Higher Memory
Addresses

STACK DIAGRAM

Lower Memory
Addresses

REGISTERS DIAGRAM

%RAX

Return Value (caller-saved)

%RBP

(callee-saved)

%RDI

1st argument (caller-saved)

%R10

(caller-saved)

%RSI

2nd argument (caller-saved)

%R11

(caller-saved)

%RDX

3rd argument (caller-saved)

%R12

(callee-saved)

%RCX

4th argument (caller-saved)

%R13

(callee-saved)

%R8

5th argument (caller-saved)

%R14

(callee-saved)

%R9

6th argument (caller-saved)

%R15

(callee-saved)

%RBX

Callee-saved

SPECIAL REGISTERS

%RSP

Address of stack "top"
(lowest stack address)

%RIP

addr. of next instruction
(in text section)

COMMON INSTRUCTIONS

mov a, b – copy a into b
movs a, b – store sign-extended a into b
movz a, b – store zero-extended a into b
lea a, b – store address of memory addressing expression a in b

push a – push a onto stack
pop a – pop a value from the top of the stack into a

call target - push return address onto the stack and jump to target label/address
ret – pop return address from stack and jump there

add a, b – store sum a+b into b
sub a, b – store difference b-a into b
imul a, b – store signed product a*b into b
and a, b – store bitwise AND a&b into b
or a, b – store bitwise OR a|b into b
shl/sal a, b – store left shift b<<a into b
shr a, b - store logical right shift b<<a into b
sar a, b – store arithmetic right shift b<<a into b

cmp a, b – set condition codes based on difference b-a
test a, b – set condition codes based on bitwise AND a&b

jg – jump if greater than (zero)
je – jump if equal to (zero)
jne – jump if not equal to (zero)
jle – jump if less than or equal to (zero)
jmp target – jump to target

MEMORY ADDRESS SYNTAX

$D(R_b, R_i, S)$ => $Mem[Reg[R_b] + S*Reg[R_i] + D]$
S can only be 1, 2, 4, or 8

Remember that **lea** calculates an address but does not access the address.