Hochschule Augsburg
Fakultät für Informatik

Sommersemester 2014

Klausur

Prüfungsfach:	Systemnahe Programmierung
Datum/Uhrzeit:	9. Juli 2014 / 13:30 Uhr
Raum:	wie angeküundigt
Prüfer:	Dr. Hubert Högl
Dauer:	**60** Minuten
Hilfsmittel:	keine

Hinweise:

1. Dieses Angabenblatt hat auch eine Rückseite.
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.

Aufgabe 1 (12 Punkte)

Geben Sie fr jede der Sprachen **Assembler, C und Python** Antworten auf die folgenden Fragen:

a) Ist die Sprache **portabel**?

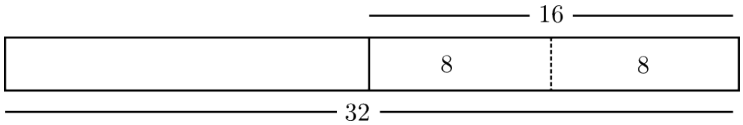
b) Falls portabel, mit **welcher Technik** wird die Portabilitt erreicht?

c) Gibt es den Begriff der **Adresse** in der Sprache?

d) Eine Zeile der Sprache entspricht in etwa **wie vielen Maschinenbefehlen**?

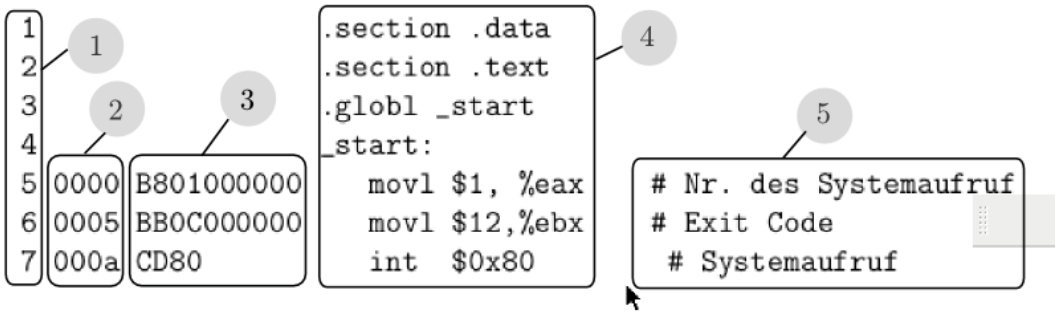
Aufgabe 2 (8 Punkte)

Schreiben Sie die wichtigsten **vier allgemeinen Register** und **vier speziellen Register** des x86 hin. Benennen Sie jeweils wie in der folgenden Abbildung das gesamte Register, den unteren 16-Bit Teil und die beiden unteren 8-Bit Teile (bitte auf den Antwortbogen bertragen).



Aufgabe 3 (10 Punkte)

Erlutern Sie kurz die **fnf Bestandteile** 1 bis 5 in der folgenden Abbildung eines **Assembler Listings**:



Aufgabe 4 (4 Punkte)

Erlutern Sie, warum **unterschiedliche Prozesse** auf Ihrem Rechner immer an den **gleichen Adressen** liegen. Wenn man sich die **physikalischen Adressen** ansieht, dann kann eine Speicherzelle im Speicher nur von einem Prozess genutzt werden, unterschiedliche Prozesse mssen also an verschiedenen physikalischen Adressen liegen. Wie funktioniert das?

Aufgabe 5 (6 Punkte)

Welche **drei Mglichkeiten** gibt es, um in Assembler den **Array Datentyp** zu realisieren? Ein Array sei eine Folge einer bestimmten Anzahl von Elementen mit gleichem Typ. Verdeutlichen Sie jede Variante mit einer kleinen Skizze. Schreiben Sie fr jede Variante den **Assembler-Code fr die Iteration** ber alle Elemente.

```
        # Code noch unvollstaendig!
array:  .long   3, 9, 4, 8
```

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 (6 Punkte)

In **welche Abschnitte** ist jedes Programm im Speicher gegliedert und wozu dienen diese Abschnitte? Zeichnen Sie ein **Bild des gesamten Linux Programmes** beim Start.

Aufgabe 8 (4 Punkte)

Obwohl Sie den Systemaufruf `poll()` wahrscheinlich nicht kennen, knnen Sie seinen Aufruf in Assembler sicher skizzieren. Dieser Aufruf hat die Nummer 168.

```
int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

In Ihrer Lsung nennen Sie die Parameter einfach `fds`, `nfds` und `timeout`. Wohin mssen diese Parameter bergeben werden?

Aufgabe 9 (9 Punkte)

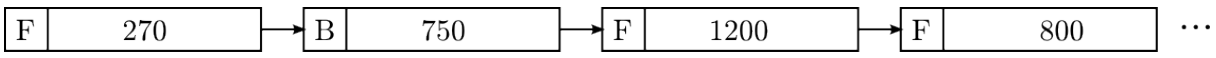
Sie erinnern sich noch an den Kurztest in diesem Semester. Fllen Sie die Funktion `timespow2()` aus.

```
#   Aufgabe: timespow(3, 2) + timespow(2, 3)

    .section .data
    .section .text
    .globl _start

_start:
    pushl $2        # b
```

```
    pushl $3        # x
    call  timespow2
    addl  $8, %esp
    pushl %eax
    pushl $3        # b
    pushl $2        # x
    call  timespow2
    addl  $8, %esp
    popl  %ebx
    addl %eax, %ebx
    movl $1, %eax
    int $0x80

# timespow2(x, b)
#   return x * 2^b
#   Trick: x * 2^b = shift argument x left by b bits
#                    shll %cl, %ebx  (shift ebx left by cl bits)
.type timespow2, @function
timespow2:
    ..................... # 1 Prolog
    .....................# 2 Prolog
    ..................... # 3 Argument holen
    ..................... # 4 Argument holen
    ..................... # 5 Schieben
    ..................... # 6 Ergebnis ablegen
    ..................... # 7 Epilog
    ..................... # 8 Epilog
    ..................... # 9 Zurueckkehren
```

Aufgabe 10 (10 Punkte)

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



a) **Wo liegt der Heap-Speicher** im Speicher eines Prozesses?

b) **Welcher Block** wird bei einem `allocate(600)` Aufruf nach dem Algorithmus aus dem Buch von Bartlett belegt?

c) **Welche Nachteile** hat dieser einfache Algorithmus?

d) Wie kann der einfache Algorithmus **verbessert** werden?

e) Wie kann der gesamte, dem Heap zur Verfgung stehende Speicher **vergrssert** werden?

Ende der Prfung