Praktikum Mikrorechner 4 (Bitmanipulation und Spezialregister)

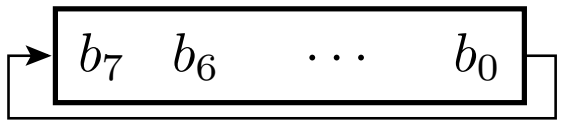
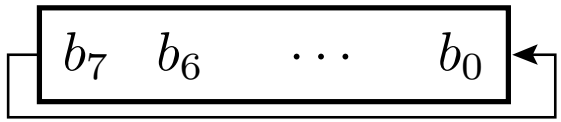
Prof. Kemnitz

Institut für Informatik, Technische Universität Clausthal

November 5, 2014

Logische Operationen

Logische Operationen für Bitvektoren

bitweise Operation	Befehlscode
UND	anl
ODER	orl
EXOR	xrl
Negation	cpl
Rotation rechts (rr)	
Rotation links (rl)	

- Extraktion von Bits und Teilbitvektoren
- Zusammensetzen von Datenobjekten
- Mengenoperationen
- Nachbildung von Operationen (z.B. Multiplikation und Division für Zahlen größer 8 Bit)

Beispiel acc = $0\,0\,\bar{x}_1 x_0 1\,1\,\bar{y}_1 y_0$

```
x   data 60h
y   data 61h
z   data 62h
tmp data 63h
...
mov a, x    ; a: x7 x6 x5 x4 x3 x2 x1 x0
anl a, #03h ; a:  0  0  0  0  0  0 x1 x0
rl a        ; a:  0  0  0  0  0 x1 x0  0
rl a        ; a:  0  0  0  0 x1 x0  0  0
rl a        ; a:  0  0  0 x1 x0  0  0  0
rl a        ; a:  0  0 x1 x0  0  0  0  0
mov tmp, a
mov a, y    ; a: y7 y6 y5 y4 y3 y2 y1 y0
anl a, #03h ; a:  0  0  0  0  0  0 y1 y0
orl a, tmp  ; a:  0  0 x1 x0  0  0 y1 y0
xrl a, #2eh ; a:  0  0 x1 x0  1  1 y1 y0
```

Spezialregister (SFR)

SFR – Special Function Register

Speicherplätze, deren Inhalte spezielle Bedeutungen haben. Lesbar und schreibbar über direkte Adressierung im internen Adressbereich 80h bis 0ffh. Monitorbefehle für den SFR-Zugriff:

```
rb 90=13 ; Wert 13h auf Adresse 90h schreiben
rb 90    ; Wert von Adresse 90h lesen
```

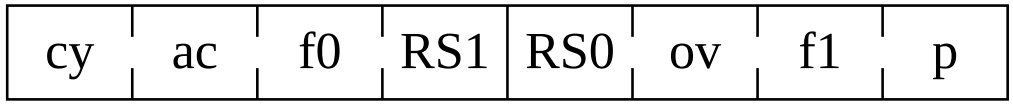
acc: Akku (Adresse e0h)

```
mov a, 20h;   Akku = DS(20h) => e5 20
mov acc, 20h; Akku = DS(20h) => 85 e0 20
; unterschiedliche Befehle, gleiche Funktion
```

B: Spezialregister für die Multiplikation und Division (Adresse f0h)

```
mul ab ; a := Byte0(a*b), b = Byte1(a*b)
div ab ; a := Int(a/b),   b = Rest(a/b)
```

psw: Programmstatuswort (Adresse d0h)



cy	Übertrag für positive ganze Zahlen, ...
ac	Bit für die BCD-Korrektur bei der Umwandlung von Hexadezimal- in Dezimalzahlen
f0, f1	freibenutzbare Flags
rs1, rs0	Auswahl der Registerbank
ov	Überlauf für vorzeichenbehaftete Zahlen
p	Paritätsbit

Registerbankumschaltung

Bank 0 einschalten:

```
anl psw, #0e7h
```

	cy	ac	f0	RS1	RS0	ov	f1	p	
	cy	ac	f0	RS1	RS0	ov	f1	p	
∧	1	1	1	0	0	1	1	1	(0e7h)
	cy	ac	f0	0	0	ov	f1	p	

Bank 1 einschalten:

```
anl psw, #0e7h
orl psw, #008h
```

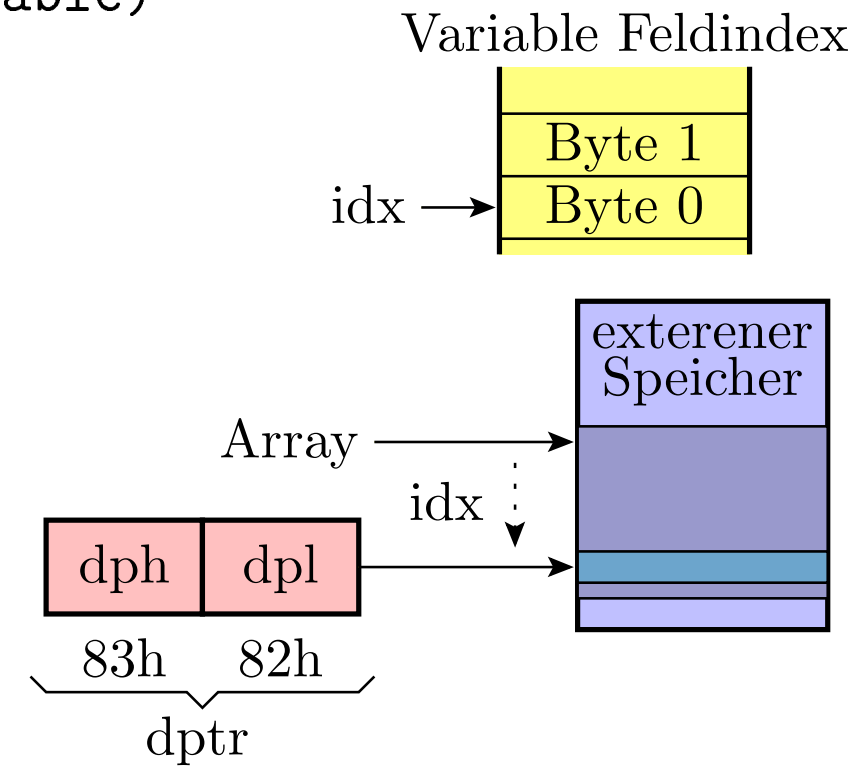
	cy	ac	f0	RS1	RS0	ov	f1	p	
	cy	ac	f0	RS1	RS0	ov	f1	p	
∧	1	1	1	0	0	1	1	1	(0e7h)
∨	0	0	0	0	1	0	0	0	(008h)
	cy	ac	f0	0	1	ov	f1	p	

Datenzeiger dptr (dph, dpl)

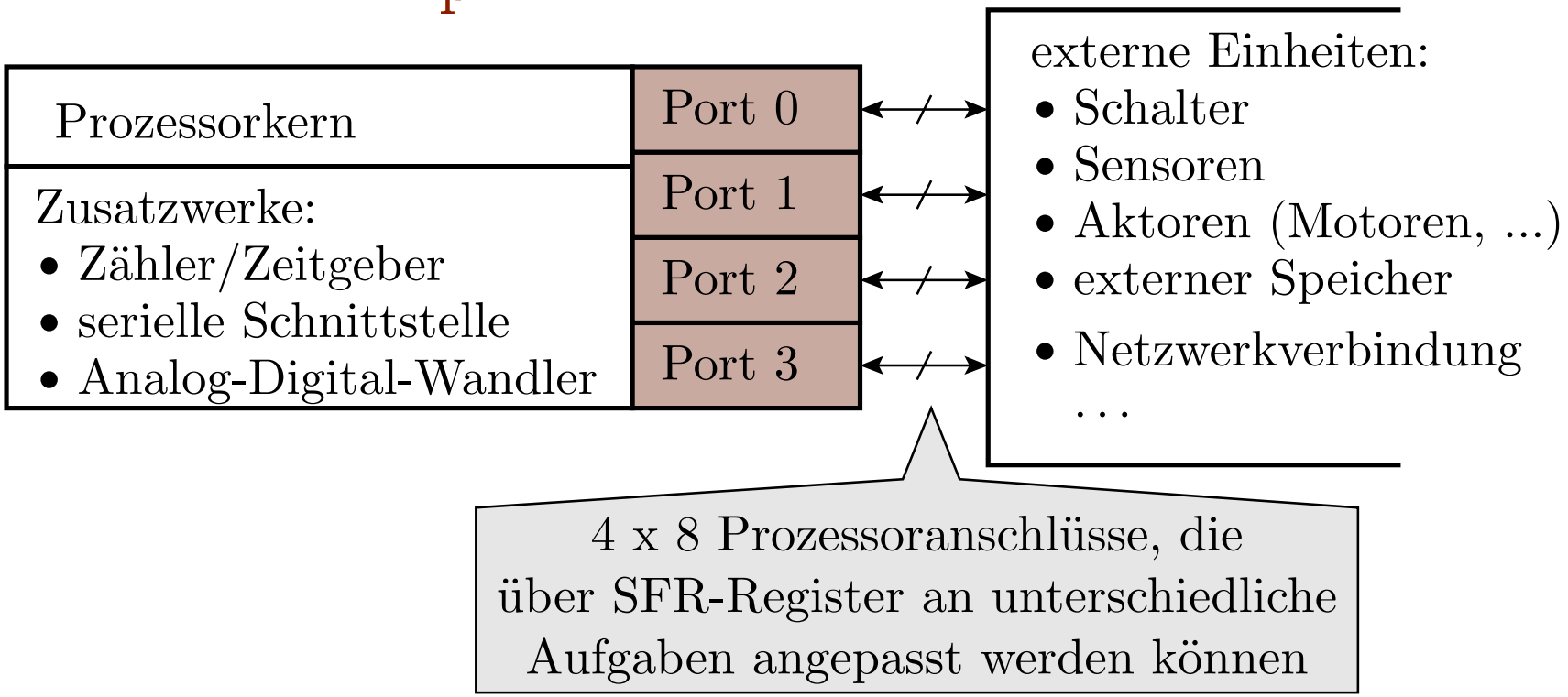
```
mov dptr, #adr16 ; Lade dptr mit einer Konstanten
```

- Berechnete Adressierung, z.B. Akku = ExtSp(Array+idx)

```
ReadArray MACRO Array, idx
; Array: Feldanfang (Konstante)
; idx  : Feldindex (Variable)
; V: psw (carry)
    mov dptr, #Array
    mov a, idx
    add a, dpl
    mov dpl, a
    mov a, idx-1
    addc a, dph
    mov dph, a
    mov a, @dptr
ENDM
```



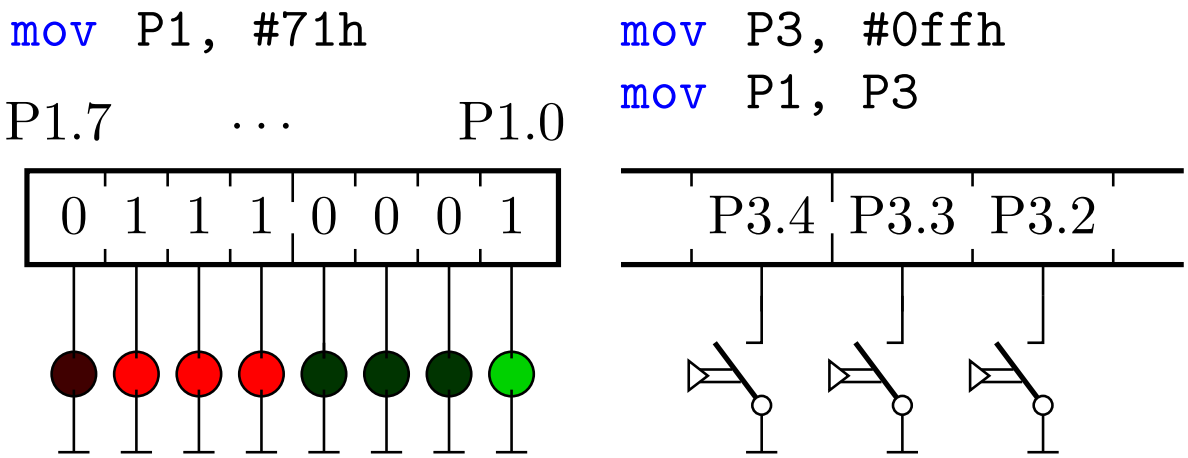
Ports und Peripherie



Ports:

- Spezialregister für die bit- und byteweise Ein- und Ausgabe
- an Port 0, Port 2 und 2 Bits von Port 3 ist der externe Speicher angeschlossen
- 2 weitere Bits von Port 3 dienen als serielle Schnittstelle

Nutzbare Ports



- Der Ausgabewert von Port 1 wird von den Leuchtdioden angezeigt ⇒ wichtige Debugg-Hilfe
- Über Port 3 können auf den Bits 2 bis 4 die Zustände der Schalter eingelesen werden.
- Ein Port-Bit kann nur gelesen werden, wenn auf das Bit vorher eine 1 ausgegeben wurde.

XRAM

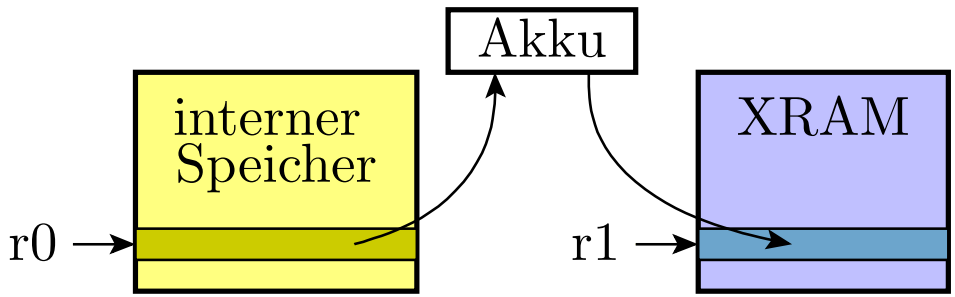
- Ein-/Ausblendung des XRAMs in den/aus dem externen Adressraum durch Setzen/Löschen des Bits xmap (Bit 0 im SFR syscon)

```
syscon data 0b1h; symb. Name vereinbaren

    ...
    xram_ein MACRO
      mov a, syscon     ; Wert von syscon lesen
      orl a, #1         ; Bit 0 setzen
      mov syscon, a     ; veränderten Wert schreiben
    ENDM

     xram_aus MACRO
      mov a, syscon
      anl a, #0feh
      mov syscon, a
    ENDM
```

Makro zum Schreiben des XRAMs

- wie externer Speicher
- als Zeigerregister außer dptr auch r0 und r1 nutzbar

```
wrxx_mac MACRO ziel, quelle
; XRAM(ziel)=DS(quelle)
; V: acc, r0, r1
    xram_ein
    mov r0, quelle
    mov r1, ziel
    mov a, @r0
    movx @r1, ziel
    xram_aus
ENDM
```



Aufgaben

Aufgabe 4.1: Bitmanipulation

Eintwickeln Sie ein Programm, das aus den 1 byte großen Variablen x und y mit Hilfe von Bit-Operationen eine Variable z

$$z_i = \begin{cases} x_i \vee y_i & \text{wenn} \quad i = \{0, 1\} \\ \overline{x_i \vee y_i} & \text{wenn} \quad i = \{4, 5\} \\ 0 & \text{sonst} \end{cases}$$

berechnet. Bereiten Sie vor dem Test 4 Testbeispiele incl. Soll-Ergebnisse vor und testen Sie damit Ihr Programm.

Aufgabe 4.2: Logische Operationen mit Schaltern

Programmieren Sie die logische Gleichung:

$$\mathrm{LD7} = \left(\overline{\mathrm{bt0}} \wedge \overline{\mathrm{bt1}}\right) \vee \left(\overline{\mathrm{bt0}} \wedge \overline{\mathrm{bt2}}\right) \vee \left(\overline{\mathrm{bt1}} \wedge \overline{\mathrm{bt2}}\right)$$

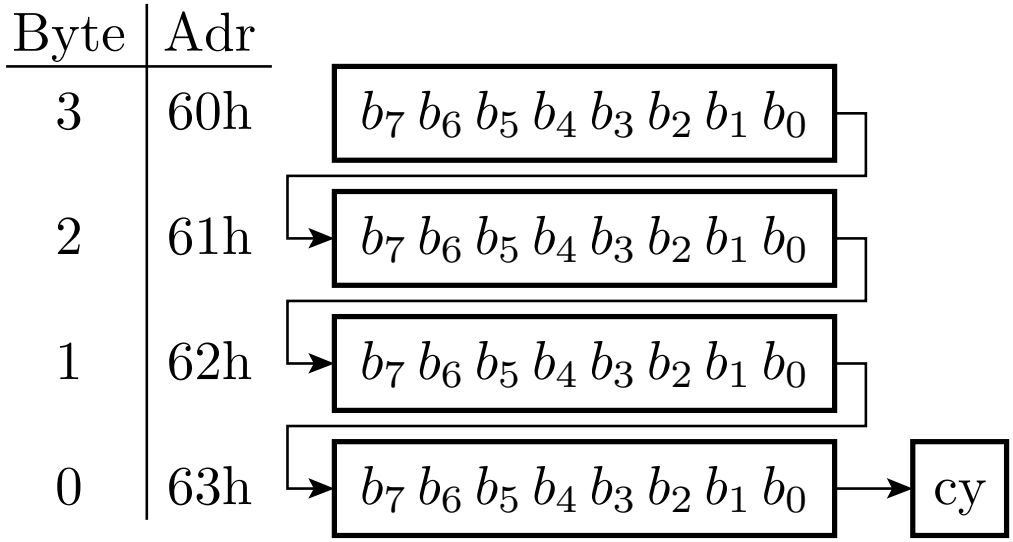
Das Einlesen der Tasterwerte, die eigentliche Berechnung und die Ausgabe an die Leuchtdioden sollen in einer Endlosschleife ausgeführt werden :

```
org 100h
loop:
   Eingabe
   Berechnung
   Ausgabe
   ljmp loop
```

bt0	0	1	0	1	0	1	0	1
bt1	0	0	1	1	0	0	1	1
bt2	0	0	0	0	1	1	1	1
LD7								

Aufgabe 4.3: Halbierung

Entwicklung eines Programms, das eine 4-Byte positive Zahl halbiert:



Laden Sie zum Testen die Variable mit 0ae57213fh und halbieren Sie den Wert viermal hintereinander im Schrittbetrieb.

Aufgabe 4.4: Übertrag und Überlauf

Test des Carry- und des Overflow-Bits. Schreiben Sie ein Programm mit einer 8-Bit-Addition. Bereiten Sie Testbeispiele in der Form der nachfolgenden Tabelle vor, in denen alle vier Variationen des Carry- und des Overflow-Bits vorkommen, und arbeiten Sie die im Schrittbetrieb ab.

	a	b	$a+b$	cy	ov
hexadezimal					
positive ganze Z.*					
2er-Komplement*					

(* Wertangabe dezimal)