DS – Übung 8

U 8.1)

Das C-Programm

```
#include <stdio.h>

int main()
{
    long n = 101;
    do{
        while (!(--n % 3));
        if (!(n % 5))
            printf("5 divides, but 3 does not divide %3ld\n", n);
    }
    while (n > 0);
    return 0;
}
```

erzeugt folgende Ausgabe:

```
5 divides, but 3 does not divide 100
5 divides, but 3 does not divide  95
5 divides, but 3 does not divide  85
5 divides, but 3 does not divide  80
5 divides, but 3 does not divide  70
5 divides, but 3 does not divide  65
5 divides, but 3 does not divide  55
5 divides, but 3 does not divide  50
5 divides, but 3 does not divide  40
5 divides, but 3 does not divide  35
5 divides, but 3 does not divide  25
5 divides, but 3 does not divide  20
5 divides, but 3 does not divide  10
5 divides, but 3 does not divide   5
```

Übersetzen Sie es in GNU-Assembler und beachten dabei folgende Punkte:

a) Verwenden Sie sowohl die **do while**-Schleife, die **while**-Schleife (beachten Sie das Semikolon hinter der **while**-Schleife) als auch die **if**-Abfrage wie angegeben.

b) Verwenden Sie mindestens je einmal **test** und **cmp**.

c) Achten Sie darauf, dass Sie keine Berechnung 0/0 erzeugen.

d) Vergeben Sie für die Labels aussagekräftige Namen (kein .Lx).

e) Die Ausgabe soll der oben angegebenen entsprechen.

Übung 8.2)

Übersetzen Sie u.a. das GNU-Assembler-Programm in C. Verwenden Sie dabei für die Ein- und Ausgaben **scanf()** und **printf()**.

```
.section .rodata
.align 8
jumptable:
  .quad default
  .quad case_1
  .quad case_2
  .quad default
  .quad case_4
  .quad case_5
  .quad default

in_text:
  .asciz "Enter number in [0, 5]: "
out_text:
  .asciz "x = "
enter:
  .byte '\n'

#####################################

.section .bss
.lcomm n, 1
.lcomm dummy, 1

#####################################

.section .data
x:    .int 10
i:    .int 1

#####################################

.section .text
.globl _start

_start:
  pushq %rbp
  movq %rsp, %rbp

  jmp testFor
for:
  movq $1, %rax
  movq $1, %rdi    #stdout
  movq $in_text, %rsi
  movq $24, %rdx
  syscall          #SYS_WRITE

  movq $0, %rax
  movq $0, %rdi    #stdin
  movq $n, %rsi
  movq $1, %rdx
  syscall          #SYS_READ



  movq $0, %rax
  movq $0, %rdi    #stdin
  movq $dummy, %rsi
  movq $1, %rdx
  syscall          #SYS_READ

  subb $48, n
  movb n, %r8b
  movzbl %r8b, %r8d

switch:
  cmpl $5, %r8d
  ja switch_end
  jmp *jumptable(, %r8, 8)
case_1:
  incl x
case_2:
  movl x, %eax
  movq $7, %r9
  mulq %r9
  movl %eax, x
  jmp switch_end
case_4:
  addl $3, x
  jmp switch_end
case_5:
  addl %r8d, x
  jmp switch_end
default:
  movl $0, x

switch_end:
  incl i

testFor:
  cmpl $3, i
  jle for

  movq $1, %rax
  movq $1, %rdi    #stdout
  movq $out_text, %rsi
  movq $4, %rdx
  syscall          #SYS_WRITE

  movl x, %eax
  cqto
  movq $10, %rdi
  idivq %rdi #x % 10

  addq $48, %rdx
  movl %edx, x
  movq $1, %rax
  movq $1, %rdi    #stdout
  movq $x, %rsi
  movq $1, %rdx
  syscall          #SYS_WRITE

  movq $1, %rax
  movq $1, %rdi
  movq $enter, %rsi
  movq $1, %rdx
  syscall          #SYS_WRITE

  movq $60, %rax
  xorq %rdi, %rdi
  popq %rbp
  syscall          #SYS_EXIT
```