multDiv.s

```
.section .data
  format1:
    .asciz "-3 * 4 = %d\n"
  format2:
    .asciz "dec(10e7 * 20e7) = hex(%lX%lX)\n"
  format3:
    .asciz " 17 /  3 = %2d r %2d\n"
  format4:
    .asciz "-17 /  3 = %2d r %2d\n"
  format5:
    .asciz " 17 / -3 = %2d r %2d\n"
  format6:
    .asciz "-17 / -3 = %2d r %2d\n"

.section .text
.globl main
.type main, @function
main:
  pushq %rbp
  movq %rsp, %rbp


  # Examples for multiplication
  # ===========================

  # -3 * 4
  movq $-3, %rbx
  # 2 args only possible using imulq
  imulq $4, %rbx # result in rbx
  # print result
  movq $format1, %rdi
  movq %rbx, %rsi
  movq $0, %rax
  call printf

  # 10'000'000 * 20'000'000
  movq $10000000, %rbx
  movq $20000000, %rax
  # mulq only possible using 1 arg
  mulq %rbx # result in rdx|rax
  # print result
  movq $format2, %rdi
  movq %rdx, %rsi
  movq %rax, %rdx
  movq $0, %rax
  call printf

  # Examples for division
  # =====================

  # Usage of divq with 2 args
  # 17 / 3
  movq $17, %rax
  movq $0, %rdx  # divq is for unsigned only
  movq $3, %rbx
```

```
divq %rbx, %rax # 2nd arg (dividend) must be rax
                # quotient in rax, remainder in rdx
movq $format3, %rdi
movq %rax, %rsi
# remainder is already in rdx
movq $0, %rax
call printf

# same Example with 1 arg
# 17 / 3
movq $17, %rax
cqto      # sign-extends rax to rdx|rax
movq $3, %rbx
divq %rbx # quotient in rax, remainder in rdx
movq $format3, %rdi
movq %rax, %rsi
# remainder is already in rdx
movq $0, %rax
call printf

# idivq only possible with 1 arg
# -17 / 3
movq $-17, %rax
cqto       # sign-extends rax to rdx|rax
movq $3, %rbx
idivq %rbx # quotient in rax, remainder in rdx
movq $format4, %rdi
movq %rax, %rsi
# remainder is already in rdx
movq $0, %rax
call printf

# 17 / -3
movq $17, %rax
cqto       # sign-extends rax to rdx|rax
movq $-3, %rbx
idivq %rbx # quotient in rax, remainder in rdx
movq $format5, %rdi
movq %rax, %rsi
# remainder is already in rdx
movq $0, %rax
call printf

# -17 / -3
movq $-17, %rax
cqto       # sign-extends rax to rdx|rax
movq $-3, %rbx
idivq %rbx # quotient in rax, remainder in rdx
movq $format6, %rdi
movq %rax, %rsi
# remainder already in rdx
movq $0, %rax
call printf

# exit main
movq $0, %rax
popq %rbp
ret
```