

# Hinweise für die Klausur

*Assembler-Code-Schreibweise unbedeutend! Voraussetzung: ausreichend Kommentare!*

**Aber:**

- Code muss direkt umsetzbar sein
- Anzahl der Register beschränkt
- in sich konsistent

**Beispiel (OK)**

```
%accu = %accu + 1           add #1           addl $1, %eax
if (%accu == 0) then goto label  cmp #0           cmpl $0, %eax
                                beq label        je label
```

**Beispiel (Falsch!)**

```
if (x == 0 && y == 0) x = y + z
for (...) ...
```

# Aufgabe 1

**Compilieren Sie folgendes Unterprogramm:**

```
void proc(void) {
    struct rec {
        long int val;
        short int f[12];
    };
    struct rec x;
    register long int i;

    x.val = 10;
    for (i = 0; i < 12; i++) {
        x.f[i] = 0;
    }
    ...
}
```

# Aufgabe 1

## *Lösung:*

```
proc:  subl $28, %esp
       movl $10, (%esp)
       movl $0, %eax
       jmp test
loop:  movw $0, 4(%esp,%eax,2)
       incl %eax
test:  cmpl $12, %eax
       jl loop
       ...
       addl $28, %esp
       ret
```

# Aufgabe 2

*Angenommen, jeder Befehl braucht genau einen Takt für die Ausführung auf einem 1 GHz-Prozessor. Wie lange dauert es, die Prozedur auf diesem Prozessor auszuführen?*

## *Lösung:*

*Die Befehle `subl`, ..., `jmp` sowie `addl`, `ret` werden jeweils genau einmal ausgeführt => 6 Takte.*

*Der Schleifenrumpf (`movw`, `incl`) wird 12-mal durchlaufen => 24 Takte.*

*Der Vergleich (`cmpl`, `jl`) muss einmal mehr ausgeführt werden => 26 Takte.*

*Insgesamt: 56 Takte bzw. 56ns.*

## Aufgabe 3

*Der gcc produziert (mit mehreren Optimierungs-Hinweisen) folgenden Code:*

```
proc:  subl $28, %esp
      movl $10, (%esp)
      leal 4(%esp), %eax
      xorl %ebx, %ebx
      movl %ebx, (%eax)+
      movl %ebx, (%eax)+
      movl %ebx, (%eax)+
      movl %ebx, (%eax)+
      movl %ebx, (%eax)+
      movl %ebx, (%eax)+
      addl $28, %esp
      ret
```

*Begründen Sie, warum der Code korrekt, kürzer und schneller ist!*

## Aufgabe 3

*Lösung:*

*Der gcc wandelt („loop-unrolling“)*

```
for (i = 0; i < 12; i++) x.f[i] = 0;
```

*um in*

```
x.f[0] = 0; x.f[1] = 0; ...; x.f[11] = 0;
```

*Das ergibt direkt kompiliert:*

```
movw $0, 4(%esp)
movw $0, 6(%esp)
...
movw $0, 26(%esp)
```

*Äquivalent zu*

```
movw $0, 4(%esp)
movw $0, 6(%esp)
```

*ist*

```
movl $0, 4(%esp)
```

*Statt*

```
movl $0, ...  
movl $0, ...
```

*ist besser (kürzerer Code)*

```
movl $0, %ebx (oder noch kürzer xorl %ebx, %ebx)  
movl %ebx, ...
```

*Damit*

```
xorl %ebx, %ebx  
movl %ebx, 4(%esp)  
movl %ebx, 8(%esp)  
...  
movl %ebx, 24(%esp)
```

*Aufsteigende, kontinuierliche Adressen => Lösung wie oben (kürzerer Code)*

## Aufgabe 4

*Welchem C-, C++ bzw. Java-Code könnte der folgende Assembler-Code entsprechen?*

```
movl $0, %ebx  
movl $0, %eax  
jmp labelB  
labelA: movl $1234, %ecx  
movl (%ecx, %eax, 4), %ecx  
shrl %ecx  
addl %ecx, %ebx  
movl %eax, %ecx  
imull %ecx, %ecx  
shrl %ecx  
addl %ecx, %ebx  
incl %eax  
labelB: cmpl $1024, %eax  
jl labelA
```

## Aufgabe 4

**Lösung:**

```
long int array[1024];    /* Adresse 1234 */
register long int i;     /* Register %eax */
register long int sum;   /* Register %ebx */

sum = 0;
for (i = 0; i < 1024; i++) {
    sum = sum + array[i] / 2 + i*i / 2;
}
```

## Aufgabe 5

**Erweitern Sie die aus der Vorlesung bekannte CPU-Struktur, damit diese in der Lage ist, einen "jsr"-Befehl abzuarbeiten!**

**Lösung:**

**Parallel zum PC muss entsprechend ein SP-Register eingeführt werden. Der Wert des SP muss auf den Datenbus (um ihn zu inkrementieren/dekrementieren) und auf den Adressbus gelegt werden können (um von der Adresse zu lesen/an die Adresse zu schreiben).**

## Aufgabe 6

*Angenommen das Tor vom SP zum Datenbus hat die Nummer 14, das Tor zum Adressbus die Nummer 15 und das Übernahme-Signal für das SP-Register die Nummer 16. Schreiben Sie die für einen "rts"-Befehl notwendigen Steuerwörter!*

**Lösung:**

```
...
RTS  --   6   |   7   -   R, 15           Holen Rückkehradresse
RTS  --   7   |   8   -   R, 10, 15
RTS  --   8   |   9   -   14           Inkrementieren des SP
RTS  --   9   |  10   -   6, 14
RTS  --  10   |  11   Inc  9
RTS  --  11   |   0   Inc  9, 16
```

## Aufgabe 7

*Stellen Sie Vermutungen an, wieviele Bytes die Befehle*

```
addl $12345678, %eax
jmp  label
movl $-1, 34567890(%eax)
xorl %eax, %eax
```

*(i80x86) lang sind.*

**Lösung:**

```
addl $12345678, %eax           5 Byte
jmp  label                     5 Byte
movl $-1, 34567890(%eax)      10 Byte
xorl %eax, %eax               2 Byte
```

## Aufgabe 8

*Linux auf Intel-Prozessoren ruft Betriebssystem-Funktionen über eine Exception "int 0x80" auf. Beispiel (leicht modifiziert):*

```
write:    .globl write
          movl 12(%esp), %edx /* Länge des Puffers */
          movl 8(%esp), %ecx /* Puffer-Adresse */
          movl 4(%esp), %ebx /* Datei-Nummer */
          movl $4, %eax      /* 4: write-System-Call */
          int 0x80          /* System-Call */
          ret
```

*Wie sieht der entsprechende System-Call-Handler im Betriebssystem-Kern aus, der bei einem solchen Aufruf die sys\_write-Funktion aufrufen soll?*

## Aufgabe 8

*Lösung*

```
syscall:
    ...
    cmpl $4, %eax
    jne not_write
    pushl %edx
    pushl %ecx
    pushl %ebx
    call sys_write
    addl $12, %esp
    jmp done
not_write:...
    ...
done:    iret
```

## Aufgabe 9

Überlegen Sie sich eine Schaltung, die eine Shift-Operation durchführen kann. Es soll nicht nur um ein Bit sondern um  $N$  Bits geschoben werden können ( $N \in \{0, \dots, 7\}$ ).

Lösung:

