

Hardwarenahe Programmierung / Angewandte Informatik

Musterlösung zu den Übungsaufgaben – 21. November 2016

Aufgabe 1: Zahlensysteme

Wandeln Sie ohne Hilfsmittel

- | | | |
|--------------------|----------------------------|-----------------|
| • nach Dezimal: | • nach Hexadezimal: | • nach Binär: |
| (a) $0010\ 0000_2$ | (d) $0010\ 0000_2$ | (g) 750_8 |
| (b) 42_{16} | (e) 42_{10} | (h) 42_{10} |
| (c) 17_8 | (f) $192.168.20.254_{256}$ | (i) $AFFE_{16}$ |

Berechnen Sie ohne Hilfsmittel:

- (j) $750_8 \& 666_8$
(k) $A380_{16} + B747_{16}$
(l) $AFFE_{16} >> 1$

(Die tiefgestellte Zahl steht für die Basis des Zahlensystems.)

Lösung

- (a) $0010\ 0000_2 = 0 \cdot 1 + 0 \cdot 2 + 0 \cdot 4 + 0 \cdot 8 + 0 \cdot 16 + 1 \cdot 32 = 32_{10}$
(b) $42_{16} = 4 \cdot 16 + 2 = 66_{10}$
(c) $17_8 = 1 \cdot 8 + 7 = 15_{10}$
(d) $0010_2 = 2_{16}$, $0000_2 = 0_{16} \Rightarrow 0010\ 0000_2 = 20_{16}$
(e) $42_{10} = 32 + 10 = 2 \cdot 16 + 10 = 2A_{16}$
(f) $192 = 160 + 32 = 10 \cdot 16 + 2 \cdot 16 = 12 \cdot 16 = C0_{16}$,
 $168 = 160 + 8 = 10 \cdot 16 + 8 = A8_{16}$,
 $20 = 16 + 4 = 14_{16}$,
 $254 = 255 - 1 = FF_{16} - 1_{16} = FE_{16}$
 $\Rightarrow 192.168.20.254_{256} = C0A814FE_{16}$
(g) $750_8 = 111\ 101\ 000_2$
(h) $42_{10} = 2 \cdot 16 + 10 = 2A_{16} = 0010\ 1010_2$
(i) $A_{16} = 10_{10} = 8 + 2 = 1010_2$,
 $F_{16} = 1111_2$
 $E_{16} = F_{16} - 1_{16} = 1111_2 - 1_2 = 1110_2$
 $\Rightarrow AF_{16} = 1010\ 1111_2$
(j) $750_8 \& 666_8 = 111\ 101\ 000_2 \& 110\ 110\ 110_2 = 110\ 100\ 000_2 = 640_8$
(k)
$$\begin{array}{r} A380_{16} \\ + B747_{16} \\ \hline 15AC7_{16} \end{array}$$

(l) $AFFE_{16} >> 1 = 1010\ 1111\ 1111\ 1110_2 >> 1 = 0101\ 0111\ 1111\ 1111_2 = 57FF_{16}$

Aufgabe 2: Mikro-Controller

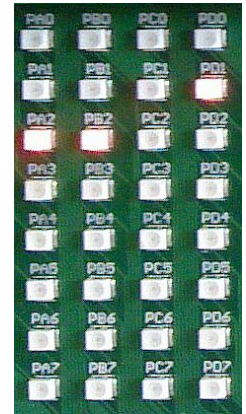
An die vier Ports eines ATmega16-Mikro-Controllers sind Leuchtdioden angeschlossen:

- von links nach rechts an die Ports A, B, C und D,
- von oben nach unten an die Bits Nr. 0 bis 7.

Wir betrachten das folgende Programm:

```
#include <avr/io.h>

int main (void)
{
    DDRA = DDRB = DDRC = DDRD = 0xff;
    PORTA = 0x1f;
    PORTB = PORTD = 0x10;
    PORTC = 0xfc;
    while (1);
    return 0;
}
```



- Was bewirkt dieses Programm?
- Wozu dient die erste Zeile des Hauptprogramms?
- Was würde stattdessen die Zeile `DDRA, DDRB, DDRC, DDRD = 0xff;` bewirken?
- Schreiben Sie das Programm so um, daß die durch das Programm dargestellte Figur spiegelverkehrt erscheint.
- Wozu dient das `while (1)`?

Lösung

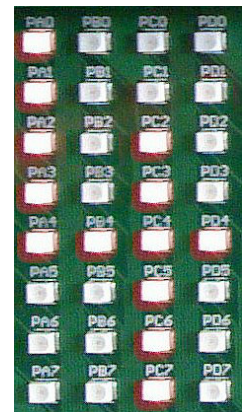
- Was bewirkt dieses Programm?**

Es läßt die folgenden LEDs leuchten:

- in der ersten Spalte die LEDs PA0 bis PA4,
- in der dritten Spalte die LEDs PC2 bis PC7,
- in der zweiten und vierten Spalte die LED PB4 bzw. PD4.

Alle anderen LEDs sind dunkel.

Es entsteht eine Figur, die an die Ziffer 4 erinnert.
(Siehe nebenstehende Skizze.)



- Wozu dient die erste Zeile des Hauptprogramms?**

Diese Anweisung weist den vier *Data Direction Registers* DDRA bis DDRD den Wert 0xff zu (binär: 8 Einsen) und konfiguriert dadurch die Ports PORTA bis PORTD komplett (also mit jeweils allen 8 Bits) als Output-Ports.

- Was würde stattdessen die Zeile `DDRA, DDRB, DDRC, DDRD = 0xff;` bewirken?**

Der Komma-Operator in C bewirkt, daß der linke Operand berechnet und ignoriert wird; der Rückgabewert ist der rechte Operand. Da nur der Ausdruck `DDRD = 0xff` einen **Seiteneffekt** hat, würde nur die Zuweisung an DDRD ausgeführt; DDRA bis DDRB erhielten weiterhin **undefinierte** Werte.

- Schreiben Sie das Programm so um, daß die durch das Programm dargestellte Figur spiegelverkehrt erscheint.**

Die Zuweisungen an die Ports müssen wie folgt vertauscht werden:

```
PORTD = 0x1f;
PORTC = PORTA = 0x10;
PORTB = 0xfc;
```

(e) **Wozu dient das `while (1)`?**

Es bewirkt eine Endlosschleife. Diese ist nötig, weil außer dem Programm nichts auf dem Mikro-Controller läuft. Es gibt nichts, wohin das Programm (via `return`) zurückkehren könnte.

Würde das `return 0` tatsächlich ausgeführt, hätte dies ein **undefiniertes Verhalten** des Mikro-Controllers – also einen **Absturz** – zur Folge.

Aufgabe 3: Arrays mit Zahlen

Wir betrachten das folgende Programm
(Datei: `aufgabe-3.c`):

```
#include <stdio.h>

void f (int *s0, int *s1)
{
    while (*s0 >= 0)
    {
        int *s = s1;
        while (*s >= 0)
            if (*s0 == *s++)
                printf ("%d_", *s0);
        s0++;
    }
    printf ("\n");
}

int main (void)
{
    int a[] = { 10, 4, 3, 7, 12, 0, 1, -1 };
    int b[] = { 7, 14, 0, 8, 9, 22, 10, -1 };
    f (a, b);
    return 0;
}
```

(a) Was bewirkt die Funktion `f` und warum?

(b) Von welcher Ordnung (Landau-Symbol) ist die Funktion und warum?

Wir beziehen uns hierbei auf die Anzahl der Vergleiche in Abhängigkeit von der Länge der Eingabedaten `s0` und `s1`. Für die Rechnung dürfen Sie beide Längen mit n gleichsetzen, obwohl sie normalerweise nicht gleich sind.

(c) Was passiert, wenn Sie beim Aufruf der Funktion für einen der Parameter den Wert `NULL` übergeben und warum?

(d) Was passiert, wenn Sie das Hauptprogramm wie folgt abändern (`aufgabe-3d.c`) und warum?

```
int main (void)
{
    int a[] = { 10, 4, 3, 7, 12, 0, 1 };
    int b[] = { 7, 14, 0, 8, 9, 22, 10 };
    f (a, b);
    return 0;
}
```

(e) Beschreiben Sie – in Worten und/oder als C-Quelltext –, wie sich die Funktion `f` effizienter gestalten läßt, wenn man die ihr übergebenen Arrays `s0` und `s1` als sortiert voraussetzt.

Hinweis: Wie würden Sie als Mensch die Aufgabe erledigen?

(f) Von welcher Ordnung (Landau-Symbol) ist Ihre effizientere Version der Funktion und warum?

Lösung

(a) **Was bewirkt die Funktion `f` und warum?**

Die Funktion vergleicht jedes Element des Arrays `s0` mit jedem Element des Arrays `s1` und gibt bei Gleichheit das Element aus. Sie gibt also die Schnittmenge beider Arrays aus.

(b) **Von welcher Ordnung (Landau-Symbol) ist die Funktion und warum?**

Die Funktion enthält zwei ineinandergeschachtelte Schleifen, die jeweils bis n gehen. Die Ordnung ist somit $\mathcal{O}(n^2)$.

(c) **Was passiert, wenn Sie beim Aufruf der Funktion für einen der Parameter den Wert `NULL` übergeben und warum?**

In den Bedingungen beider `while`-Schleifen wird jeweils einer der als Parameter übergebenen Zeiger dereferenziert: `while (*s0 > 0)` greift auf den Inhalt von `s0` zu, `while (*s > 0)` auf den von `s1`, da vorher in der Initialisierung von `s` eine Zuweisung `s = s1` erfolgte.

Wenn einer dieser Zeiger den Wert `NULL` hat, bewirkt dies einen **unzulässigen Speicherzugriff** und somit einen **Absturz**.

- (d) Was passiert, wenn Sie das Hauptprogramm wie folgt abändern ([aufgabe-3d.c](#)) und warum?

```
int main (void)
{
    int a[] = { 10, 4, 3, 7, 12, 0, 1 };
    int b[] = { 7, 14, 0, 8, 9, 22, 10 };
    f (a, b);
    return 0;
}
```

Beide Schleifen in der Funktion `f()` brechen ab, sobald sie auf eine negative Zahl stoßen.

In dem abgeänderten Hauptprogramm haben die Arrays keine Ende-Markierung mehr. Demzufolge lesen beide Schleifen immer weiter, bis sie im Speicher „zufällig“ auf eine negative Zahl stoßen. Dies kann einen **unzulässigen Speicherzugriff** und somit einen **Absturz** zur Folge haben.

- (e) Beschreiben Sie – in Worten und/oder als C-Quelltext –, wie sich die Funktion `f` effizienter gestalten läßt, wenn man die ihr übergebenen Arrays `s0` und `s1` als sortiert voraussetzt.

Lösung in Worten:

Wenn beide Arrays sortiert sind, besteht die Möglichkeit, beide Arrays gleichzeitig mit jeweils einem Index durchzugehen:

- Wenn das Element in `s0` kleiner ist als das in `s1`, erhöhen wir den Index für `s0`.
- Wenn das Element in `s0` größer ist als das in `s1`, erhöhen wir den Index für `s1`.
- Wenn die Elemente in `s0` und `s1` gleich groß sind, geben wir das Element aus und erhöhen beide Indices.

Lösung als C-Quelltext:

Die Dateien [loesung-3e-1.c](#) und [loesung-3e-2.c](#) setzen das o. a. Verfahren um.

- [loesung-3e-1.c](#) verwendet `int`-Variable als Array-Indices.
- [loesung-3e-2.c](#) verwendet stattdessen Zeiger-Arithmetik.

Beide Lösungen sind in gleicher Weise korrekt.

- (f) Von welcher Ordnung (Landau-Symbol) ist Ihre effizientere Version der Funktion und warum?

Da nur noch eine Schleife bis `n` ausgeführt wird, ist die Ordnung $\mathcal{O}(n)$.