
Prüfung

Prüfungsfach: Systemnahe Programmierung
Datum/Uhrzeit: 5. Juli 2019 / 10:30 Uhr
Raum: J3.19
Prüfer: Dr. Hubert Högl
Dauer: **60** Minuten
Hilfsmittel: keine

Hinweise:

1. Dieses Angabenblatt hat auch eine **Rückseite!** Bitte sofort überprüfen.
2. **Schreiben** Sie bitte **nicht** auf das **Angabenblatt**. Verwenden Sie für Ihre Antworten die separat ausgeteilten Bögen. **Die Angaben dürfen Sie behalten.**
3. Schreiben Sie **nicht mit Bleistift**.

Viel Glück!

Aufgabe 1 (6 Punkte)

Bei den folgenden Aufgaben ist immer genau eine der Auswahlmöglichkeiten korrekt:

(a) Welche effektive Adresse wird bei der folgenden Adressierung verwendet:

```
movl 16(%ecx, %edx, 4), %eax    # ecx = 0xf00, edx = 0x10
```

Auswahl: a1 0xf80 a2 0xf416 a3 0xf50

(b) Wie liegen bei Little Endian die einzelnen Bytes des folgenden Wortes (4 Byte) von niedriger zu hoher Adresse im Speicher: 0x12345678

Auswahl: a1 12 34 56 78 a2 78 56 34 12 a3 87 65 43 21

(c) Ab welchem Label startet ein Programm, nachdem es vom Betriebssystem in den Speicher geladen wurde?

Auswahl: a1 main a2 _text a3 _start

(d) Das `esp` Register zeigt auf den Stack. Sie möchten nun mit dem GDB die obersten fünf Werte auf dem Stack in 32-Bit Breite ausgeben.

Auswahl: a1 x/5xw \$esp a2 x/5xw %esp a3 p/5s \$esp

(e) Bei einem Systemaufruf werden die Argumente übergeben

Auswahl: a1 auf dem Stack a2 auf dem Heap a3 über Register

(f) Was dürfen rekursive Funktionen auf keinen Fall haben?

Auswahl: a1 globale Variablen a2 lokale Variablen a3 einen Returnwert

Aufgabe 2 (8 Punkte)

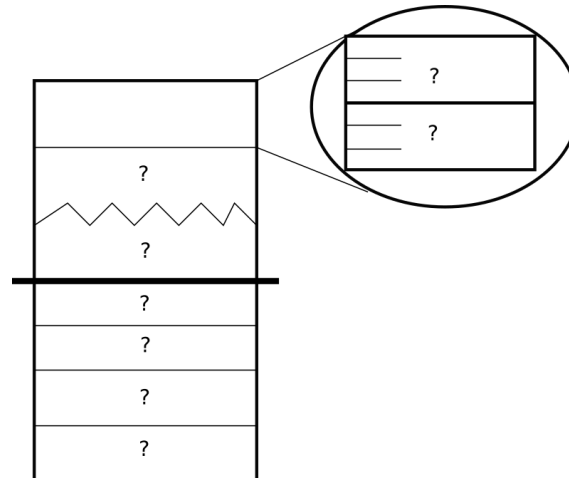
Zeichnen Sie ein Diagramm, das die wesentlichen Teile von CPU und Speicher enthält, so dass Sie den **fetch – decode – execute** Zyklus an Hand der Ausführung der folgenden Maschinenbefehle

```
movl $<adr>, %eax
movl (%eax), %ebx
incl %ebx
```

erklären können. Denken Sie auch an die Register **MAR**, **MDR** und **IR**.

Aufgabe 3 (8 Punkte)

Benennen Sie die verschiedenen Abschnitte in dem folgendem Abbild eines Prozesses:



Aufgabe 4 (6 Punkte)

Sie finden eine ausführbare Datei `main` die 236 Byte gross ist. Da Sie nicht wissen, was das Programm macht, geben Sie das Kommando `objdump -D main` ein, das folgende Ausgabe macht:

Disassembly of section `.text`:

```
08048054 <.text>:
8048054:  b8 01 00 00 00      mov    $0x1,%eax
8048059:  bb 0c 00 00 00      mov    $0xc,%ebx
804805e:  cd 80               int    $0x80
```

Nun ist alles klar!

(a) Was bedeutet `section .text`? (1 Punkt)

- (b) Was bedeuten die verschiedenen Spalten der Ausgabe? (3 Punkte)
- (c) Was macht das Programm? (2 Punkte)

Aufgabe 5 (10 Punkte)

- (a) Welche Vorteile bietet eine 64-bit Architektur? (4 Punkte)
- (b) Welche General-Purpose Register hat der x64-64? (ohne FPU- und XMM Register). (4 Punkte)
- (c) Was hat sich bei den **system calls** im Vergleich zu 32-Bit verändert? (2 Punkte).

Aufgabe 6 (6 Punkte)

Was versteht man unter **robusten Programmen**, so wie es im Kapitel 7 im Buch von Bartlett steht? Beantworten Sie die folgenden Punkte:

- a) Warum ist das **Testen** eines Programmes wichtig?
- b) Was ist **error handling**?
- c) Was wird im Programm `add_year()` (Kap. 7) gemacht, um es robuster zu machen?

Aufgabe 7 (18 Punkte)

Punkte: (a) 4, (b) 2, (c) 2, (d) 2, (e) 2, (f) 4, (g) 2

Das folgende Programm `main.s` kann mit einem Kommandozeilenargument aufgerufen werden, z.B. so: `main 5`.

```
1      .section .data
2      .section .text
3      .globl _start
4
5      _start:
6          movl    8(%esp), %edx
7          movb    (%edx), %al
8          subb    $0x30, %al
9
10         movl    $0, %ecx
11         movl    $1, %ebx
12 L1:    decb    %al
13         addl    %ebx, %ecx
14         addl    $1, %ebx
15         cmpb    $0, %al
16         jne     L1
```

```

17
18     movl    %ecx, %ebx
19     movl    $1, %eax
20     int     $0x80

```

- Was macht das Programm? Kommentieren Sie die drei Abschnitte 6 - 8, 10 - 16 und 18 - 20. Bitte keine trivialen Kommentare verwenden!
- Was wird ziemlich sicher passieren, wenn man das Programm ohne Kommandozeilenargument aufruft?
- Wie kann man den Fall (b) abfangen, d.h. wie kann man das Programm robuster machen? Beschreiben Sie knapp die Lösung, es ist kein Quelltext erforderlich.
- Was wird passieren, wenn man das Programm mit einer mehrstelligen Zahl aufruft, z.B. `main 12`?
- Was muss man tun, damit der Fall (d) funktioniert (kurze Beschreibung ohne Quelltext).
- Schreiben Sie statt des Abschnitts 10 – 16 eine Funktion `fkt()` mit Prolog und Epilog, die man so aufruft:

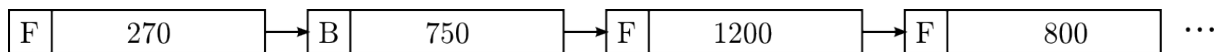
```
s = fkt(int n);
```

Das Argument `n` ist die Zahl von der Kommandozeile, der Rückgabewert ist eine ganze Zahl in `eax`.

- Statt das Ergebnis über den Exitcode zu erhalten, möchten Sie die Funktion `printf()` aus der C-Bibliothek aufrufen. Schreiben Sie den nötigen Assembler-Quelltext hin.

Aufgabe 8 (10 Punkte)

Die folgende Abbildung zeigt eine **Kette aus freien (F) und belegten (B) Blöcken auf dem Heap**. Die Zahl gibt die Grösse des Blockes an.



- Wo liegt der Heap-Speicher** im Speicher eines Prozesses?
- Welcher Block** wird bei einem `allocate(600)` Aufruf nach dem Algorithmus aus dem Buch von Bartlett belegt?
- Welche Nachteile** hat dieser einfache Algorithmus?
- Wie kann der einfache Algorithmus **verbessert** werden?
- Wie kann der gesamte, dem Heap zur Verfügung stehende Speicher **vergrößert** werden?

Ende der Prüfung
