CS107 Handy one-page of x86-64

Common instructions

mov src, dst	dst = src
movsbl src, dst	byte to int, sign-extend
movzbl src, dst	byte to int, zero-fill
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)
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
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
jb label	jump < (unsigned) CF=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
cmov src, register	reg = src when condition holds, using same condition suffixes as jmp

Instruction suffixes

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

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

Condition codes/flags

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

Registers

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

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]