Handling Multiple Arguments in Assembly Language

```
.section .rodata
out:
  .asciz "result = %d\n"

.section .text

------------------------------------------------------------------------
                                  main
------------------------------------------------------------------------

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

  call call_proc
  movq $out, %rdi
  movq %rax, %rsi
  call printf

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

------------------------------------------------------------------------
                               call_proc
------------------------------------------------------------------------

# function call_proc()
.globl call_proc
.type call_proc, @function
call_proc:
  subq $32, %rsp      # Allocate 32-byte
  movq $1, 24(%rsp)   # Store 1 in &x1
  movl $2, 20(%rsp)   # Store 2 in &x2
  movw $3, 18(%rsp)   # Store 3 in &x3
  movb $4, 17(%rsp)   # Store 4 in &x4
  leaq 17(%rsp), %rax # Create &x4
  movq %rax, 8(%rsp)  # Store &x4 as arg8
  movb $4, (%rsp)     # Store 4 as arg7
  leaq 18(%rsp), %r9  # Pass &x3 as arg6
  movw $3, %r8w       # Pass 3 as arg5
  leaq 20(%rsp), %rcx # Pass &x2 as arg4
  movl $2, %edx       # Pass 2 as arg3
  leaq 24(%rsp), %rsi # Pass &x1 as arg2
  movq $1, %rdi       # Pass 1 as arg1
  # Call another function: proc()
  call proc
```

```
  # Retrieve changes to memory
  # Get x2 and convert to long
  movslq 20(%rsp), %rdx
  # function call_proc() continued

  # Compute x1 + x2
  addq 24(%rsp), %rdx
  # Get x3 and convert to int
  movswl 18(%rsp), %eax
  # Get x4 and convert to int
  movsbl 17(%rsp), %ecx
  # Compute x3-x4
  subl %ecx, %eax
  # Convert to long
  cltq
  # Compute (x1+x2) * (x3-x4)
  imulq %rdx, %rax
  # Deallocate stack frame
  addq $32, %rsp
  # Return / End
  # of function call_proc()
  ret

-------------------------------------------------------------------------
                                  proc
-------------------------------------------------------------------------

.globl proc
.type proc, @function
# void proc(a1, a1p, a2, a2p, a3, a3p, a4, a4p)
# Arguments passed as follows:
#   a1  in %rdi    (64 bits)
#   a1p in %rsi    (64 bits)
#   a2  in %edx    (32 bits)
#   a2p in %rcx    (64 bits)
#   a3  in %r8w    (16 bits)
#   a3p in %r9     (64 bits)
#   a4  at %rsp+8  ( 8 bits)
#   a4p at %rsp+16 (64 bits)
proc:
  addq %rdi, (%rsi)   # *a1p += a1
  addl %edx, (%rcx)   # *a2p += a2
  addw %r8w, (%r9)    # *a3p += a3
  movq 16(%rsp), %rax # rax = a4p
  movb 8(%rsp), %dl   # dl = a4 (Teil von %rdx)
  addb %dl, (%rax)    # *a4p = dl
  ret
```