# CS107 x86-64 Reference Sheet

Indicate	Common instructions  mov src, dst dst = src  movsbl src, dst byte to int, sign-extend		<pre>push src</pre>	
add src, dst dst = src sub src, dst dst = src imul src, dst dst = strc imul src, dst dst = smc effect as imul source in sconstant value    mmediate		reg = src when condition holds,	call fn push %rip, jmp to fn	
add         src, dst         dst = src           imul src, dst         dst = src           imul src, dst         dst = st (arith inverse)           imulq S         signed full multiply same effect as imulq         Addressing modes           idivq S         signed divide R[%rdx]:R[%rax] = R[%rdx]:R[%rax] mod S R[%rdx] < R[%rdx]:R[%rax] / S         Swal           divq S         signed divide - same effect as idivq R[%rax] / S (R[%rax] < - SignExtend(R[%rax])         Register           cutount, dst         dst <= count         dst >= count (logical shift)         R is register           sal count, dst         dst >= count (logical shift)         dst >= src           sar count, dst         dst >= src         or src, dst         dst = src           cor src, dst         dst = src         mov ½A433de, dst           cor src, dst         dst = src         mov ½A433de, dst           cor src, dst         dst = src         mov ½A433de, dst           cor src, dst         dst = src         mov ½A433de, dst           set dst         sets byte at dst to 1 when condition holds, 0 otherwise, using same condition suffixes as jmp         Indirect           jip label         jump to label (unconditional)         jump equal ZF=1         jum peacle (SF=OF)           jie label         jump >= (signed) SF=OF)         jis (signed) SF=OF) <td><b>lea</b> addr, dst</td> <td>dst = addr</td> <td colspan="2">Condition codes/flags</td>	<b>lea</b> addr, dst	dst = addr	Condition codes/flags	
R[%rdx] R[%rax] <- S * R[%rax]   Example source operands to mov	<pre>sub src, dst imul src, dst</pre>	dst -= src dst *= src	SF Sign flag CF Carry flag	
idivq S signed divide R[%rdx] <- R[%rdx]   Signed divide R[%rdx]   Signed divide - Same effect as idivq R[%rdx]   R[%rax]   Signed divide - Same effect as idivq R[%rdx]   R[%rax]   Signed divide - Same effect as idivq R[%rdx]   Ris register Register Rov	•	R[%rdx]:R[%rax] <- S * R[%rax]		
signed divide   R[%rdx] < R[%rdx] < R[%rdx]   R[%rax]   Mod   S   R[%rax]   S   S   Register   Mov %rax, dst   Move of the source in Section   Mode, of the source in Section   Mode, of the source	шити э			
divq S unsigned divide - same effect as idivq 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 >>= count (logical shift) shr count, dst dst = src source in %R register  or src, dst dst  = src source in %R register  Direct  mov @x4033d@, dst  example of ex	idivq S	$R[\%rdx] \leftarrow R[\%rdx]:R[\%rax] \mod S$	\$val	
sal count, dst dst <= count sar count, dst dst >= count (arith shift) sar count, dst dst >= count (logical shift) and src, dst dst >= count (logical shift) and src, dst dst = src src src src, dst dst = src src src, dst dst = src	diva S unsig		Register	
sar count, dst dst >>= count (arith shift) shr count, dst dst >>= count (logical shift) and src, dst dst &= src or src, dst dst  = src	-			
and src, dst dst &= src or src, dst dst  = src xor src, dst dst  = src not dst cmp a, b dst, set flags set dst set dst set spyte 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 jns label jump not equal ZF=0 js label jump not negative SF=0 jg label jump > (signed) ZF=0 and SF=OF jge label jump > (signed) SF=OF ja label jump > (signed) SF=OF ja label jump > (unsigned) CF=0 and ZF=0 jae label jump > (unsigned) CF=0 jbe label jump > (unsigned) CF=1 jbe label jump < (unsigned) CF=1 jum label jump < (unsigned) CF=1 jum label jump < (unsigned) CF=1 jump < (siglacement) source read from Mem[%R]  R is register source read from Mem[%R]  R is register D is displacement source read from Mem[%R + D]  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 loase 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[0xaddr]	<b>sar</b> count, dst	dst >>= count (arith shift)		
xor src, dst not dst  dst = src  dst = dst (bitwise inverse)  cmp a, b		, ,		
not dst dst = ~dst (bitwise inverse)  cmp a, b b-a, set flags 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 jine label jump not equal ZF=0 js label jump not negative SF=0 jg label jump > (signed) ZF=0 and SF=OF jge label jump > (signed) ZF=0 or SF!=OF ja label jump < (signed) ZF=1 or SF!=OF ja label jump > (unsigned) CF=0 and ZF=0 jine label jump > (unsigned) CF=0 and ZF=0 jine label jump < (unsigned) CF=1 jine label jump > (unsigned) CF=1 or ZF=1 source read from Mem[0xaddr]    Indirect mov (%rax), dst mov 8(%rax), dst mov 8		·		
test a, b  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 not negative SF=0  jg label  jump > (signed) ZF=0 and SF=OF  jge 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  jbe label  jump < (unsigned) CF=1 or ZF=1  jump < (unsigned) CF=1 or ZF=1				
test a, b  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 not negative SF=1 jis label jump > (signed) ZF=0 and SF=OF jge 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 jbe label jump > (unsigned) CF=1 jump < (unsigned) CF=1 or ZF=1  sets byte at dst to 1 when condition (%R) R is register source read from Mem[%R]  D(%R) R is register D is displacement source read from Mem[%R + D]  Indirect scaled-index mov 8(%rsp, %rcx, 4), dst D(%R8), RI, S) R is register D is displacement source read from Mem[%R + D]  Indirect scaled-index mov 8(%rsp, %rcx, 4), dst D(%R8), RI, S) R is register for index indirect scaled-index source read from Mem[%R + D]  Indirect scaled-index indirect scaled	<b>cmp</b> a, b	b-a, set flags		
set dst  sets byte at dst to 1 when condition holds, 0 otherwise, using same condition suffixes as jmp  imp label  jump to label (unconditional)  je label  jump equal ZF=1  jne label  jump not equal ZF=0  js label  jump not negative SF=0  jg label  jump > (signed) ZF=0 and SF=OF  jge label  jump < (signed) SF!=OF  jle label  jump < (signed) ZF=1 or SF!=OF  ja label  jump > (unsigned) CF=0  ja label  jump > (unsigned) CF=0  jb label  jump < (unsigned) CF=1				
jmplabeljump to label (unconditional)mov 8(%rax), dstjelabeljump equal ZF=1D(%R)jnelabeljump not equal ZF=0R is registerjslabeljump negative SF=1D is displacementjnslabeljump not negative SF=0D is displacementjglabeljump > (signed) ZF=0 and SF=OFjglabeljump > (signed) SF=OFjllabeljump < (signed) ZF=1 or SF!=OFjalabeljump > (unsigned) CF=0 and ZF=0jalabeljump > (unsigned) CF=0jblabeljump < (unsigned) CF=1jblabeljump < (unsigned) CF=1 or ZF=1	<b>set</b> dst	holds, 0 otherwise, using same	R is register	
je label jump equal $ZF=1$ jne label jump not equal $ZF=0$ js label jump not negative $SF=1$ jns label jump not negative $SF=0$ jg label jump > (signed) $ZF=0$ and $ZF=0$ jl label jump >= (signed) $ZF=0$ jump >= (unsigned) $ZF=0$ jae label jump >= (unsigned) $ZF=0$ jump		condition cannot do jimp		
jge labeljump >= (signed) SF=OFIndirect scaled-indexjl labeljump < (signed) SF!=OF	<pre>je label jne label js label jns label</pre>	jump equal ZF=1 jump not equal ZF=0 jump negative SF=1 jump not negative SF=0	D(%R) R is register D is displacement	
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  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:		, , , ,	Indirect scaled-index	
	<pre>jl label jle label ja label jae label jb label</pre>	jump < (signed) SF!=OF jump <= (signed) ZF=1 or SF!=OF jump > (unsigned) CF=0 and ZF=0 jump >= (unsigned) CF=0 jump < (unsigned) CF=1	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:	

# CS107 x86-64 Reference Sheet

### **Registers**

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

#### byte

b

word (2 bytes) W

1 long /doubleword (4 bytes)

quadword (8 bytes)

**Instruction suffixes** 

Suffix is elided when can be inferred from operands. e.g. operand %rax implies q,

%eax implies 1, and so on

%r12-%15 Caller-owned

# **Register Names**

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