MEMORY DIAGRAM

Higher Addresses

Lower 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

Address of next instruction

COMMON INSTRUCTIONS

Data movement:
mov a, b – copy a into b
movs a, b – copy sign-extended a into b
movz a, b – copy zero-extended a into b
lea a, b – save address of memory
addressing expression a into b

Stack:
push a – push a on stack
pop a – pop value from top of stack into a

Procedures:
call *target* - push return address on stack
and jump to *target*
ret – pop return address from stack
and jump to return address

Arithmetic
add a, b – save sum (a+b) into b
sub a, b – save difference (b-a) into b
imul a, b – save signed product (a*b) into b
and a, b – save bitwise AND (a&b) into b
or a, b – save bitwise OR (a|b) into b

Shift: save b shifted ____ by a into b
shl/sal a, b – to the left (b<<a)
shr a, b – logically to the right (b>>a)
sar a, b – arithmetically to the right (b>>a)

Compare/test: set condition codes/flags…
cmp a, b – based on difference (b-a)
test a, b – based on bitwise AND (a&b)

Conditional jump: jump to *target* if…
jg *target* – greater than (zero)
je *target* – equal to (zero)
jne *target* – not equal to (zero)
jle *target* – less than or equal to (zero)
Unconditional jump:
jmp *target* – jump to *target*

MEMORY ADDRESS SYNTAX

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