CT1 Übungsaufgaben Log- Shift-Operationen und Casting

Aufgabe 1 – Logische Instruktionen

Geben Sie die Assemblerinstruktionen für die folgenden Fälle an.

a) Invertieren des Inhaltes des Registers R1 (Bilden des Einerkomplementes)

```
MVNS   R1,R1
```

b) Verändern Sie den Inhalt des Registers R1 so, dass
- die Bits 3..0 alle eins,
- die Bits 7..4 alle null,
- die Bits 17-16 alle invertiert und
- alle übrigen Bits unverändert sind.

```
MOVS  R0,#0xF
ORRS  R1,R0      ; 3-0 -> 1
MOVS  R0,#0xF0
BICS  R1,R0        ; 7-4 -> 0
MOVS  R0,#0x30
LSLS  R0,#12       ;0x30000 ,shift 4 nibbles = 12bit
EORS  R1,R0      ; 17-16 invertiert
```

Aufgabe 2 – Multiplikation mit einer Konstanten

Schreiben Sie eine Codesequenz in Assembler welche den vorzeichenlosen Inhalt des Registers R7 mit der Dezimalzahl 43 multipliziert und im Register R7 abspeichert.

a) Verwenden Sie die Multiplikationsinstruktion

```
MOVS R0,#43
MULS R7,R0,R7
```

b) Ersetzen Sie die Multiplikationsinstruktion durch Shift- und Additionsbefehle

```
MOVS R7,R0     ;1
LSLS R0,R0,#1 ;2
ADDS R7,R7,R0
LSLS R0,R0,#2 ;8=4*2
ADDS R7,R7,R0
LSLS R0,R0,#2 ;32=8*4
ADDS R7,R7,R0
```

Aufgabe 3 – Reverse Engineering

Die folgenden Assemblersequenzen wurden durch den bcc Compiler aus einem C Programm erzeugt. Schreiben Sie einen möglichen C-Code, welcher diese Assemblersequenz erzeugt.

Die Speicherstellen im Assembler werden den ursprünglichen Variablen im C-Programm wie folgt zugeordnet:

```
ux    DCD   ?     □□    uint32_t ux
uy    DCD   ?     □□    uint32_t uy
uz    DCD   ?     □□    uint32_t uz
```

	Assembler	**C**

Beispiel	`LDR R0,=ux`
	`LDR R1,[R0]`
	`ADDS R1,#1`
	`STR R1,[R0]`
		`ux++;`

a)	`LDR R0,=ux`
	`LDR R1,[R0]`
	`LSLS R1,R1,#1`
	
	`LDR R2,=uy`
	`LDR R3,[R2]`
	`ADDS R3,R1`
	`LSLS R3,R3,#3`
	`STR R3,[R2]`
		`Lsg:`
		`uy = 8 * (2 * ux + uy);`
		`oder`
		`uy = ((ux << 1) + uy) << 3;`

b)	`LDR R0,=ux`
	`LDR R1,[R0]`
	`LDR R2,=uy`
	`LDR R3,[R2]`
	`LDR R4,=uz`
	`LDR R5,[R4]`
	`LSRS R1,R1,#3`
	`LSLS R3,R3,#4`
	`ORRS R1,R3`
	`MVNS R1,R1`
	`ANDS R1,R5`
	`STR R1,[R4]`
		`Lsg:`
		`uz = ~( (ux >> 3) | (uy << 4)) & uz;`
		`oder`
		`uz = ~( (ux / 8) | (uy * 16)) & uz;`

Aufgabe 4 – Explicit Casting in C

a) Gegeben ist der folgende C Code:

```
uint8_t ux = 100;
int8_t  sx = (int8_t)ux;
```

Als welche Dezimalzahl wird der Inhalt der Variable sx nach dem Cast interpretiert?

100d --> ux hat Speicherinhalt 0x64 oder Binär 0110‘0100
sx hat denselben Speicherinhalt, wird aber als signed interpretiert. Da das höchstwertigste Bit nicht gesetzt ist, hat die Interpretation aber in diesem Fall keinen Einfluss.
sx wird ebenfalls als 100d interpretiert

b) Gegeben ist der folgende C Code:

```
int8_t  sx  = -10;
uint8_t ux = (uint_8)sx;
```

Als welche Dezimalzahl wird der Inhalt der Variable ux nach dem Cast interpretiert?

-10d --> sx hat Speicherinhalt 0xF6 oder Binär 1111‘0110
(-128 + 64 + 32 + 16 + 4 + 2)
ux hat denselben Speicherinhalt, wird aber als unsigned interpretiert. So ergibt sich 128 + 64 + 32 + 16 + 4 + 2 = 246d