

# Grundlagen der Informatik III

Wintersemester 2010/2011

Wolfgang Heenes, Patrik Schmittat



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

## 2. Aufgabenblatt

08.11.2010

**Hinweis:** Der Schnelltest und die Aufgaben sollen in den Übungsgruppen bearbeitet werden. Die Hausaufgaben sind in der Kalenderwoche 46 (15.11. bis 19.11.) bei den Tutoren in **physikalischer Form** (handschriftlich oder gedruckt) abzugeben. Bei allen Abgaben ist der Name des Tutors und die Übungsgruppe deutlich anzugeben. Bei Teamabgaben wird nur eine Lösung eingereicht, die alle Namen der Teammitglieder enthält.

Schicken Sie Ihre Lösungen von Programmieraufgaben *zusätzlich* zur schriftlichen Abgabe per E-Mail an Ihren Tutor. Kommentieren Sie Ihren Quellcode.

### Aufgabe 1: Schnelltest

Fragen	Antworten
1. Der Instruktionssatz in der Assembler-Programmierung ist	<input type="checkbox"/> eine Maschinensprache zur Beschreibung der Aktionen, die ein Prozessor ausführen kann. <input type="checkbox"/> eine von der Hardwareebene stark abstrahierte Programmiersprache. <input type="checkbox"/> eine Schnittstelle (Interface) zwischen Compiler und Rechner. <input type="checkbox"/> eine möglichst einfach gehaltene, primitive Programmiersprache.
2. Die Zahl $17173193_{10} = 1\ 00000110\ 00001010\ 11001001_2$ wird als 32-Bit-Wort im Speicher einer Big-Endian-Architektur abgelegt. Wie sieht das zugehörige Speicherlayout aus?	<input type="checkbox"/> $\begin{array}{cccccccc} 10000000 & 01100000 & 01010000 & 10010011 \\ 31 & 24 & 23 & 16 & 15 & 8 & 7 & 0 \end{array}$ <input type="checkbox"/> $\begin{array}{cccccccc} 00000001 & 00000110 & 00001010 & 11001001 \\ 31 & 24 & 23 & 16 & 15 & 8 & 7 & 0 \end{array}$ <input type="checkbox"/> $\begin{array}{cccccccc} 11001001 & 00001010 & 00000110 & 00000001 \\ 31 & 24 & 23 & 16 & 15 & 8 & 7 & 0 \end{array}$ <input type="checkbox"/> $\begin{array}{cccccccc} 10010011 & 01010000 & 01100000 & 10000000 \\ 31 & 24 & 23 & 16 & 15 & 8 & 7 & 0 \end{array}$
3. Die gleiche Zahl wird im Speicher einer Little-Endian-Architektur abgelegt. Wie sieht dort das zugehörige Speicherlayout aus?	<input type="checkbox"/> $\begin{array}{cccccccc} 10000000 & 01100000 & 01010000 & 10010011 \\ 31 & 24 & 23 & 16 & 15 & 8 & 7 & 0 \end{array}$ <input type="checkbox"/> $\begin{array}{cccccccc} 00000001 & 00000110 & 00001010 & 11001001 \\ 31 & 24 & 23 & 16 & 15 & 8 & 7 & 0 \end{array}$ <input type="checkbox"/> $\begin{array}{cccccccc} 11001001 & 00001010 & 00000110 & 00000001 \\ 31 & 24 & 23 & 16 & 15 & 8 & 7 & 0 \end{array}$ <input type="checkbox"/> $\begin{array}{cccccccc} 10010011 & 01010000 & 01100000 & 10000000 \\ 31 & 24 & 23 & 16 & 15 & 8 & 7 & 0 \end{array}$
4. Welche Aussagen über Assembler-Programmierung sind korrekt?	<input type="checkbox"/> Programmcode kann an beliebiger Stelle im Speicher liegen und trotzdem ausgeführt werden. <input type="checkbox"/> Zugriffe auf Variablen (im .data Teil) können nur über den Befehl <code>leal</code> (load effective address) durchgeführt werden. <input type="checkbox"/> Assemblerprogramme können mit wenig Aufwand von einem System auf ein anderes portiert werden. <i>Fortsetzung nächste Seite...</i>

Fragen	Antworten
	<input type="checkbox"/> Spezielle Erweiterungen des Prozessors sind durch Assembler nutzbar und erlauben es Lösungen zu bestimmten Problemen schneller zu berechnen als in Hochsprachen.
<b>5.</b> Welche Aussagen treffen auf den Stack zu?	<input type="checkbox"/> Das Betriebssystem sorgt dafür, dass Werte automatisch vom Stack entfernt werden, nachdem sie nicht mehr benötigt werden. <input type="checkbox"/> Es können immer nur Elemente am Ende des Stacks angefügt werden. <input type="checkbox"/> Der Stack arbeitet nach dem first-in-first-out (FIFO) Prinzip. <input type="checkbox"/> Rekursive Funktionsaufrufe können mittels Kellerspeicher realisiert werden. <input type="checkbox"/> Der Stack kann beliebig groß werden bis der verfügbare Speicher im Rechner verbraucht ist.

## Aufgabe 2: Kontrollstrukturen

Wandeln Sie die folgenden Codebeispiele in IA32-Assembler um. Die Variablen  $a, b, c$  sind in den Registern `%eax`, `%ebx`, `%ecx` abgelegt.

a) `if (a < b) c++;`

b) 

```
c = 0;
while (a > 0) {
    a -= b;
    c++;
}
```

c) `a = (b < c/2) ? b : c;`

Diese Anweisung ist eine Kurzschreibweise für `if (b < c/2) a = b; else a = c;`

d) 

```
a = 0;
for (b = 0; b < 1024; b++) {
    a += feld[b];
}
```

## Aufgabe 3: Speicherzugriff

Die Variablen  $a, b, c$  sind in den Registern `%eax`, `%ebx`, `%ecx` abgelegt. Zusätzlich befindet sich ein Array an der Speicheradresse des Labels `feld` mit 1024 Datenworten à 32-Bit, dies entspricht also folgender Java-Deklaration: `int[] feld = new int[1024];`

Wandeln Sie die folgenden Befehle in IA32-Assembler um. Sie brauchen keine Bereichsprüfung des Arrays durchzuführen, beachten Sie jedoch wieder, dass die Werte und Variablen als long anzusehen sind.

a) `a = feld[5];`

- b) `feld[c] = b;`
- c) `feld[a]++;`
- d) `feld[c] = feld[b] + feld[feld[0]];`

## Aufgabe 4: Die Methode der Sprungtabellen

Gegeben sei die folgende Java-Anweisung:

```
public class SwitchStatement {  
  
    public static void main(String[] args) {  
        int i = 4, j = 0;  
  
        switch(i) {  
            case 4: case 6: case 88: j = 1; break;  
            case 5: case 86: case 87: j = 2; break;  
            case 7: j = 3; break;  
            case 85: j = 4; break;  
            case 8: case 89: j = 5; break;  
        }  
  
        System.out.println(j);  
    }  
}
```

Benutzen Sie die Methode der Sprungtabellen. Wenn Sie die verwendeten Werte in dem `switch` untersuchen, können Sie zwei Wertebereiche ausmachen. Arbeiten Sie daher mit zwei Sprungtabellen, getrennt für jeden dieser Bereiche.

- a) Formulieren Sie ein äquivalentes IA32-Assemblerprogramm, welches die Sprungtabelle im `.data`-Teil ablegt.
- b) Testen Sie Ihr Programm mit folgenden Werten:
  - `i = 3`
  - `i = 7`
  - `i = 20`
  - `i = 86`
  - `i = 90`

## Hausaufgabe 1: Matrikelprüfung (4 Punkte)

Jedes Semester müssen neue Matrikelnummern für die Erstsemester erstellt werden. Da diese einem System folgen, müssen sie erst berechnet werden. Nun ist aber das VB-Programm<sup>1</sup> von Ihrem Vorgänger zu langsam und Sie entscheiden sich es in Assembler zu schreiben, damit es mit der Flut von Einschreibungen klar kommt.

Das Testverfahren ist eine gewichtete Summe über die Ziffern wobei die letzte Stelle die Prüfsumme ist.

Matrikelnummer: abcdefg  
 $(a * 9 + b * 7 + c * 3 + d * 9 + e * 7 + f * 3) \bmod 10 = g$

- a) Schreiben Sie ein IA32-Assemblerprogramm (ATT- oder Intel-Syntax) und kommentieren Sie den Code.
- b) Testen Sie, welche der folgenden Matrikelnummern gültig sind:

<sup>1</sup> Visual Basic

- 0939182
- 1234567
- 0611813
- 1398140
- 1569203

## Hausaufgabe 2: Sortieren (6 Punkte)

Gegeben sei nun das folgende Java-Programm. Es sortiert eine Reihung mit zehn Integern (**a**[0] bis **a**[9]).

```
import java.util.Arrays;

public class BubbleSort {

    public static void main(String[] args) {
        int a[] = {16, 5, 18, 12, 11, 66, 62, 1, 9, 33};
        int n = 10; // Nicht unbedingt notwendig in Java, aber in IA-32

        for (int i = n - 1; i > 0; i--) {
            for (int j = 0; j < i; j++) {
                if (a[j] > a[j+1]) {
                    int temp = a[j];
                    a[j] = a[j+1];
                    a[j+1] = temp;
                }
            }
        }

        // Ausgabe muss nicht programmiert werden,
        // hilft aber bei der Fehlersuche
        System.out.println(Arrays.toString(a));
    }
}
```

Benutzen Sie dieses Schema, um ein entsprechendes IA32-Assemblerprogramm (ATT- oder Intel-Syntax, Kommentierung nicht vergessen) zu schreiben. Verwenden Sie soweit sinnvoll den **loop**-Befehl. Die zu sortierenden Daten stehen im .data-Teil. Als Datentyp soll long verwendet werden.

Testen Sie Ihr Programm mit folgenden Werten:

- a = (10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
- a = (16, 5, 18, 12, 11, 66, 62, 1, 9, 33)