Aufgabe 1 - 24.10.2022

Aufgabe 1.1 - Befehle in der Konsole

Befehl: hostname

```
si34yiz@rspi08:~$ hostname
rspi08.inf-ra.uni-jena.de
```

Befehl: lscpu

```
si34yiz@rspi08:~$ lscpu
Architecture:            aarch64
  CPU op-mode(s):        32-bit, 64-bit
  Byte Order:            Little Endian
CPU(s):                  4
  On-line CPU(s) list:   0-3
Vendor ID:               ARM
  Model name:            Cortex-A72
    Model:               3
    Thread(s) per core:  1
    Core(s) per cluster: 4
    Socket(s):           -
    Cluster(s):          1
    Stepping:            r0p3
    CPU max MHz:         1500,0000
    CPU min MHz:         600,0000
    BogoMIPS:            108.00
    Flags:               fp asimd evtstrm crc32 cpuid
Caches (sum of all):
  L1d:                   128 KiB (4 instances)
  L1i:                   192 KiB (4 instances)
  L2:                    1 MiB (1 instance)
Vulnerabilities:
  Itlb multihit:         Not affected
  L1tf:                  Not affected
  Mds:                   Not affected
  Meltdown:              Not affected
  Mmio stale data:       Not affected
  Retbleed:              Not affected
  Spec store bypass:     Vulnerable
  Spectre v1:            Mitigation; __user pointer sanitization
  Spectre v2:            Vulnerable
  Srbds:                 Not affected
  Tsx async abort:       Not affected
```

Aufgabe 1.2 - Binäre Darstellung von char und int

l_data1

```
unsigned char l_data1  = 1;
> data1 = 00000001
```

Die Ausgabe ist die normale Binärdarstellung der Zahl 1_{10}.

l_data2

```
unsigned char l_data2 = 255;
> data2 = 11111111
```

Die Ausgabe ist die normale Binärdarstellung der Zahl 255_{10}.

l_data3

```
unsigned char l_data3 = l_data2 + 1;
> data3 = 00000000
```

Es wird eine 1 zur 255 addiert. Das Ergebnis ist $256_{10}(100000000_2)$. Zur Speicherung würden min. 9 Bits benötigt, der Char-Datentyp kann allerdings nur 8 Bit speichern. Somit kommt es zum ‚overflow', also wird die führende 1 ignoriert und wieder bei 0 angefangen.

l_data4

```
unsigned char l_data4 = 0xA1;
> data4 = 10100001
```

Ausgabe ist die Binärdarstellung der Zahl $A1_{16}$. Das führende ‚0x' zeigt an, dass die Zahl in Hexadezimaldarstellung angegeben wird.

l_data5

```
unsigned char l_data5 = 0b1001011;
> data5 = 01001011
```

Ausgabe ist die Binärdarstellung der Zahl 01001011_2. Das führende ‚0b' zeigt an, dass die Zahl in Binärdarstellung angegeben wird.

l_data6

```
unsigned char l_data6 = 'H';
> data6 = 01001000
```

Ausgabe ist der ASCII-Codepoint des Buchstaben ‚H' (Hex: 48 → 0100 1000).

l_data7

```
char l_data7 = -4;
> data7 = 11111100
```

Ausgabe ist das Radix-Komplement der Zahl 4_{10} ($0000\ 0100_2$). Dafür wird ein Bitflip durchgeführt (Ergebnis: $1111\ 1011_2$) und eine 1 addiert.

l_data8

```
unsigned int l_data8 = 1u << 11;
> data8 = 00000000000000000000100000000000
```

Die Zahl auf der linken Seite (1_{10} als unsigned integer) wird um 11 Stellen nach links verschoben (An die ‚leeren' Stellen wird eine 0 eingesetzt).

l_data9

```
unsigned int l_data9 = l_data8 << 21;
> data9 = 00000000000000000000000000000000
```

Das Ergebnis von l_data8 wird um weitere 21 Stellen nach links verschoben. Somit wurde die 1 insgesamt um 32 Stellen nach links verschoben. Es tritt derselbe Effekt wie bei l_data3 auf: Das 33ste Bit (was die ursprüngliche 1 wäre) wird verworfen und die Zählung fängt erneut bei 0 an.

l_data10

```
unsigned int l_data10 = 0xFFFFFFFF >> 5;
> data10 = 00000111111111111111111111111111
```

Das Ergebnis von $FFFFFFFF_{16}$ wird um 5 Stellen nach rechts verschoben, also auf der linken Seite mit Nullen aufgefüllt.

l_data11

```
unsigned int l_data11 = 0b1001 ^ 0b01111;
> data11 = 00000000000000000000000000000110
```

Ein bitweises XOR wird auf die beiden Zahlen angewandt. Dabei werden nur die Bits 2 und 3 auf 1 gesetzt, da Zahl1 dort jeweils 0 und Zahl2 dort jeweils 1 ist.

l_data12

```
unsigned int l_data12 = ~0b1001;
> data12 = 11111111111111111111111111110110
```

Durch den Operator wird ein bitweises Komplement also ein Bitflip der Zahl durchgeführt.

l_data13

```
unsigned int l_data13 = 0xF0 & 0b1010101;
> data13 = 00000000000000000000000001010000
```

Ein bitweises Und (AND) wird auf die beiden Zahlen angewandt.

l_data14

```
unsigned int l_data14 = 0b001 | 0b101;
> data14 = 00000000000000000000000000000101
```

Ein bitweises inklusives Oder (OR) wird auf die beiden Zahlen angewandt.

l_data15

```
unsigned int l_data15 = 7743;
> data15 = 00000000000000000001111000111111
```

Normale Binärdarstellung der Zahl 7743_{10}.

l_data16

```
int l_data16 = -7743;
> data16 = 11111111111111111110000111000001
```

Binärdarstellung des Radix-Komplements der Zahl 7743_{10}.