x86-64 Reference Sheet (GNU assembler format)

Instructions

Data movement

`movq` Src, Dest	Dest ← Src
`movsbq` Src,Dest	Dest (quad) ← Src (byte), sign-extend
`movzbq` Src,Dest	Dest (quad) ← Src (byte), zero-extend

Conditional move

`cmove` Src, Dest	Equal / zero
`cmovne` Src, Dest	Not equal / not zero
`cmovs` Src, Dest	Negative
`cmovns` Src, Dest	Nonnegative
`cmovg` Src, Dest	Greater (signed >)
`cmovge` Src, Dest	Greater or equal (signed ≥)
`cmovl` Src, Dest	Less (signed <)
`cmovle` Src, Dest	Less or equal (signed ≤)
`cmova` Src, Dest	Above (unsigned >)
`cmovae` Src, Dest	Above or equal (unsigned ≥)
`cmovb` Src, Dest	Below (unsigned <)
`cmovbe` Src, Dest	Below or equal (unsigned ≤)

Control transfer

`jmp` Label	jump
`je` Label	jump equal
`jne` Label	jump not equal
`js` Label	jump negative
`jns` Label	jump non-negative
`jg` Label	jump greater (signed >)
`jge` Label	jump greater or equal (signed ≥)
`jl` Label	jump less (signed <)
`jle` Label	jump less or equal (signed ≤)
`ja` Label	jump above (unsigned >)
`jb` Label	jump below (unsigned <)
`jmp *`Src	jump indirect (register)
`pushq` Src	%rsp ← %rsp – 8, Mem[%rsp] ← Src
`popq` Dest	Dest ← Mem[%rsp], %rsp ← %rsp + 8
`call` Label	push addr next instruction, `jmp label`
`ret`	%rip ← Mem[%rsp], %rsp ← %rsp + 8
`endbr64`	End branch target (no-op)
`leave`	%rsp ← %rbp ; `popq` %rbp

Arithmetic operations

`leaq` Src, Dest	Dest ← address of Src
`incq` Dest	Dest ← Dest + 1
`decq` Dest	Dest ← Dest − 1
`addq` Src, Dest	Dest ← Dest + Src
`subq` Src, Dest	Dest ← Dest − Src
`imulq` Src, Dest	Dest ← Dest ∗ Src
`xorq` Src, Dest	Dest ← Dest ^ Src
`orq` Src, Dest	Dest ← Dest | Src
`andq` Src, Dest	Dest ← Dest & Src
`negq` Dest	Dest ← − Dest
`notq` Dest	Dest ← ∼ Dest
`salq` k, Dest	Dest ← Dest $\ll k$ (also `shlq`)
`sarq` k, Dest	Dest ← Dest $\gg k$ (arithmetic)
`shrq` k, Dest	Dest ← Dest $\gg k$ (logical)
`cmpq` Src2, Src1	Set CCs Src1 – Src2
`testq` Src2, Src1	Set CCs Src1 & Src2

Integer registers

`%rax`	Return value
`%rbx`	Callee saved
`%rcx`	4th argument
`%rdx`	3rd argument
`%rsi`	2nd argument
`%rdi`	1st argument
`%rbp`	Callee saved
`%rsp`	Stack pointer
`%r8`	5th argument
`%r9`	6th argument
`%r10`	Scratch register
`%r11`	Scratch register
`%r12`	Callee saved
`%r13`	Callee saved
`%r14`	Callee saved
`%r15`	Callee saved

Addressing modes

- **Immediate**
  $val Val
  val: constant integer value
  `movq $7, %rax`
- **Normal**
  (R) Mem[Reg[R]]
  R: register R specifies memory address
  `movq (%rcx), %rax`
- **Displacement**
  D(R) Mem[Reg[R]+D]
  R: register specifies start of memory region
  D: constant displacement D specifies offset
  `movq 8(%rdi), %rdx`
- **Indexed**
  D(Rb,Ri,S) Mem[Reg[Rb]+S*Reg[Ri]+D]
  D: displacement: 1, 2, or 4 byte constant
  Rb: base register: any integer register
  Ri: index register: any, except %esp
  S: scale: 1, 2, 4, or 8
  `movq 0x100(%rcx,%rax,4), %rdx`

Instruction suffixes

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

Condition codes

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