Grundlagen der Informatik III

Wintersemester 2010/2011

Wolfgang Heenes, Patrik Schmittat

TECHNISCHE UNIVERSITÄT DARMSTADT

2. Aufgabenblatt mit Lösungsvorschlag

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*	■ *eine Maschinensprache zur Beschreibung der Aktionen, die ein Prozessor ausführen kann.* ☐ *eine von der Hardwareebene stark abstrahierte Programmiersprache.* ■ *eine Schnittstelle (Interface) zwischen Compiler und Rechner.* ■ *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?*	☐ $\underset{31\ \ 24}{\underline{10000000}}\ \underset{23\ \ 16}{\underline{01100000}}\ \underset{15\ \ 8}{\underline{01010000}}\ \underset{7\ \ 0}{\underline{10010011}}$ ☐ $\underset{31\ \ 24}{\underline{00000001}}\ \underset{23\ \ 16}{\underline{00000110}}\ \underset{15\ \ 8}{\underline{00001010}}\ \underset{7\ \ 0}{\underline{11001001}}$ ■ $\underset{31\ \ 24}{\underline{11001001}}\ \underset{23\ \ 16}{\underline{00001010}}\ \underset{15\ \ 8}{\underline{00000110}}\ \underset{7\ \ 0}{\underline{00000001}}$ ☐ $\underset{31\ \ 24}{\underline{10010011}}\ \underset{23\ \ 16}{\underline{01010000}}\ \underset{15\ \ 8}{\underline{01100000}}\ \underset{7\ \ 0}{\underline{10000000}}$
3. *Die gleiche Zahl wird im Speicher einer Little-Endian-Architektur abgelegt. Wie sieht dort das zugehörige Speicherlayout aus?*	☐ $\underset{31\ \ 24}{\underline{10000000}}\ \underset{23\ \ 16}{\underline{01100000}}\ \underset{15\ \ 8}{\underline{01010000}}\ \underset{7\ \ 0}{\underline{10010011}}$ ■ $\underset{31\ \ 24}{\underline{00000001}}\ \underset{23\ \ 16}{\underline{00000110}}\ \underset{15\ \ 8}{\underline{00001010}}\ \underset{7\ \ 0}{\underline{11001001}}$ ☐ $\underset{31\ \ 24}{\underline{11001001}}\ \underset{23\ \ 16}{\underline{00001010}}\ \underset{15\ \ 8}{\underline{00000110}}\ \underset{7\ \ 0}{\underline{00000001}}$ ☐ $\underset{31\ \ 24}{\underline{10010011}}\ \underset{23\ \ 16}{\underline{01010000}}\ \underset{15\ \ 8}{\underline{01100000}}\ \underset{7\ \ 0}{\underline{10000000}}$
4. *Welche Aussagen über Assembler-Programmierung sind korrekt?*	■ *Programmcode kann an beliebiger Stelle im Speicher liegen und trotzdem ausgeführt werden.* ☐ *Zugriffe auf Variablen (im .data Teil) können nur über den Befehl* `leal` *(load effective address) durchgeführt werden.* ☐ *Assemblerprogramme können mit wenig Aufwand von einem System auf ein anderes portiert werden.* *Fortsetzung nächste Seite...*

Fragen	*Antworten*
	■ *Spezielle Erweiterungen des Prozessors sind durch Assembler nutzbar und erlauben es Lösungen zu bestimmten Problemen schneller zu berechnen als in Hochsprachen.*
5. *Welche Aussagen treffen auf den Stack zu?*	☐ *Das Betriebssystem sorgt dafür, dass Werte automatisch vom Stack entfernt werden, nachdem sie nicht mehr benötigt werden.* ■ *Es können immer nur Elemente am Ende des Stacks angefügt werden.* ☐ *Der Stack arbeitet nach dem first-in-first-out (FIFO) Prinzip.* ■ *Rekursive Funktionsaufrufe können mittels Kellerspeicher realisiert werden.* ☐ *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++;
```

Lösungsvorschlag:

```
  cmpl %ebx, %eax
  jge skip
  incl %ecx
skip:
```

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

Lösungsvorschlag:

```
  movl $0, %ecx
while:
  cmpl $0, %eax
  jle end
  subl %ebx, %eax
  incl %ecx
  jmp while
end:
```

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

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

Lösungsvorschlag:

```
  movl %ecx,%eax
  shrl $1,%eax
  cmpl %eax,%ebx
  jge else
then:
  movl %ebx,%eax
  jmp .end
else:
  movl %ecx,%eax
.end:
```

d)

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

Lösungsvorschlag:

```
  movl $0, %eax
  movl $0, %esi
for:
  cmpl $1024, %esi
  jge end
  addl feld(, %esi, 4), %eax
  incl %esi
  jmp for
end:
```

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];`

```
movl $5, %edi
movl feld(, %edi, 4), %eax
```

b) `feld[c] = b;`

```
movl %ecx, %edi
movl %ebx, feld(, %edi, 4)
```

c) `feld[a]++;`

```
movl %eax, %edi
incl feld(, %edi, 4)
```

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

```
movl $0, %esi # Adresse erstes Feldelement
movl feld(,%esi,4),%edi
movl feld(,%edi,4),%eax
movl %ebx, %esi
addl feld(, %esi, 4), %eax
movl %eax, feld(, %ecx, 4) # jedes Register verwendbar, Hinweise in Vorlesung beachten
```

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.*

Lösungsvorschlag:

```
.data
intout:  .string "Ergebnis␣%d\n"
stab1: .long .L4, .L5, .L6, .L7, .L8
stab2: .long .L85, .L86, .L87, .L88, .L89
i: .long 4 # Auswahl
j: .long 0

.text
.globl main
main:

    # Werte laden
    movl i, %ebx
```

```
    movl $0, %eax
    # i <= 3
    cmpl $3, %ebx
    jle end
    # 4 <= i <= 8 (erste Tabelle)
    cmpl $8, %ebx
    jle first
    # i <= 84
    cmpl $84, %ebx
    jle end
    # 85 <= i <= 89 (zweite Tabelle)
    cmpl $89, %ebx
    jle second
    # i >= 90
    jmp end
first:
    # Offset abziehen und reinspringen
    subl $4, %ebx
    jmp *stab1(, %ebx, 4)
second:
    # Offset abziehen und reinspringen
    subl $85, %ebx
    jmp *stab2(, %ebx, 4)

.L4: .L6: .L88:  # j = 1
    movl $1, %eax
    jmp end
.L5: .L86: .L87: # j = 2
    movl $2, %eax
    jmp end
.L7:             # j = 3
    movl $3, %eax
    jmp end
.L85:            # j = 4
    movl $4, %eax
    jmp end
.L8: .L89:       # j = 5
    movl $5, %eax
    jmp end
end:
    movl %eax, j

pushl %eax # Wert in j ausgeben
pushl $intout
call printf

# Exit
movl $1, %eax
int $0x80
```

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[1] 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) mod 10 = g
```

a) *Schreiben Sie ein IA32-Assemblerprogramm (ATT- oder Intel-Syntax) und kommentieren Sie den Code.*

Lösungsvorschlag:

```
.data
intout: # Fuer printf
    .string "Wert␣%u\n"
mulval: # Gewichte (umgekehrte Reihenfolge)
    .long 3, 7, 9, 3, 7, 9
checkval: # Pruefziffer
    .long 0

.text
.globl main

main:
# Matrikelnummer fuer die Berechnung
mov $1234567, %eax

    # Fuer Summe der Zahlen
    movl $0, %ebx
    # Schleifenzaehler
    movl $0, %ecx
    # Konstant 10
    movl $10, %esi
    # Pruefziffer extrahieren
    cdq
    divl %esi
    movl %edx, checkval
for:
    # for i = 0; i < 6; i++
    cmpl $6, %ecx
    jge endfor
    # Ziffer extrahieren
    cdq
    divl %esi
    # Passendes Gewicht laden
    movl %ecx, %edi
    # Ziffer * Gewicht
    imull mulval(, %edi, 4), %edx
    # Aufsummieren
    addl %edx, %ebx
    # i++
```

[1] Visual Basic

```
        incl %ecx
        jmp for
    endfor:
        # Mod 10 rechnen
        movl %ebx, %eax
        cdq
        divl %esi
        # Ergebnis feststellen
        movl $0, %eax
        cmpl %edx, checkval
        jne notmatr
        movl $1, %eax
    notmatr:

    # Wert im %eax ausgeben
    pushl %eax
    pushl $intout
    call printf

    # Exit
    movl $1, %eax
    int $0x80
```

b) *Testen Sie, welche der folgenden Matrikelnummern gültig sind:*

Lösungsvorschlag:

- x 0939182
- x 1234567
- ✓ 0611813
- x 1398140
- x 1569203

Hausaufgabe 2: Sortieren (6 Punkte)

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

```
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)*

Lösungsvorschlag:

```
.data
intout: # Fuer printf
    .string "Wert␣%u\n"
a: # Werte die sortiert werden sollen
    .long 16, 5, 18, 12, 11, 66, 62, 1, 9, 33
n: .long 10

.text
.globl main

main:
    # n in Register laden
    movl n, %ecx
    # konstant 4
    movl $4, %esi
    decl %ecx
for1:
    # for i = n - 1; i > 0; i--
    movl $0, %edi
for2:
    # for j = 0; j < i; j++
    cmpl %edi, %ecx
    jle endfor2
    # a[j] und a[j+1] laden
    movl a(, %edi, 4), %eax
    movl a(%esi, %edi, 4), %ebx
    # if a[j] > a[j+1]
    cmpl %eax, %ebx
    jge nosort
    # Werte im Speicher tauschen
    movl %ebx, a(, %edi, 4)
    movl %eax, a(%esi, %edi, 4)
nosort:
    incl %edi
    jmp for2
endfor2:
    loop for1

    # Sortiertes Array ausgeben
```

```
    movl n, %ecx
    decl %ecx
printfor:
    # for i = n - 1; i > 0; i--
    movl %ecx, %edi
    # ecx sichern!
    pushl %ecx
    # Wert an a[edi] ausgeben
    pushl a(, %edi, 4)
    pushl $intout
    call printf
    addl $8, %esp
    popl %ecx
    loop printfor

# Exit
movl $1, %eax
int $0x80
```