Grundlagen der Informatik III

Wintersemester 2010/2011

Wolfgang Heenes, Patrik Schmittat

TECHNISCHE UNIVERSITÄT DARMSTADT

3. Aufgabenblatt mit Lösungsvorschlag

15.11.2010

Hinweis: Der Schnelltest und die Aufgaben sollen in den Übungsgruppen bearbeitet werden. Die Hausaufgaben sind in der Kalenderwoche 47 (22.11. bis 26.11.) 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. *Wo wird bei einem Funktionsaufruf durch den Befehl* `call` *die Rücksprungadresse gesichert?*	■ *Stack* ☐ *Im (.data) Teil* ☐ *Register* ☐ *Festplatte / Massenspeicher*
2. *Wo werden lokale Variablen einer Funktion abgelegt?*	■ *Stack* ☐ *Im (.data) Teil* ■ *Register* ☐ *Festplatte / Massenspeicher*
3. *Wo werden globale Variablen abgelegt?*	☐ *Stack* ■ *Im (.data) Teil* ☐ *Register* ☐ *Festplatte / Massenspeicher*

Aufgabe 2: Zugriffe bei Reihungen

Gegeben ist folgendes IA32-Assemblerprogramm.

```
.data
intout:     .string "Ergebnis␣%d␣\n"
data_items: .long 1,2,3,4,5,6,7
 n:         .long 7 # Anzahl der Elemente
.text
.globl main
main:
     movl $0,%edi # 0 in Indexregister
     movl $0,%ebx # %ebx mit Null initialisieren
.L1:
     movl data_items(,%edi,4),%eax #
     addl %eax,%ebx
     addl $1,%edi
```

```
    cmpl n,%edi
    jle .L1

.ende:  pushl %ebx # Wert im %ebx ausgeben
        pushl $intout
        call printf

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

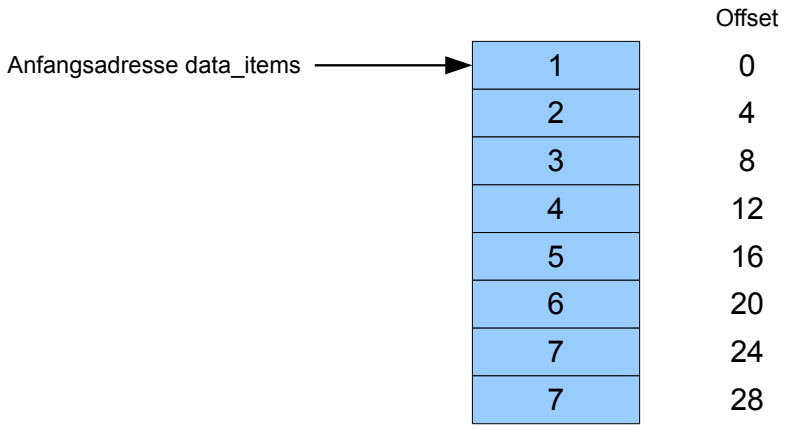
a) *Welcher Wert steht im Register %ebx nach Ausführung des Programms?*

Lösungsvorschlag:

Im Register steht der Wert 35. Da n im Speicher hinter $data\_items$ abgelegt wird, wird der Wert von n noch zum Ergebnis hinzuaddiert.

b) *Veranschaulichen Sie sich das Ergebnis in einer Grafik.*

Lösungsvorschlag:



Aufgabe 3: Bitmanipulation

Zum Schutz vor illegalen Kopien verlangt ein Programm bei der Installation eine achtstellige (hexadezimale) Seriennummer. Um die Gültigkeit einer solchen Seriennummer zu prüfen, werden die Bits der Stellen 7 bis 9 und 23 bis 25 jeweils als eine Zahl interpretiert (Es wird von rechts nach links von 0 bis 31 gezählt). Wenn die Summe dieser beiden Zahlen 12 beträgt, ist die Seriennummer gültig. Hierbei betrachten wir das niederwertigste Bit als Bit mit Index 0.

a) *Gegeben sei die Seriennummer $42C27F91_{16}$. Zeigen Sie, dass dies eine gültige Seriennummer ist.*

Lösungsvorschlag:

Aus der Binärdarstellung `0100 0010 1100 0010 0111 1111 1001 0001` geht hervor, dass an der 7 bis 9 Stelle der Binärwert $a = 111_2$ und an der 23. bis 25. Stelle der Binärwert $b = 101_2$ steht. Addition dieser beiden Werte ergibt 1100_2, was dem Wert 12 entspricht. Damit ist die angegebene Seriennummer gültig.

b) *Wie viele gültige Seriennummern gibt es?*

Lösungsvorschlag:

Eine gültige Seriennummer hat folgende Form: `XXXX XXbb bXXX XXXX XXXX XXaa aXXX XXXX`. Die Bit-Werte der mit `X` markierten Stellen sind egal, und die Werte `aaa` und `bbb` müssen so gewählt sein, dass `aaa` + `bbb` = 12. Durch Probieren findet man drei Möglichkeiten, um die Zahl 12 als Summe zweier 3-Bit-Zahlen darzustellen: $6+6$, $5+7$ und $7+5$. Für die 6 Bits `aaa` und `bbb` gibt es also 3 mögliche Kombinationen, während die übrigen $32-6=26$ Bits beliebige Werte annehmen können. Folglich gibt es $3 \cdot 2^{26}$ gültige Seriennummern.

c) *Erzeugen Sie eine weitere gültige und eine ungültige Seriennummer. Zeigen Sie deren Gültigkeit bzw. Ungültigkeit.*

Lösungsvorschlag:

Eine gültige Seriennummer ist z.B. `1111 0110 1101 0000 0111 1000 1101 1101`
Eine ungültige ist z.B. `0100 0000 1100 0110 1000 1001 0001 0001`.

d) *Schreiben Sie ein IA32-Assemblerprogramm, welches die Gültigkeit einer Seriennummer prüft. Nehmen Sie hierzu an, dass die Seriennummer zu Beginn im Register* ***%eax*** *gespeichert ist. Ist diese Seriennummer gültig, soll das Register* ***%eax*** *am Ende des Codes den Wert* 1 *enthalten, andernfalls den Wert* 0*.*

Lösungsvorschlag:

```
.data
intout:
    .string "Wert␣%u\n"

.text
.globl main

main:
# Zu ueberpruefende Seriennummer
movl $0x42C27F91, %eax

    # Stellen 7 - 9 extrahieren
    movl %eax, %ebx
    andl $0x00000380, %ebx
    shrl $7, %ebx
    # Stellen 23 - 25 extrahieren
    movl %eax, %ecx
    andl $0x03800000, %ecx
    shrl $23, %ecx
    # Testen ob Addition 12 ergibt
    addl %ebx, %ecx
    movl $0, %eax
    cmpl $12, %ecx
    jne end
    movl $1, %eax
end:

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

# Exit
movl $1, %eax
```

```
     int $0x80
```

Aufgabe 4: Set-Instruktionen beim IA32

Gegeben ist folgendes IA32-Assemblerprogramm.

```
.data
 intout: .string "Ergebnis␣%d␣\n"
 a:      .long  300000
 b:      .long  300000

.text
.globl main
main:
     movl   a,%eax
     movl   b,%ebx
     imull  %eax,%ebx

.ende:   pushl %ebx
         pushl $intout
         call printf

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

a) *Was erwarten Sie für eine Ausgabe? Welche Statusflags werden sicherlich gesetzt?*

Lösungsvorschlag:

Ergebnis der Multiplikation ist: -194313216
Das Overflowflag und das Negativflag werden gesetzt.

b) *Erweitern Sie das Programm um eine Ausgabe der Statusflags. Die Befehle (Set-Instructions) zur Auswertung der Statusflags sind z. B. in der Befehlsreferenz von Intel[1], Seite 416 ff. zu finden. Zur Konvertierung des 8-Bit Wertes auf ein 32-Bit Wert können Sie den folgende Befehl verwenden: movzbl %al,%eax.*

Lösungsvorschlag:

```
.data
 intout: .string "Overflow␣%d␣\n"
 a:      .long  300000
 b:      .long  300000

.text
.globl main
main:
     movl   a,%eax
     movl   b,%ebx
     imull  %eax,%ebx
     seto   %dl   # seto - Overflowflag
                  # sets - Signflag
     movzbl %dl,%edx

.ende:   pushl %edx
```

[1] vgl. SVN-Server, Material ⇒ Referenz ⇒ Intel ⇒ 253667.pdf

```
            pushl $intout
            call printf

    # Exit
    movl $1, %eax
    movl $75, %ebx  # ERROR CODE = 75 - Overflow
    int $0x80
```

c) *Unter Unix können Programme sogenannte Error-Codes übergeben. Dazu wird in das Register %ebx eine Nummer geschrieben, die einem Error-Code entspricht. Der Exit-Teil der IA32-Assemblerprogramme wird wie folgt ergänzt.*

```
...
# Exit
movl $1, %eax
movl $0, %ebx  # ERROR CODE = 0, keine Fehler
int $0x80
```

Erweitern Sie Ihr Programm um die Übergabe des Error-Codes (für den Overflow ist das Nummer 75) unter Verwendung des Registers %ebx. Den letzten Error-Code kann man auf der Unix-Shell mittels **echo $?** *abfragen.*

Lösungsvorschlag:

Listing, s. oben

Hausaufgabe 1: Unterprogramm zum Sortieren (5 Punkte)

*Gegeben sei wieder das folgende Java-Programm (vgl. Übung 2). 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));
    }

}
```

Das Sortierprogramm soll als Unterprogramm in IA32-Assembler realisiert werden. Das zu sortierende Feld und die Länge des Feldes (stehen beide im .data-Teil, als Datentyp soll long verwendet werden) werden durch eine Referenzübergabe (call by reference) an das Unterprogramm übergeben. Das Unterprogramm sortiert die Zahlenfolge und schreibt das Ergebnis in den Speicher. Es werden keine Werte zurückgegeben.

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)*

a) *Schreiben Sie das IA32-Assemblerprogramm. Schreiben Sie auch das Hauptprogramm mit dem Unterprogrammaufruf. Kommentieren Sie Ihren Code.*

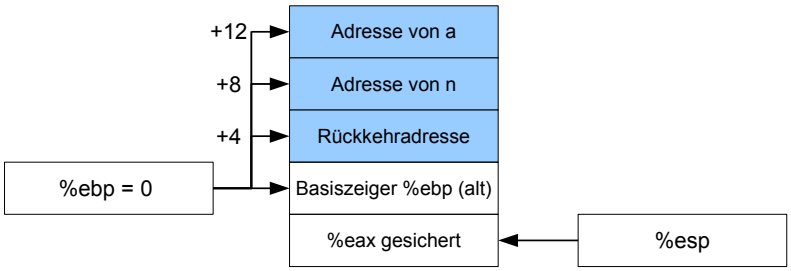
Lösungsvorschlag:

```
.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
sort:
    # Zustand sichern
    pushl %ebp
    movl %esp, %ebp
    pushl %eax
    pushl %ebx
    pushl %ecx
    pushl %edi
    # Argumente vom Stack holen
    movl 12(%ebp), %ebx
    movl 8(%ebp), %edi
    # Wirkliche Arraygroesse laden
    movl (%edi), %ecx
    decl %ecx

fori:
    # for i = n - 1; i > 0; i--
    movl $0, %edi

forj:
    # for j = 0; j < i; j++
    # if (a[j] > a[j+1])
    movl (%ebx, %edi, 4), %eax
    cmpl 4(%ebx, %edi, 4), %eax
    jle endif
    # then
    xchgl %eax, 4(%ebx, %edi, 4)
    movl %eax, (%ebx, %edi, 4)

endif:
    incl %edi
    cmpl %ecx, %edi
    jl forj

    loop fori

    # Zustand wiederherstellen
    popl %edi
    popl %ecx
    popl %ebx
```

```
    popl %eax
    popl %ebp
    # Ruecksprung
    ret

main:
    # Argumente fuer sort() pushen
    leal a, %edi
    pushl %edi
    leal n, %edi
    pushl %edi
    call sort
    addl $8, %esp

    # 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
```

b) *Skizziern Sie den Aufbau des Stacks nach dem Aufruf des Unterprogramms.*

Lösungsvorschlag:



Hausaufgabe 2: Ägyptisches Potenzieren (5 Punkte)

Das im folgenden beschriebene Verfahren wird u. a. auch beim RSA-Verschlüsselungsverfahren verwendet. Ähnlich dem Ägyptischen Multiplizieren existiert ein Algorithmus zum leichten Potenzieren natürlicher Zahlen, der als Ägyptisches Potenzieren bezeichnet wird. Die Vorschrift zur Berechnung von $b^e \bmod m$*, mit* b *als Basis,* e *als Exponent und modulo* m *lautet wie folgt:*

Der Exponent wird immer halbiert (unter Wegwerfen eines ggf. anfallenden Restes), die Basis stets quadriert (modulo m). Aufmultipliziert werden sogleich oder schlußendlich diejenigen Basen (modulo m), bei denen in der Exponentenhalbierung ein Rest weggeworfen wurde.

Beispiel: $b = 62$, $e = 17$ *und* $m = 77$

Exponent	*Basis*	*relevant?*
17	62	•
8	$3844 \bmod 77 = 71 \bmod 77 = -6$	
4	36	
2	$1296 \bmod 77 = 64 \bmod 77 = -13$	
1	$169 \bmod 77 = 15$	•

Ergebnis: $62 \cdot 15$ *mod* $77 = 930$ *mod* $77 = 6$

a) *Berechnen Sie mittels Ägyptischen Potenzierens* 5^7 *mod* 77.

Lösungsvorschlag:

Exponent	Basis	relevant?
7	5	•
3	25	•
1	$625 \bmod 77 = 9$	•

Ergebnis: $5 \cdot 25 \cdot 9 \bmod 77 = 1125 \bmod 77 = 47$

b) *Schreiben Sie nun eine Funktion* `aegypt_pot` *in IA32-Assembler, die folgende Schnittstelle besitzt:*
`int aegypt_pot(int b, int n, int m)`
Diese Funktion berechnet die n-te Potenz von b (modulo m) mittels Ägyptischen Potenzierens wie oben beschrieben. Die Parameter werden als Werte übergeben. Das Ergebnis wird im Prozessorregister %eax an das Hauptprogramm zurückgegeben. Schreiben Sie auch das Hauptprogramm mit dem Unterprogrammaufruf. Kommentieren Sie Ihren Code.

Lösungsvorschlag:

```
.data
 intout: .string "Ergebnis␣%d␣\n"
 x:      .long 62
 n:      .long 17
 m:      .long 77
 erg:    .long 0
.text
.globl main

aegypt_pot:
    pushl %ebp              # Base-Pointer sichern
    movl %esp, %ebp         # Stack -> Base
    pushl %ebx              # Register retten
    pushl %ecx
    pushl %edx
    pushl %edi
    pushl %esi
    movl 16(%ebp), %ebx     # Basis in ebx
    movl 12(%ebp), %ecx     # Exponent in ecx
    movl 8(%ebp), %edi      # m in edi
    movl $1, %esi           # hilf := 1
schl:
    cmpl $1, %ecx           # while (e>=1)
    jl   funend
    shrl $1, %ecx           # if odd(e)
```

```
    jnc  m1                 # e := e div 2
    movl %esi, %eax
    mull %ebx               # hilf := hilf * b
    movl %eax, %esi
m1:
    movl %ebx, %eax
    mull %ebx
    movl %eax, %ebx
    cmpl %edi, %ebx         # if b>m
    jle  schl
    cmpl $1, %ecx           # if e=1
    jne  m2
    movl %ebx, %eax
    subl %edx, %edx
    divl %edi               # b := b mod m
    movl %edx, %ebx
    jmp  schl
m2:
    movl %ebx, %eax         # else
    subl %edx, %edx
    divl %edi               # b := (b mod m) - m
    subl %edi, %edx
    movl %edx, %ebx
    jmp  schl
funend:
    movl %esi, %eax
    subl %edx, %edx         # Produkt durch m teilen
    divl %edi
    movl %edx, %eax

    popl %esi               # Register wieder herstellen
    popl %edi
    popl %edx
    popl %ecx
    popl %ebx
    popl %ebp
    ret                     # Return in Hauptprogramm

main:
    pushl x
    pushl n
    pushl m
    call aegypt_pot
    addl $12, %esp
    movl %eax, erg

.ende:
    pushl %eax              # Wert im %eax ausgeben
    pushl $intout
    call printf

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