Subtraktion von binären Zahlen
Drei Schritte zu Subtraktion:
Das Einerkomplement Das Zweierkomplement Die Subtraktion von Dualzahlen
Einerkomplement
Was ist das Komplement von Dualzahlen? Man bildet das sogenannte Einerkomplement, indem man jede Zahl durch ihr Gegenteil ersetzt, also die 0 durch die 1 und die 1 durch die 0.

01011010 wird zu 10100101
11101101 wird zu 00010010

Das Zweierkomplement Das Zweierkomplement entspricht dem Einerkomplement, nur wird zusätzlich noch 00000001 addiert.
01011010 wird im Einerkomplement zu 10100101 im Zweierkomplement zu 10100110 11101101 wird im Einerkomplement zu 00010010 im Zweierkomplement zu 00010011

Konvertierung von Festkommazahlen Dez zu Bin 10,2

Vorkommastelle 10 = 1010
Nachkommastelle
0,2 * 2 = 0,4 + 0 *MSB*
0,4 * 2 = 0,8 + 0
0,8 * 2 = 0,6 + 1
0,6 * 2 = *0,2* + 1 *LSB*

Sobald es sich wiederholt kann aufgehört werden.
0, 2 = 0,0011
10,2 $\ominus$ 1010,00110011 $\approx$ 0,19921875
$\Longrightarrow Eine Abweichung von -0,00078125$

Konvertierung von Fließkommazahlen Dez zu Bin $18{,}4_{10}$

$18_{10} \ominus 10010_2 0,4_{10} \ominus 0,011_2$

mov vs. ldr

ldr Funktion	mov
r1, [r2] speichere Wert von r2 in r1	r1, r2
r1, =255 speichere 255 in r1	r1, 255
Bewegt Speicher/Register -	Bewegt Register
32-Bit -	8-Bit

Die Subtraktion von Dualzahlen Der Satz lautet: Die Subtraktion von 2 Zahlen erfolgt durch die Addition des Zweierkomplementes. Als konkretes Beispiel nehmen wir dazu die Rechnung 14-9=5.
9 ist im Dualsystem 00001001. Das Einerkomplement zu 00001001 ist 11110110. Das Zweierkomplement 11110111. Dies addieren wir nun zu 14 also 00001110.

```
 00001110
+11110111
 ========
 00000101
```

Auch hier wäre die richtige Zahl eigentlich 00000101 Übertrag 1, da wir den Übertrag jedoch nicht speichern können, bleiben wir bei 00000101 was ja der Dezimalzahl 5 entspricht.

Little-/Bigendian

Assemblerbefehle

```
AREA MyCommonBlock, COMMON, ALIGN = 10 ; Read-Write-Data
MyCommonBlock bezeichnet die Anfangsadresse des Speicherblocks
COMMON: vom Linker mit Nullen initialisierter Speicherbereich
Alignment mit 2^{10} erzeugt eine Blockgrenze bzw.   anfang   mit n * 1024

mov r0, #0x21
Lade #0x21 in Register R0:
R0
00000021
```

Angabe negativer Konstanten mov r1, -10

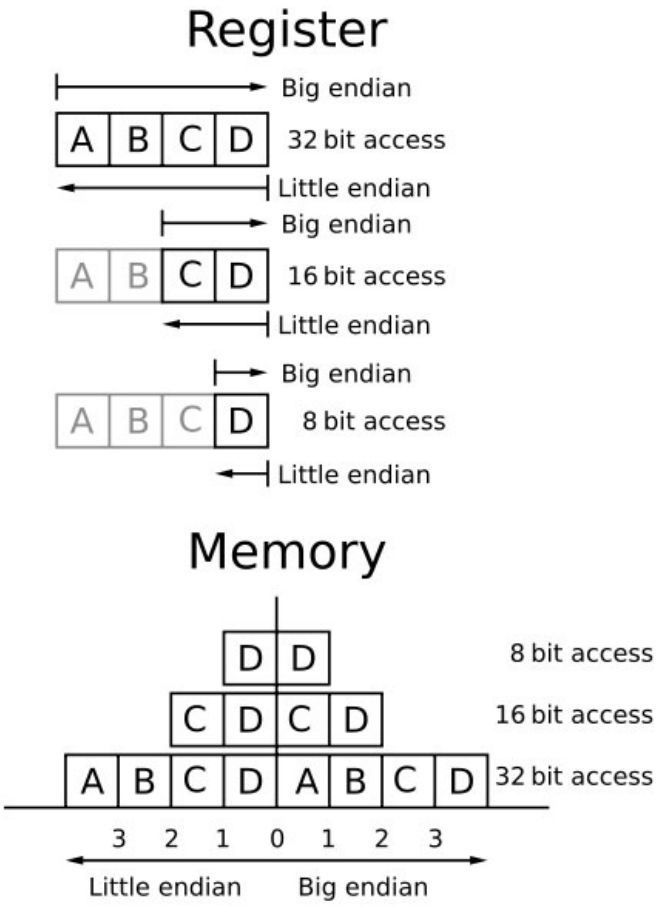

Figure 1: A simple caption

```
DCB  8 Bit
DCW 16 Bit
DCD 32 Bit
```

Lösungen für Tests

1 Lösung für Test 1:

2 Lösung für Test 2:

2.1

Aufgabe:
Die **vorzeichenlose** Zahl in r0 soll durch 4 geteilt werden. Das Ergebnis soll in r1 stehen.
Geben Sie den Befehl an:

```
MOV r0,r1,ASR #2
```

Erklärung: Eine Verschiebeoperation nach **links** um 1 Bit entspricht der **Multiplikation** mit 2
und eine Verschiebeoperation nach **rechts** um 1 Bit entspricht der **Division** mit 2.
Warum # 2 statt 4? # 1 $\rightarrow$ x $\div$ 2; # 2 $\rightarrow$ x $\div$ 2 $\div$ 2 $\rightarrow$ x $\div$ 4

Nachschlagen: Kapitel 8.5.5

2.2

Aufgabe:
Das Datenfeld Var1 beginne bei Adresse 0x2000. Welcher Wert (hex.) vsteht nach Ausführung des Befehls in r0?

```
Var1    DCB     10, 'A', 0xA, '1'

                ldr r0, =Var1
```

Lösung r0 = 0x2000

Erklärung: Lade die Adresse von Var1 in r0.

Nachschlagen: Kapitel 7.5.3

2.3

Das Datenfeld Tab beginne bei Adresse 0x2000. Geben Sie die Speicherinhalte (hex.) von Adresse 0x2000 - 0x2003 an?

```
Var1    DCB     0x10, 'A', 10, '1'
```

Lösung 0x2000: 41 0A 31 10

Erklärung: 0x10 → Hexadezimal → 10
'A' → ASCII → 41
10 → Hexadezimal → A
'1' → ASCII → 41

Nachschlagen: Kapitel 7.4.3 Folie 18 → Wie werden die Sachen gespeichert?
Kapitel 6.4 → Reihenfolge im Speicher

2.4

Folgendes Datenfeld sei gegeben:

```
Var1    DCD     0x10, 0xAA12
```

Geben Sie die Assemblerbefehle an, um das erste Datenwort des Feldes Var1 nach r1 zu kopieren

```
ldr r0, =Var1 ; Arraystartadresse laden
ldr r1, [r0] ; Erstes Element des Arrays
```

Erklärung: Warum nicht mov?
mov kopiert nur ein Datenwort Syntax: MOV <wohin>, <woher,was> → Daten >
Nachschlagen von MOV: Kapitel 6.9.3

Nachschlagen: Kapitel 7.5.3 Kapitel 7.7.3

2.5

Was steht in r0 nach folgendem Befehl (hex.)?

```
ldr             r0 , =0x1234ABCD
```

Lösung: r0 = 0x1234ABCD

Erklärung: Wenn nach '=' ein Hexwert kommt dann speichere den Wert.
Wenn Variable, dann speichere die Adresse.
Auch hier würde mov nicht funktionieren, da 0x1234ABCD > 8 Bit

Nachschlagen: http://www.keil.com/support/man/docs/armasm/armasm$_d om$136128987
https : *//www.raspberrypi.org/forums/viewtopic.php?t* = 16528

2.6

In welchem Wertebereich muss r0 liegen, damit ein Sprung nach LOOP erfolgt? (dezimal oder hex.)

```
mov             r1 , #-15
cmp             r0 , r1
bge             LOOP    ; if greater or equal
```

Größer oder gleich:
Kleiner oder gleich:

Lösung: Größer oder gleich: -15
Kleiner oder gleich: 255

Erklärung: greater or equal → r1 muss >= -15 mov r1 → 8-Bit → 1 muss <= 255

Nachschlagen: bge → Kapitel 8.3.5

2.7

Was steht in r0 nach folgender Befehlssequenz (hex.)?

```
ldr             r1, =0xFFFFFF87
mov             r0, #0x78
and             r0, r1
```

```
r1  1111 1111 1111 1111 1111 1111 1000 0111
r0  0000 0000 0000 0000 0000 0000 0111 1000
-------------------------------------------
r0  0000 0000 0000 0000 0000 0000 0000 0000
```

Erklärung: logisches UND nur wenn gleiche Werte in r1 und r0 stehen → 1

Nachschlagen: Kapitel 8.4.3