

# Hardwarenahe Programmierung

## Musterlösung zur Klausur vom 6. Februar 2017

### Aufgabe 2: Speicherformate von Zahlen

Wir betrachten das folgende Programm ([aufgabe-2.c](#)):

```
1 #include <stdio.h>
2 #include <stdint.h>
3
4 int main (void)
5 {
6     uint16_t numbers[] = { 25928, 27756, 11375, 30496, 29295, 25708, 2593, 0 };
7     printf ("%s", numbers);
8     return 0;
9 }
```

Das Programm wird kompiliert und ausgeführt:

```
$ gcc -Wall aufgabe-2.c -o aufgabe-2
aufgabe-2.c: In function 'main':
aufgabe-2.c:7:3: warning: format '%s' expects argument of type 'char ',
                but argument 2 has type 'uint16_t ' [-Wformat]
$ ./aufgabe-2
Hello, world!
```

- (a) Erklären Sie die beim Compilieren auftretende Warnung. (2 Punkte)
- (b) Erklären Sie die Ausgabe des Programms.  
Welche Endianness hat der verwendete Rechner? (4 Punkte)
- (c) Erklären Sie die Ausgabe des Programms, wenn Sie den Datentyp `uint16_t` durch `uint32_t` ersetzen. (3 Punkte)
- (d) Erklären Sie die Ausgabe des Programms und die beim Compilieren auftretenden Warnungen, wenn Sie den Datentyp `uint16_t` durch `uint8_t` ersetzen. Warum tritt die Warnung aus den vorherigen Aufgabenteilen nicht mehr auf? (3 Punkte)

### Lösung

- (a) **Erklären Sie die beim Compilieren auftretende Warnung.**

Die Funktion `printf()` mit der Formatspezifikation `%s` erwartet als Parameter einen String, d. h. einen Zeiger auf `char`. Der übergebene Parameter `numbers` ist hingegen der Name eines Arrays und somit kompatibel zu Zeigern auf Elemente des Arrays, also auf `uint16_t`, insbesondere nicht auf `char`.

- (b) **Erklären Sie die Ausgabe des Programms.**  
**Welche Endianness hat der verwendete Rechner?**

Die Funktion `printf()` hat einen Zeiger auf das Array `numbers` übergeben bekommen, gibt aber den String `"Hello,_world!\n"` aus. Demnach muß sich in den Speicherzellen des Arrays dieser String befinden:

'H'	'e'	'l'	'l'	'o'	','	'_'	'w'	'o'	'r'	'l'	'd'	'!'	'\n'
numbers[0]		numbers[1]		numbers[2]		numbers[3]		numbers[4]		numbers[5]		numbers[6]	

Wenn wir die Zeichen gemäß ASCII in Hexadezimalzahlen umrechnen, erhalten wir:

0x48	0x65	0x6c	0x6c	0x6f	0x2c	0x20	0x77	0x6f	0x72	0x6c	0x64	0x21	0x0a
numbers[0]		numbers[1]		numbers[2]		numbers[3]		numbers[4]		numbers[5]		numbers[6]	

Auf einem Big-Endian-Rechner gälte dann:

```

numbers[0] = 0x4865 = 18533
numbers[1] = 0x6c6c = 27756
numbers[2] = 0x6f2c = 28460
numbers[3] = 0x2077 = 8311
numbers[4] = 0x6f72 = 28530
numbers[5] = 0x6c64 = 27748
numbers[6] = 0x210a = 8458

```

Dies stimmt nicht mit dem Quelltext des Programms überein.

Auf einem Little-Endian-Rechner gälte dann:

```

numbers[0] = 0x6548 = 25928
numbers[1] = 0x6c6c = 27756
numbers[2] = 0x2c6f = 11375
numbers[3] = 0x7720 = 30496
numbers[4] = 0x726f = 29295
numbers[5] = 0x646c = 25708
numbers[6] = 0x0a21 = 2593

```

Dies stimmt mit dem Quelltext des Programms überein.

Zusammenfassung: `printf()` interpretiert die Speicherzellen des Arrays `numbers` als ASCII-Zeichen. Wenn man die Zahlen Little-Endian im Speicher ablegt, ergibt sich daraus der String `"Hello,_world!\n"`. Der verwendete Rechner hat daher insbesondere die Endianness Little-Endian.

Die oben nicht dargestellte Zahl `numbers[7]` enthält eine Null. Diese dient als Ende-Markierung des Strings und sieht in Big-Endian und in Little-Endian gleich aus (`0x00 0x00`).

- (c) **Erklären Sie die Ausgabe des Programms, wenn Sie den Datentyp `uint16_t` durch `uint32_t` ersetzen.**

Die Ausgabe lautet dann: `He`.

Wenn wir 32-Bit-Zahlen mit 16-Bit-Werten initialisieren, werden die Zahlen von links mit Nullen aufgefüllt. Auf einem Little-Endian-Rechner bedeutet das, daß sie rechts angehängt werden. Nachfolgende Speicherzellen enthalten daher Nullen:

'H'	'e'	0x00	0x00	'l'	'l'	0x00	0x00	'o'	','	0x00	0x00	usw.
numbers[0]				numbers[1]				numbers[2]				

Die Funktion `printf()` interpretiert das Null-Zeichen hinter `He` als String-Ende-Markierung und hört entsprechend an dieser Stelle mit der Ausgabe auf.

- (d) **Erklären Sie die Ausgabe des Programms und die beim Compilieren auftretenden Warnungen, wenn Sie den Datentyp `uint16_t` durch `uint8_t` ersetzen. Warum tritt die Warnung aus den vorherigen Aufgabenteilen nicht mehr auf?**

Die Ausgabe lautet: `Hlo ol!`. Sie enthält nur jeden zweiten Buchstaben des Strings `"Hello,_world!"`.

Die Warnungen lauten:

```

aufgabe-2d.c:6:25: warning: unsigned conversion from 'int' to
'unsigned char' changes value from '25928' to '72' [-Woverflow]
aufgabe-2d.c:6:32: warning: unsigned conversion from 'int' to
'unsigned char' changes value from '27756' to '108' [-Woverflow]
...

```

Die Warnungen kommen daher, daß `uint8_t`-Variable nur vorzeichenlose 8-Bit-Zahlen, also Zahlen von 0 bis 255, aufnehmen können. Zusätzliche Bits werden abgeschnitten.

Im Falle der ersten Zahl `25928`, die ja die Buchstaben 'H' und 'e' enthält, sind die unteren 8 Bit allein das 'H' (ASCII-Wert: 72). Daher landet auch nur das 'H' im Array.

Die zweite Zahl `27756`, die die nächsten beiden Buchstaben (beide 'l') enthält, wird als `108` gespeichert, also nur als ein einzelnes 'l'.

Dies setzt sich über alle 16-Bit-Zahlen fort und erklärt, wieso nur jeder zweite Buchstabe in der Ausgabe erscheint.

Die Warnung aus den vorherigen Aufgabenteilen tritt hier nicht mehr auf. Dies liegt daran, daß `printf()` ein Array von `char`-Variablen erwartet und ein Array von `uint8_t`-Variablen bekommt. `char`-Variable haben typischerweise 8 Bit und sind daher kompatibel mit `uint8_t`-Variablen.