichen Bits sollen bleiben, wie sie gerade sind.

Bitnr.	7	6	5	4	3	2	1	0
R16	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**
Maske	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**
UND.	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**

Dies lässt sich mittels einer Ver-UND-ung von **R16** mit einer Maske als konstantem Wert erreichen:

```
ANDI    R16,    0b10011100
```

In der Maske werden die gewünschten, zu beeinflussenden Bits auf „0" gesetzt. Die UND-Funktion setzt dann den Wert an der gewünschten Position auf „0" und die anderen Bits bleiben unangetastet.

Toggeln von einzelnen Bits

Ein „Toggeln" lässt sich mit einer EXKLUSIV-ODER-Verknüpfung erreichen. Allerdings gibt es diesen Befehl nicht mit einer Konstanten als Operand, so dass die Konstante zuvor in eines der Allzweckregister geladen werden muss.

Angenommen, im Register **R16** sollen die Bits **3**, **5** und **7** toggeln und die restlichen Bits sollen bleiben, wie sie gerade sind.

Bitnr.	7	6	5	4	3	2	1	0
R16	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**
Maske	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**0**
XOR.	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**

Dies lässt sich mittels einer Ver-XOR-ung von **R16** mit einem weiteren Register, z.B. **R17**, welches die Konstante enthält, erreichen:

```
LDI     R17,    0b10101000
EOR     R16,    R17
```

In der Maske werden die gewünschten, zu „toggelnden" Bits auf „1" gesetzt. Die XOR-Funktion toggelt dann den Wert an der gewünschten Position und die anderen Bits bleiben unangetastet.

Aufgaben

1. Informieren Sie sich in der Befehlsreferenz über die „Bedingten Sprungbefehle".
2. Skizzieren Sie für das Beispielprogramm auf der ersten Seite einen „Programm-Ablauf-Plan" (PAP).
3. Erstellen Sie ein Projekt für das Beispielprogramm. Ergänzen Sie es durch eine Warteschleife, damit das Toggeln der LED sichtbar wird.
4. Ändern Sie das Programm derart, dass Sie den Befehl **sbic** an Stelle von **sbis** verwenden. Testen Sie die Wirkungsweise ohne Aufruf der Warteschleife im Debugger und mit der Warteschleife mittels LED auf dem Breadboard.