FAKULTÄT FÜR INFORMATIK
TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Rechnertechnik und Rechnerorganisation
Prof. Dr. Martin Schulz

Einführung in die Rechnerarchitektur

Wintersemester 2017/2018

Tutorübung 5 - Lösungsvorschläge

20.11–24.11.2017

---

1. (a)

```
        cmp al, 5
        jge marke1

        cmp al, 4
        jg marke1

        sub al, 5       ; Vorsicht: al ist danach verändert!
        jge marke1

        mov bl, 5
        cmp al, bl
        jge marke1

        cmp al, 5
        jg marke1
        je marke1

        cmp al, 5
        jl marke2       ; Umkehrung des Vergleichs
        jmp marke1
marke2:
        ...
```

(b)

```
int ebx;
ebx = 50;             // <--- Schleifenzählerinitialisierung
while (ebx <= 60) {   // <--- Schleifenbedingung
        fkt(ebx);         // <-+- Schleifenkörper
        ebx = ebx + 1;    // <-+   <--- Zählerfortschaltung
}
```

Übersetzung in Assembler:

```
        mov ebx, 50
        jmp cond

body:
        mov eax, ebx
        call fkt
        add ebx, 1                ; zählerfortschaltung

cond:
        cmp ebx, 60
        jle body
```

Der Befehl „`jmp cond`“ kann nur dann weggelassen werden, wenn bereits bei der Übesetzung klar ist, dass der Schleifenkörper mindestens einmal durchlaufen wird. Bei der Iteration von 50 bis 60 ist das der Fall.

(c)
```
        mov eax, 100
loop:
        call fkt
        dec eax       ; setzt auch schon die flags...
        cmp eax, -1   ; kann man weglassen, wenn man mit jl springt
        jne loop
```

(d)
```
        mov eax, 0
outer_loop:
        mov ebx, 0
inner_loop:
        ...
        inc ebx
        cmp ebx, eax
        jle inner_loop
        inc eax
        cmp eax, 101
        jne outer_loop
```

2.
```
strcpy:
        mov al, [esi]
        mov [edi], al
        inc esi
        inc edi
        cmp al, 0
        jne strcpy
        ret
```

3. Bitoperationen

(a)
```
        mov ebx, eax
        shr ebx, 3
        and ebx, 0xf
```

(b)
```
push eax
and eax, 0x7
shl eax, 5
and ebx, 0xffffff1f
or ebx, eax
pop eax
```

(c)
```
and eax, 0x7800
cmp eax, 0x4800
je irgendwohin
```

(d)
```
sar eax, 4
```

(e)
```
and eax, 63
```

(f)
```
mov cl, bl
mov ebx, 1
shl ebx, cl
not ebx
and eax, ebx
```

4. (a) Dank $a \cdot b = b \cdot a$ kann man die Fakultät auch „rückwärts“ mit Hilfe einer absteigenden Schleife (mit kleiner werdendem Zähler) berechnen. Das Programm bräuchte noch ein paar Fehlererkennungen (z. B. $n < 0$). Optimiert ist es auch nicht, ein `mul` könnte man sich sparen.

```
fak_mit_schleife:
        mov ecx, eax  ; mul braucht eax, daher Zähler in ecx
        mov eax, 1    ; erster multiplikand, damit wir mit n*1
   anfangen
        cmp ecx, 0    ; sonderfall für 0!=1
        je fak_fertig
fak_loop:
        mul ecx       ; eax := eax * ecx
        dec ecx       ; ecx := ecx - 1
        jne fak_loop  ; Schleife solange ecx !=1 (bzw. ecx-1 != 0)
fak_fertig:
        ret           ; mul-ergebnis schon in eax, nichts weiter nö
   tig
```

(b)
```
;Parameter n in eax, Ergebnis n! in eax
fakr:
        cmp eax, 0    ; Rekursionsende
        je fakr_fertig
        push eax      ; sichere n
        sub eax, 1
        call fakr     ; danach (n-1)! in eax
        pop ebx       ; hole n
        mul ebx       ; (n-1)!*n
        ret
fakr_fertig:
        mov eax, 1
        ret
```

Die Schleifenvariante ist noch etwas kürzer als die Rekursion und benötigt zur Laufzeit keinen Stack zur Speicherung der Zwischenergebnisse.