Grundlagen der Informatik III

Wintersemester 2010/2011

Wolfgang Heenes, Patrik Schmittat

TECHNISCHE UNIVERSITÄT DARMSTADT

3. Aufgabenblatt

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?

b) Veranschaulichen Sie sich das Ergebnis in einer Grafik.

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.

b) Wie viele gültige Seriennummern gibt es?

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

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.

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?

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.

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.

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)

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

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

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

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: $62 \cdot 15 \bmod 77 = 930 \bmod 77 = 6$

a) Berechnen Sie mittels Ägyptischen Potenzierens $5^7 \bmod 77$.

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.