Lösungen zu den Aufgaben zu Basiswissen IT

1. **Der (nicht so) geheime Code**

   (a) Fasst man die Bytes als ASCII-Code auf erhält man folgenden String: `Musicology`. (Die intendierte Aufgabenstellung war: `01001101 01110101 01110011 01101001 01100011 01101111 01101100 01101111 01100111 01111001`)

   (b) Der Code besteht aus Paketen zu von 8-Bit Dualzahlen. Als Hexadezimalzahlen aufgefasst ergibt sich:
   `0x4d 0x75 0x73 0x69 0x63 0x6f 0x6c 0x6f 0x67 0x79`
   In Dezimalzahlen umgewandelt:
   `77 117 115 105 99 111 108 111 103 121`
   und in Oktalzahlen:
   `0115 0165 0163 0151 0143 0157 0154 0157 0147 0171`

   (c) Die Checksumme ist 43, das entspricht im ASCII-Code dem Pluszeichen '+';

2. **Na logisch logisch**

   (a) Die XOR Funktion (Exklusiv-Oder) entspricht dem umgangssprachlichen "Entweder-Oder", d.h "Entweder a oder b wahr, aber nicht beide". Mit NOT, AND, OR ausgedrückt:

   $$a \texttt{ XOR } b = (a \texttt{ OR } b) \texttt{ AND NOT } (a \texttt{ AND } b)$$

   Oder:

   $$a \texttt{ XOR } b = (a \texttt{ OR } b) \texttt{ AND } (\texttt{NOT } a \texttt{ OR NOT } b)$$

   (b) Wir zeigen nur exemplarisch den ersten Fall
   $a \texttt{ AND } b = \texttt{ NOT } (\texttt{NOT } a \texttt{ OR NOT } b)$
   Fasst man in den Wahrheitstafeln (WT) das erste Argument als die Zeilen und das zweite als die Spalten auf, so kann man feststellen, dass Negation des ersten Arguments der Vertauschung von Zeilen und Negation des zweiten Arguments einer Vertauschung der Spalten entspricht. Die Negation einer ganzen Aussage entspricht der Vertauschung von 0en und 1en in der ganzen WT. Gehen wir von der rechten Seite aus. Die `OR` Tafel lautet:

OR	1	0
1	1	1
0	1	0

Nun vertauschen wir die Zeilen (`NOT` a):

	1	0
1	1	0
0	1	1

Dann die Spalten (`NOT` b):

	1	0
1	0	1
0	1	1

und negieren die ganze Tafel (`NOT` (`NOT` a `OR` `NOT` b)):

	1	0
1	1	0
0	0	0

Das ist aber gerade die `AND`-Tafel.

(c) Wieviel verschiedene Wahrheitstafeln gibt es? $2^4 = 16$.

3. **Little Big Endian**
Der Kellerspeicher tut im Prinzip das, was wir wollen: Er dreht die Reihenfolge der Daten um, die wir in ihm ablegen. Das führt auf folgendes simples Programm:

```
0x01  MOV 0x01  # Lade Byte 0x01
0x02  PUSH      # ...  und ab auf den Stapel
0x02  MOV 0x02  # usw.
0x04  PUSH
0x05  MOV 0x03
0x06  PUSH
0x07  MOV 0x04
0x08  PUSH
0x09  POP       # Hole oberstes Byte von Stapel = Byte 0x04
0x0A  STO 0x01  # ...  und speichere es an Adresse 0x01
0x0B  POP       # usw.
0x0C  STO 0x02
0x0D  POP
0x0E  STO 0x03
0x0F  POP
0x10  STO 0x04
```

Die Zeilen `0x08` und `0x09` sind eigenlich überflüssig und stehen nur der vollständigkeithalber da. Man braucht also eigenlich nur 14 Programmschritte. Es geht aber noch etwas kürzer. Die Idee ist Byte `0x1` und `0x04` sowie Byte `0x02` und `0x03` zu vertauschen:

```
0x01  MOV 0x01  # Lade Byte 0x01
0x02  PUSH      # ...  und ab auf den Stapel
0x03  MOV 0x04  # Lade Byte 0x04
0x04  STO 0x01  # ...  und speichere es an Adresse 0x01
0x05  POP       # Hole Byte 0x01 vom Stack
0x06  STO 0x04  # ...  und speichere es an Adresse 0x04
0x07  MOV 0x02  # Lade Byte 0x02
0x08  PUSH      # ...  und ab auf den Stapel
0x09  MOV 0x03  # Lade Byte 0x03
0x0A  STO 0x02  # ...  und speichere es an Adresse 0x02
0x0B  POP       # Hole Byte 0x02 vom Stack
0x0C  STO 0x03  # ...  und speichere es an Adresse 0x03
```

So haben wir nochmal 2 Operationen gespart.