

x86-64 Reference Sheet (GNU assembler format)

Instructions

Data movement

`movq Src, Dest` $\text{Dest} \leftarrow \text{Src}$
`movsbq Src, Dest` $\text{Dest (quad)} \leftarrow \text{Src (byte), sign-extend}$
`movzbq Src, Dest` $\text{Dest (quad)} \leftarrow \text{Src (byte), zero-extend}$

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 $\geq$)
`cmovl Src, Dest` Less (signed <)
`cmovle Src, Dest` Less or equal (signed $\leq$)
`cmova Src, Dest` Above (unsigned >)
`cmovae Src, Dest` Above or equal (unsigned $\geq$)
`cmovb Src, Dest` Below (unsigned <)
`cmovbe Src, Dest` Below or equal (unsigned $\leq$)

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 $\geq$)
`jl label` jump less (signed <)
`jle label` jump less or equal (signed $\leq$)
`ja label` jump above (unsigned >)
`jb label` jump below (unsigned <)
`pushq Src` $\%rsp \leftarrow \%rsp - 8, \text{Mem}[\%rsp] \leftarrow \text{Src}$
`popq Dest` $\text{Dest} \leftarrow \text{Mem}[\%rsp], \%rsp \leftarrow \%rsp + 8$
`call label` push addr next instruction, `jmp label`
`ret` $\%rip \leftarrow \text{Mem}[\%rsp], \%rsp \leftarrow \%rsp + 8$
`endbr64` End branch target (no-op)
`leave` $\%rsp \leftarrow \%rbp ; \text{popq } \%rbp$

Arithmetic operations

`leaq Src, Dest` $\text{Dest} \leftarrow \text{address of Src}$
`incq Dest` $\text{Dest} \leftarrow \text{Dest} + 1$
`decq Dest` $\text{Dest} \leftarrow \text{Dest} - 1$
`addq Src, Dest` $\text{Dest} \leftarrow \text{Dest} + \text{Src}$
`subq Src, Dest` $\text{Dest} \leftarrow \text{Dest} - \text{Src}$
`imulq Src, Dest` $\text{Dest} \leftarrow \text{Dest} * \text{Src}$
`xorq Src, Dest` $\text{Dest} \leftarrow \text{Dest} \wedge \text{Src}$
`orq Src, Dest` $\text{Dest} \leftarrow \text{Dest} \vee \text{Src}$
`andq Src, Dest` $\text{Dest} \leftarrow \text{Dest} \wedge \text{Src}$
`negq Dest` $\text{Dest} \leftarrow -\text{Dest}$
`notq Dest` $\text{Dest} \leftarrow \sim \text{Dest}$
`salq k, Dest` $\text{Dest} \leftarrow \text{Dest} \ll k$ (also `shlq`)
`sarq k, Dest` $\text{Dest} \leftarrow \text{Dest} \gg k$ (arithmetic)
`shrq k, Dest` $\text{Dest} \leftarrow \text{Dest} \gg k$ (logical)
`cmpq Src2, Src1` Set CCs $\text{Src1} - \text{Src2}$
`testq Src2, Src1` Set CCs $\text{Src1} \& \text{Src2}$

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`

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

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