

CS107 x86-64 Reference Sheet

Common instructions

mov src, dst	dst = src
movsbl src, dst	byte to int, sign-extend
movzbl src, dst	byte to int, zero-fill
cmov src, reg	reg = src when condition holds, using same condition suffixes as jmp
lea addr, dst	dst = addr
add src, dst	dst += src
sub src, dst	dst -= src
imul src, dst	dst *= src
neg dst	dst = -dst (arith inverse)
imulq S	signed full multiply R[%rdx]:R[%rax] <- S * R[%rax]
mulq S	unsigned full multiply same effect as imulq
idivq S	signed divide R[%rdx] <- R[%rdx]:R[%rax] mod S R[%rax] <- R[%rdx]:R[%rax] / S
divq S	unsigned divide - same effect as idivq
cqto	R[%rdx]:R[%rax] <- SignExtend(R[%rax])
sal count, dst	dst <<= count
sar count, dst	dst >>= count (arith shift)
shr count, dst	dst >>= count (logical shift)
and src, dst	dst &= src
or src, dst	dst = src
xor src, dst	dst ^= src
not dst	dst = ~dst (bitwise inverse)
cmp a, b	b-a, set flags
test a, b	a&b, set flags
set dst	sets byte at dst to 1 when condition holds, 0 otherwise, using same condition suffixes as jmp
jmp label	jump to label (unconditional)
je label	jump equal ZF=1
jne label	jump not equal ZF=0
js label	jump negative SF=1
jns label	jump not negative SF=0
jg label	jump > (signed) ZF=0 and SF=OF
jge label	jump >= (signed) SF=OF
jl label	jump < (signed) SF!=OF
jle label	jump <= (signed) ZF=1 or SF!=OF
ja label	jump > (unsigned) CF=0 and ZF=0
jae label	jump >= (unsigned) CF=0
jb label	jump < (unsigned) CF=1
jbe label	jump <= (unsigned) CF=1 or ZF=1

push src	add to top of stack Mem[--%rsp] = src
pop dst	remove top from stack dst = Mem[%rsp++]
call fn	push %rip, jmp to fn
ret	pop %rip

Condition codes/flags

ZF	Zero flag
SF	Sign flag
CF	Carry flag
OF	Overflow flag

Addressing modes

Example source operands to **mov**

Immediate

mov \$0x5, dst

\$val

source is constant value

Register

mov %rax, dst

%R

R is register

source in %R register

Direct

mov 0x4033d0, dst

0xaddr

source read from Mem[0xaddr]

Indirect

mov (%rax), dst

(%R)

R is register

source read from Mem[%R]

Indirect displacement

mov 8(%rax), dst

D(%R)

R is register

D is displacement

source read from Mem[%R + D]

Indirect scaled-index

mov 8(%rsp, %rcx, 4), dst

D(%RB,%RI,S)

RB is register for base

RI is register for index (0 if empty)

D is displacement (0 if empty)

S is scale 1, 2, 4 or 8 (1 if empty)

source read from:

Mem[%RB + D + S*%RI]

CS107 x86-64 Reference Sheet

Registers

<code>%rip</code>	Instruction pointer
<code>%rsp</code>	Stack pointer
<code>%rax</code>	Return value
<code>%rdi</code>	1st argument
<code>%rsi</code>	2nd argument
<code>%rdx</code>	3rd argument
<code>%rcx</code>	4th argument
<code>%r8</code>	5th argument
<code>%r9</code>	6th argument
<code>%r10,%r11</code>	Callee-owned
<code>%rbx,%rbp, %r12-%15</code>	Caller-owned

Instruction suffixes

<code>b</code>	byte
<code>w</code>	word (2 bytes)
<code>l</code>	long /doubleword (4 bytes)
<code>q</code>	quadword (8 bytes)

Suffix is elided when can be inferred from operands. e.g. operand `%rax` implies `q`, `%eax` implies `l`, and so on

Register Names

64-bit register	32-bit sub-register	16-bit sub-register	8-bit sub-register
<code>%rax</code>	<code>%eax</code>	<code>%ax</code>	<code>%al</code>
<code>%rbx</code>	<code>%ebx</code>	<code>%bx</code>	<code>%bl</code>
<code>%rcx</code>	<code>%ecx</code>	<code>%cx</code>	<code>%cl</code>
<code>%rdx</code>	<code>%edx</code>	<code>%dx</code>	<code>%dl</code>
<code>%rsi</code>	<code>%esi</code>	<code>%si</code>	<code>%sil</code>
<code>%rdi</code>	<code>%edi</code>	<code>%di</code>	<code>%dil</code>
<code>%rbp</code>	<code>%ebp</code>	<code>%bp</code>	<code>%bpl</code>
<code>%rsp</code>	<code>%esp</code>	<code>%sp</code>	<code>%spl</code>
<code>%r8</code>	<code>%r8d</code>	<code>%r8w</code>	<code>%r8b</code>
<code>%r9</code>	<code>%r9d</code>	<code>%r9w</code>	<code>%r9b</code>
<code>%r10</code>	<code>%r10d</code>	<code>%r10w</code>	<code>%r10b</code>
<code>%r11</code>	<code>%r11d</code>	<code>%r11w</code>	<code>%r11b</code>
<code>%r12</code>	<code>%r12d</code>	<code>%r12w</code>	<code>%r12b</code>
<code>%r13</code>	<code>%r13d</code>	<code>%r13w</code>	<code>%r13b</code>
<code>%r14</code>	<code>%r14d</code>	<code>%r14w</code>	<code>%r14b</code>
<code>%r15</code>	<code>%r15d</code>	<code>%r15w</code>	<code>%r15b</code>