Übungen für die Einführung in die Assemblerprogrammierung mit dem Prozessor c515c

1 Transportbefehle

1.1 Verwendung nur Akku und Register (R0, R1, ... R7)

1.1.1 Kopieren Sie den Wert aus Register1 nach Register7.

Mögliche Befehle:
```
mov   A,Rr        (r= 0 bis 7)
mov   Rr,A
```

Lösung:

1.1.2 Vertauschen Sie die Werte aus Register4 und Register7.

Mögliche Befehle:
```
mov   A,Rr        (r= 0 bis 7)
mov   Rr,A
```

Lösung:

1.1.3 Wie 1.1.2, jedoch zusätzlich Verwendung des Befehls
```
xch   A,Rr
```

Lösung:

1.1.4 Laden eines Registers mit einer Konstanten (dezimal, binär, hexadezimal)

Mögliche Befehle:
```
mov   A,#const8
mov   Rr,#const8  (r= 0 bis 7)
```

Anmerkung:

Mögliche Zahlenformatangaben: dezimal, hexadezimal, binär

```
#23     #23H          #0101110B
```

1.1.5 Füllen Sie die Register R5 bis R7 mit der Konstanten 55H
Verwenden Sie die bisher bekannten Befehle. Gleichzeitig soll so wenig wie möglich Programmspeicher verbraucht werden.

Lösung:

1.2 Arbeiten mit dem internen Datenspeicher

1.2.1 Schreiben Sie in die interne Datenspeicherzelle 0E8H den Wert 0AAH.
Mögliche Befehle: mov dadr,#const8 (dadr = 00 bis 0FFH, Ausnahme E0H)

Lösung:

2 Arithmetische Befehle

2.1 Inkrement / Dekrement

2.1.1 wie 1.1.5, jedoch im internen Datenspeicherbereich 20H bis 34H alle Speicherzellen mit der Konstanten 55H füllen. Verbessern Sie das Programm durch die zusätzlichen Befehle und Verwendung der **registerindirekten** Adressierung
mögliche zusätzliche Befehle

mov	@Ri,A	(i = 0 oder 1)
inc	Rr	
djnz	Rr,relAdr	

Lösung:

2.1.2 Laden Sie die Speicherstelle 0E8H (Port4) nacheinander mit den Werten 00 bis 20H. Verwenden Sie Befehle, die Sie u.a. bei Aufgabe 2.1.1 kennengelernt haben.

Lösung:

2.1.3 Entwerfen Sie ein Programm, welches die Speicherzellen 20H bis 2FH mit den Werten 00 bis 0FH füllt, d.h. der Inhalt der Speicherzelle ist gleich der niederwertigen Hexziffer der Adresse.

Lösung:

2.1.4 Im internen Datenspeicher sei ein Bytefeld mit dem Namen Feld und der Länge 23 Byte definiert. Entwerfen Sie ein Programm, welches dieses Feld um ein Byte im Speicher rotiert, d.h. Byte 0 zur Stelle 1 schieben u.s.w. bis zum Byte N, welches an die Stelle 0 kopiert wird.

Lösung:

2.1.5 Im internen Datenspeicher sei ein Bytefeld mit dem Namen Feld und der Länge 23 Byte definiert. Entwerfen Sie ein Programm, welches an der Stelle i ein Byte einfügt. Alle Bytes ab der Stelle i werden dabei um eine Position nach oben verschoben. Das Byte an der Stelle N geht verloren.

Lösung:

2.2 Addition / Subtraktion

2.2.1 Laden Sie den Akku mit dem Wert 55H und R6 mit 0AAH. Addieren Sie den Inhalt von Akku und R6 und geben Sie das Ergebnis an Port4 aus.
neuer Befehl: add A,Rr

Lösung:

2.2.2 Im internen Datenspeicher stehen an den Adressen Zahl1 und Zahl2 je eine Binärzahl zu je N*8 Bit, d.h. je N Byte. Schreiben Sie ein Assemblerprogramm, welches diese beiden Zahlen addiert. Das Ergebnis soll an die selbe Stelle in den Speicher geschrieben werden, an dem die erste Zahl steht, also an die Zahl1.

In Pascal ausgedrückt lautet die Aufgabenstellung:

Binärzahl1 := Binärzahl1 + Binärzahl2;

neue Befehle:
```
addc    A,Rr
addc    A,@Ri
addc    A,dadr
addc    A,#const8
clr     A
```

Lösung:

2.2.3 Wie vorangegangene Aufgabe, jedoch sollen zwei **8-stellige gepackte BCD-Zahlen** addiert werden (N=4).
neuer Befehl: da A

Lösung:

3 Logische Befehle

3.1 UND

3.1.1 Entwerfen Sie ein Programmstück, welches das Bit D_3 des Akku auf 0 setzt, jedoch die anderen Bits unverändert lässt.
neuer Befehl: anl A,#const8

Lösung:

3.1.2 Im Akku steht eine gepackte BCD-Zahl. Trennen Sie die beiden Ziffern und speichern Sie die niederwertige in R3 und die höherwertige in R4ab.
neue Befehle: rl A
rr A
rlc A
rrc A

Lösung:

3.1.3 Aufbauend auf Aufgabe 2.2.3
Schreiben Sie ein Programm, welches die in Zahl1 abgespeicherte **gepackte** BCD-Zahl nach Zahl3 kopiert und dabei **entpackt**.

Lösung:

4 Unterprogrammtechnik

4.1 CALL

4.1.1 Formulieren Sie die Lösung der Aufgabe 3.1.3 so um, dass sie als Unterprogramm verwendet werden kann. Dabei gilt folgende Vereinbarung:
Ein: R0: Adresse der gepackten BCD-Zahl
R1: Adresse der entpackten BCD-Zahl
R2: Anzahl der gepackten Bytes (= Anzahl der BCD-Ziffern / 2)
Aus: Entpackte Bytes unter der Adresse von R1 im internen Datenspeicher
Zerstört: A, Flags

Neue Befehle:

call	Adresse
push	dadr
pop	dadr
ret	

Schreiben Sie zusätzlich ein kleines Hauptprogramm, in dem die Register mit den Eingabewerten versorgt werden und in dem auch der Stackpointer auf den Anfangswert 07H gesetzt wird

Lösung:

4.1.2 In einem Bitfeld von 64 Bit soll ein beliebiges Bit gelöscht werden. Im Register R1 steht die Nummer des zu löschenden Bits, in R0 steht die Anfangsadresse des Bitfeldes (Byte mit der ersten 8 Bits)

Lösung:

4.1.3 Formulieren Sie die Aufgabe „**Löschen** eine Bits im Bitfeld“ um, in die Aufgabe „**Setzen** eines Bits im Bitfeld“. Schreiben Sie diese als Unterprogramm mit zusätzlichem Hauptprogramm und testen Sie es sorgfältig aus.

Lösung:

5 Kombinierte Übungen

5.1 Übungen mit dem Mikrocontroller-Koffer

5.1.1 Schreiben Sie ein Programm, welches die Tastatur des Mikrocomputerlehrsystemes einliest und den Wert der gedrückten Ziffer an den LEDs des Port 4 (interne Adresse 0E8H) anzeigt. Für die Tastaturabfrage gibt es das folgende Unterprogramm:

Tast_1:	;an der Adresse 0A0C0H	
Ein:	-	
Aus:	ACC	Wert der gedrückten Taste (0 bis F)
zerstört:	Flags	

Lösung:

5.1.2 Definieren Sie im Code-Bereich einen Text von 15 Zeichen (einschließlich Leerzeichen). Schreiben Sie ein Programm, welches zuerst die LCD-Anzeige löscht und dann die Tasten einliest. Bei jedem Tastendruck soll die i-te Stelle des Textes auf dem Display an der i-ten Position angezeigt werden. Der Wert des Tastendruckes ist = i.

Wird die Taste F gedrückt, so soll das Programm von vorne beginnen.

Als Unterprogramme können Sie verwenden:

LCD_init	Display-Hardware initialisieren
Ein:	-
Aus:	-
zerstört:	Flags
LCD_clr	Display löschen
Ein:	-
Aus:	-
zerstört:	Flags
goto_xy	Cursor an eine bestimmte Stelle setzen
Ein:	ACC Cursorposition (00 bis 31)
Aus:	-
zerstört:	ACC, Flags, DPTR
ASC_out	Zeichen auf Display ausgeben (Cursor wird um 1 Position verschoben)
Ein:	ACC ASCII-Zeichen für Ausgabe
Aus:	-
zerstört:	Flags, DPTR
Tast_1	Tastendruck von Tastatur einlesen
Ein:	-
Aus:	ACC Wert der gedrückten Taste (0 bis F)
zerstört:	Flags

Lösung:

5.1.3 Abweichend zur Übung 5.1.2 soll nun zum Hauptprogramm eine Interrupt-Routine verwendet werden, um die Tastatur einzulesen. Die Interrupt-Routine liest den Wert der gedrückten Taste ein und speichert diesen im internen Datenspeicher ab, in der Speicherzelle „Taste“. Außerdem soll nun das Interruptprogramm die Zeichen des Ratetextes zum Display senden.

Die Nummer des zu sendenden Textes wird nun nicht von der Tastatur sondern von einem Zähler geholt, den das Hauptprogramm bearbeitet. Auf diese Art und Weise ist die Stelle des auszugebenden Zeichens ein Zufallsereignis.

Das Hauptprogramm besteht also aus einer Schleife, in der ein Zähler ständig hochgezählt wird, von 00 bis 31 dezimal, d.h. von 00 bis 1FH. Dann geht es wieder bei 00 los. Außerdem soll das Hauptprogramm in der Speicherzelle Taste prüfen, ob die Taste „F“ gedrückt wurde. In diesem Falle soll das Programm neu gestartet werden.

Damit der Interrupt benutzt werden kann, muss zu Beginn des Programmes die Routine „Init_T_int0“ aufgerufen werden. Diese benötigt keine Eingaben und zerstört keine Register.

5.1.4 Die Aufgabe 5.1.3 soll weiter verbessert werden:
Durch die zufällige Auswahl des auszugebenden Zeichens kommt es häufig vor, daß ein Zeichen mehrfach ausgegeben wird, d.h. obwohl es schon auf dem Display angezeigt wird, erfolgt eine erneute Ausgabe.
Dies soll verhindert werden. Mithilfe eines Feldes der Länge 32 im internen Datenspeicher kann man sich merken, welches Zeichen schon ausgegeben wurde. Dieses Feld wird mit 00 auf allen Speicherplätzen initialisiert. Bei jeder Ausgabe eines Zeichens wird der entsprechende Platz des zusätzlichen Feldes mit 0FFH beschrieben. Vor jeder Ausgabe wird geprüft, ob das Zeichen schon ausgegeben wurde.
Wurde das Zeichen schon ausgegeben, so soll in dem Feld das nächste noch nicht ausgegeben Zeichen gesucht werden und dann ersatzweise ausgegeben werden.