X86-64 (amd64) Reference Sheet
CS31 – Introduction to Computer Systems | Swarthmore College

Data Movement and Control Transfer

data movement: ~ "copy"	**mov** S, D	D = S [Destination = Source]
load effective address	**lea** S, D	D = address of S
branch	**br** address	PC = address
compare	**cmp** src2, src1	Sets condition codes based on (src1 - src2)
test	**test** src2, src1	Sets condition codes based on (src1 & src2)
compare and branch if zero	**cbz** R, label	if R == 0, PC = address of label
compare and branch if not zero	**cbnz** R, label	if R != 0, PC = address of label
jump to label	**jmp** label	PC = address of label
jump if equal	**je** label	if cond., PC = addr. of label
jump if not equal	**jne** label	if cond., PC = addr. of label
jump if negative	**js** label	if cond., PC = addr. of label
jump if non-negative	**jns** label	if cond., PC = addr. of label
jump if greater than	**jg** label	if cond., PC = addr. of label
jump if greater than or equal	**jge** label	if cond., PC = addr. of label
jump if less than	**jl** label	if cond., PC = addr. of label
jump if less than or equal	**jle** label	if cond., PC = addr. of label
jump above	**ja** label	(unsigned jg)
jump below	**jb** label	(unsigned jl)
push	**push** S	%rsp = %rsp – 8 mem[%rsp] = S
pop	**pop** D	D = mem[%rsp] %rsp = %rsp +8
call (function)	**callq** label	push address of next instr. jmp label
leave stack frame	**leaveq**	mov %rbp, %rsp pop %rbp
return	**retq**	pop %rbp pop addres of next instr

add	add **S, D**	**D = D + S**
subtract	**sub** S, D	D = D – S
multiply	**imul** S, D	D = D * S
divide	**idiv** S	%rax = %rax / S %rdx = remainder
negate	**neg** D	D = - (D)
shift (logical) left	**shl** c, D	D = D << c
shift (logical) right	**shr** c, D	D = D >> c (logical)
shift arithmetic right	**sar** c, D	D = D >> c (arithmetic – sign)
bitwise and	**and** S, D	D = D & S
bitwise or	**or** S, D	D = D | S
bitwise xor	**xor** S, D	D = D ^ S
bitwise not	**not** D	D = ~D
increment	**inc** D	D = D + 1
decrement	**dec** D	D = D – 1

Addressing Modes

Immediate (constant): A number prefixed with **$**. Can be decimal or hexadecimal.

Examples: $8 $0x1F $-32

Register: A register name prefixed with **%**.

Examples: %rax %rbp %r15

Memory (normal): Access memory at the address stored in a register **(%reg)**.

Examples: (%rax) (%rbp)

Memory (displacement): Access memory at the address stored in a register *plus* a constant *C*:

C(%reg)

Examples: 8(%rbp) -0x10(%rsp)

Memory (index): Access memory at the address stored in a register (base) *plus* a constant, *C*, *plus* a *scale* * a *register (index)*:

C(%base, %index, scale)

Examples:
(%rax, %rcx)
0x8(% rbp, %rax, 8)

Arithmetic Operation Instruction Suffixes

b byte
w word (2 bytes)
l long (4 bytes)
q quad (8 bytes)

Condition Codes

ZF Zero Flag
SF Sign Flag (negative)
CF Carry Flag (unsigned overflow)
OF Overflow Flag (signed overflow)

Note: S = source, D = destination
Registers prefixed with **e** rather than **r** represent the lower 32-bits of a register (e.g., %eax vs %rax)