Lösungen zu verteilten Aufgaben

Aufgabe 1.1.1

Da nur eine Registerspezifikation in einem Befehl vorkommen kann (Byte 0, Bits 2 ... 0), wird der zu kopierende Wert durch den Akku geschleift :

```
MOV  A,R1      ;Byte in den Akku
MOV  R7,A      ;Byte in R7
```

Aufgabe 1.1.2

```
MOV  A,R4
MOV  R2,A      ;(R4) muss aus dem Weg

MOV  A,R7
MOV  R4,A      ;(R7) kopieren

MOV  A,R2      ;alten (R4) zurückholen
MOV  R7,A
```

Aufgabe 1.1.3

```
MOV  A,R4
XCH  R7,A      ;(R4) im Akku und (R7) vertauschen
MOV  R4,A
```

Aufgabe 1.1.4

Lade jeweils den konstanten dezimalen Wert 27 in den **Akku, R0 und R1**

```
MOV  A,#27
MOV  R0,#1BH          ;1BH = 27
MOV  R0,#0x1B         ;0x1B = 27
MOV  R1,#00011011B    ;00011011B = 27
```

Aufgabe 1.1.5

```
        MOV  A,#55H          ;Befehle benötigen 5 Bytes
        MOV  R5,A
        MOV  R6,A
        MOV  R7,A

;3 *    MOV    Rn,#55H       benötigt 6 Bytes
```

Aufgabe 1.2.1

Aufgabenstellung ist inkonsistent : Der angeführte mögliche Befehl kann die interne Datenspeicherzelle 0E8H nicht ansprechen. An der Adresse 0E8H befindet sich ein SFR.

Korrekt wäre z.B.: Schreiben Sie in die interne Datenspeicheradresse **68H** den Wert 0AAH.

```
        MOV  068H,#0AAH      ;mit # : Immediate-Wert = Adresse
                             ;ohne # : Wert = Inhalt der Speicherzelle
```

Aufgabe 2.1.1

```
        MOV  R0,#20H         ;Register für indirekte Adressierung setzen
        MOV  A,#55H          ;zu speichernder Wert
        MOV  R2,#15H         ;Schleifenzähler
Loop1:                       ;Schleifenanfang
        MOV  @R0,A           ;speichere ein Byte
        INC  R0              ;indirekte Adresse zum nächsten Byte
        DJNZ R2,Loop1        ;springe 15H mal
```

Aufgabe 2.1.2

```
        MOV  A,#00H          ;erster zu speichernder Wert
Loop2:                             ;Schleifenanfang
        MOV  0E8H,A          ;speichere ein Byte
        INC  A               ;nächster zu speichernder Wert
        CJNE A,#21H,Loop2    ;springe bis alle Bytes gespeichert sind
```

Aufgabe 2.1.3

```
        MOV  R0,#20H        ;Register für indirekte Adressierung setzen
        CLR  A              ;erster zu speichernder Wert = 0
Loop3:                              ;Schleifenanfang
        MOV  @R0,A          ;speichere ein Byte
        INC  R0             ;zur nächsten Speicherstelle
        INC  A              ;nächster zu speichernder Wert
        CJNE A,#30H,Loop3   ;springe bis alle Bytes gespeichert sind
```

Aufg. 2.1.4

```
        DSEG    AT   08H    ;Segment für internen Datenspeicher
Feld:   DS      23          ;Reserviert 23 By ab Datenspeicheradresse 08H

        CSEG    at   0      ;Programmsegment
        mov     R0,#Feld+22         ;TO Adresse auf letztes Byte
        mov     R1,#Feld+21         ;FROM Adresse auf vorletztes Byte
        mov     A,@R0       ;Letztes Byte aus dem Weg,
        mov     R2,A        ;rette es in R2
Loop4:
        mov     A,@R1
        mov     @R0,A       ;kopiere ein Byte
        dec     R0          ;zur nächsten TO Adresse
        dec     R1          ;zur nächsten FROM Adresse
        cjne    R0,#Feld,Loop4      ;springe bis 22 Bytes kopiert sind
        mov     A,R2
        mov     @R0,A       ;kopiere 23. Byte in erste Position
```

Aufg. 2.1.5

```
        DSEG    AT   10H   ;Segment für internen Datenspeicher
Feld:   DS      23         ;Reserviert 23 Bytes

i       EQU     12         ;Einfügeposition
newBy   EQU     55H        ;einzufügendes Byte

        CSEG    at   0x00  ;Programmsegment
        mov     R0,#Feld+22       ;TO Adresse auf letztes Byte
        mov     R1,#Feld+21       ;FROM Adresse auf vorletztes Byte
Loop5:
        mov     A,@R1
        mov     @R0,A             ;kopiere Byte
        dec     R0
        dec     R1
        cjne    R0,#Feld+i,Loop5  ;springe bis das i-te Byte kopiert ist
        mov     @R0,#newBy        ;Füge das neue Byte an der i-ten
                                  ;Position ein
```

Aufg. 2.2.1

```
        mov     A,#55H
        mov     R6,#0AAH
        add     A,R6              ;add -> Anfangscarry = 0
        mov     P6,A
```

Aufg. 2.2.2

```
N       EQU     8

        DSEG    AT  20H     ;Segment für internen Datenspeicher
Zahl1:  DS      N           ;Reserviert N By ab 20H
Zahl2:  DS      N           ;Reserviert N By ab 20H + N

        CSEG    at 0
        mov     R0,#Zahl1+N-1       ;letztes Byte der TO Adresse
        mov     R1,#Zahl2+N-1       ;letztes Byte der FROM Adresse
        clr     C           ;Lösche möglichen Carry
Loop6:
        mov     A,@R0
        addc    A,@R1       ;addiere Byte, berücksichtige Carry
        mov     @R0,A       ;schreibe Ergebnis zurück
        dec     R0
        dec     R1
        cjne    R1,#Zahl2-1,Loop6
```

Aufg. 2.2.3

```
Daten   SEGMENT  DATA
        RSEG     Daten
Zahl1:  DS       4
Zahl2:  DS       4
        CSEG     at 0
        mov      R0,#Zahl1+3      ;letztes Byte derTO Adresse
        mov      R1,#Zahl2+3      ;letztes Byte derFROM Adresse
        clr      C                Lösche möglichen Carry
Loop7:
        mov      A,@R0
        addc     A,@R1            addiere Byte
        da       A                ;Decimal Adjust wegen Packed Decimal Format
        mov      @R0,A            ;schreibe Ergebnis zurück
        dec      R0
        dec      R1
        cjne     R1,#Zahl2-1,Loop7
```

Aufg. 3.1.1

```
        anl      A,#0F7H          ;Maske mit Bit3 = 0
```

Aufg. 3.1.2

```
        mov      R3,A             ;rette Originalwert
        rl       A                ;schiebe Akkuinhalt um eine Position nach links
        rl       A                ;4 * rr ist auch richtig
        rl       A
        rl       A                ;siehe auch : swap  A
        anl      A,#0FH           ;lösche 4 höherwertige Bits
        mov      R4,A

        mov      A,R3
        anl      A,#0FH           ;lösche 4 höherwertige Bits
        mov      R3,A
```

Aufg. 3.1.3

```
Daten       SEGMENT   DATA
            RSEG      Daten
Zahl1:      DS        4
Zahl3:      DS        8
            CSEG      at 0
            mov       R1,#Zahl3     ;TO Adresse
            mov       R0,#Zahl1     ;FROM Adresse
Loop8:
            mov       A,@R0         ;hole gepacktes Byte
            swap      A             ;vertausche beide Halbbytes
            anl       A,#0FH
            mov       @R1,A         ;schreibe linke Ziffer zurück
            inc       R1
            mov       A,@R0         ;hole gepacktes Byte noch einmal
            anl       A,#0FH
            mov       @R1,A         ;schreibe rechte Ziffer zurück
            inc       R0
            inc       R1
            cjne      R1,#Zahl3+8,Loop8
```

Aufg. 4.1.1

```
Subrout     SEGMENT   CODE
            RSEG      Subrout     ;Unterprogramm

;INPUT : R0 = Adr. der gepackten Zahl; R1 = Adr. der entpackten Zahl; R2 = Anzahl By
;OUTPUT : Entpackte Zahl von R1 adressiert
;ZERSTÖRT : A, Flags

Unpack:
            push      AR0         ;Rette Registerinhalte
            push      AR1
            push      AR2

            cjne      R2,#0,Loop  ;springe wenn Anzahl > 0
            jmp       Ende        ;zum Ende, wenn Anzahl = 0

Loop:
            mov       A,@R0
            swap      A
            anl       A,#0FH
            mov       @R1,A       schreibe linke Ziffer zurück
            inc       R1
            mov       A,@R0
            anl       A,#0FH
            mov       @R1,A       schreibe rechte Ziffer zurück
            inc       R0
            inc       R1
            djnz      R2,Loop

Ende:
            pop       AR2         ;schreibe Registerinhalte zurück
            pop       AR1
            pop       AR0

            ret                   ;zurück zum Hauptprogramm

Daten       SEGMENT   DATA

            RSEG      Daten

Zahl1:      DS        4
Zahl3:      DS        8

            CSEG      at 0        ;Hauptprogramm

            mov       R0,#Zahl1   ;lade Argumente für das Unterprogramm
            mov       R1,#Zahl3
            mov       R2,#4

            call      Unpack      ;Rufe Unterprogramm auf
```

Aufg. 4.1.2

```
Daten       SEGMENT   DATA
            RSEG      Daten
Bitfeld:    DS        8          ;Bitfeld mit 64 Bits
            CSEG      AT         0
            mov       R0,#Bitfeld ;Adresse des Bitfelds
            mov       R1,#45     ;Bitposition
            mov       A,R1       ;Lade Dividend
            mov       B,#8       ;Lade Divisor
            div       AB         ;A : Byte Offset; B : Bitposition im Byte
            add       A,R0       ;berechne Byte Adr., die das Bit enthält
            mov       R0,A       ;rette diese Adresse
            mov       A,B
            add       A,#1       ;Bitposition + 1 (+ 1 wegen djnz)
            mov       R2,A
            mov       A,#0FEH    ;Anfangsmaske Bit 0 = 0
Loop:
            djnz      R2,Ende
            rl        A          ;Schiebe '0' nach links
            jmp       Loop
Ende:
            anl       A,@R0      ;lösche Bit
            mov       @R0,A      ;schreibe Byte zurück
```

Aufg. 4.1.3

```
Subrout     SEGMENT   CODE
            RSEG      Subrout     ;Unterprogramm

;INPUT : R0 = Adr. des 64 Bit langen Bitfeldes; R1 = Bitposition des zu setzenden Bits
;OUTPUT : Zu setzendes Bit
;ZERSTÖRT : Nichts

SetBit:
            push      ACC         ;rette zu erhaltende Werte
            push      PSW
            push      AR0
            push      AR1
            push      AR2
            push      B

            mov       A,R1        ;Lade Dividend
            mov       B,#8        ;Lade Divisor
            div       AB          ;A : Byte Offset; B : Bitposition im Byte
            add       A,R0        ;berechne Byte Adr., die das Bit enthält
            mov       R0,A        ;rette diese Adresse
            mov       A,B
            add       A,#1        ;Bitposition + 1 (+ 1 wegen djnz)
            mov       R2,A
            mov       A,#01H      ;Anfangsmaske Bit 0 = 1
Loop:
            djnz      R2,Ende
            rl        A           ;Schiebe '1' nach links
            jmp       Loop
Ende:
            orl       A,@R0       ;lösche Bit
            mov       @R0,A       ;schreibe Byte zurück

            pop       B           ;schreibe gerettete Werte zurück
            pop       AR2
            pop       AR1
            pop       AR0
            pop       PSW
            pop       ACC

            ret                   ;zurück zum Hauptprogramm


            CSEG      AT          0     ;Hauptprogramm

            mov       R0,#Bitfeld ;Adresse des Bitfelds
            mov       R1,#45      ;Bitposition

            call      SetBit      ;rufe Unterprogramm auf
```

Aufg. 5.1.1

```
Tast_1      CODE    0xA0C0              ;oder
            EXTRN   CODE    (Tast_1)

            (PUSH   PSW)
            LCALL   Tast_1  ;Lies Tastenwert
            MOV     P4,A    ;Zeige Wert an
            (POP    PSW)
```

Aufg. 5.1.2

```
            LCALL   LCD_init        ;initialisiere LCD
Lösche:
            LCALL   LCD_clr         ;LCD löschen
Schleife:
            LCALL   Tast_1          ;Lies Tastenwert

            CJNE    A,#0x0F,Anzeige
            SJMP    Lösche          ;Springe, wenn Zeichen = 'F'
Anzeige:
            PUSH    ACC             ;Rette Indexwert

            LCALL   goto_xy         ;Setzte Cursor

            MOV     DPTR,#Zeichenkette
            POP     ACC             ;Hole Indexwert zurück
            MOVC    A,@A+DPTR       ;Lies Zeichen
            LCALL   ASC_out         ;Zeige Wert an
            SJMP    Schleife
Zeichenkette:
            DB      '0123456789ABCDE'
```

Aufg. 5.1.3

```
            CSEG    AT      0
Start:
            LJMP    Init            ;Einsprung nach RESET

            ORG     0x0003          ;Einsprung Ext. Int. 0
            LJMP    ISR_Taste

MyCode      SEGMENT CODE
            RSEG    MyCode
ISR_Taste:                          ;Interrupt Service Routine
            LCALL   Tast_1          ;Lies Tastenwert
            MOV     Taste,A         ;Speichere ihn

            MOV     A,Zähler
            LCALL   goto_xy         ;Setzte Cursor

            MOV     A,Zähler
            MOV     DPTR,#Zeichenkette
            MOVC    A,@A+DPTR       ;Lies Zeichen
            LCALL   ASC_out         ;Zeige Wert an
            RETI
```

Aufg. 5.1.3 (Forts.)

```
Init:                                   ;Hauptprogramm
            LCALL   LCD_init            ;initialisiere LCD
            LCALL   LCD_clr             ;LCD löschen
            LCALL   Init_T_int0         ;ISR initialisieren
            MOV     Taste,#0
            CLR     Zähler
Schleife:
            MOV     A,Taste
            CJNE    A,#0x0F,IncZ        ;Zeichen = 'F' ?
            SJMP    Init
IncZ:
            INC     Zähler              ;Inkrementiere Zähler
            MOV     A,Zähler
            CJNE    A,#32,Schleife      ;Überlauf ?
            MOV     Zähler,#0
            SJMP    Schleife

Zeichenkette:
            DB      '0123456789ABCDEF'
            DB      '0123456789ABCDEF'

MyData      SEGMENT DATA                ;Datensegment
            RSEG    MyData
Taste:      DS      1
Zähler:     DS      1
```

Aufg. 5.1.4 Änderungen zu 5.1.3

```
Init:                                   ;Zusatz in der Initialisierung
            MOV     R0,#BelegtFeld
InitSchl:
            MOV     @R0,#0              ;Lösche Markierung
            INC     R0
            CJNE    R0,#BelegtFeld+32,InitSchl
ISR_Taste:                              ;Neue Interrupt Service Routine
            PUSH    AR2
            PUSH    AR0
            LCALL   Tast_1              ;Lies Tastenwert
            MOV     Taste,A             ;Speichere ihn
            MOV     A,Zähler            ;Lies Zähler
            MOV     R2,A
Schleife2:
            MOV     A,#BelegtFeld
            ADD     A,R2
            MOV     R0,A
            CJNE    @R0,#00,FeldFF      ;Zeichen schon angezeigt ?
            MOV     @R0,#0xFF           ;NEIN, markiere als angezeigtes Feld
            MOV     A,R2
            LCALL   goto_xy             ;Setzte Cursor
            MOV     A,R2
            MOV     DPTR,#Zeichenkette
            MOVC    A,@A+DPTR           ;Lies Zeichen
            LCALL   ASC_out             ;Zeige Wert an
            SJMP    Retour
FeldFF:     INC     R2                  ;JA, gehe zum nächsten Zeichen
            CJNE    R2,#32,Schleife2    ;Schleife, wenn nicht Ende erreicht
Retour:
            POP     AR0
            POP     AR2
            RETI
MyData2     SEGMENT DATA                ;2. Datensegment
            RSEG    MyData2
BelegtFeld: DS      32
```