x86-64 Cheat Sheet

General-Purpose Registers

Name	64-bit	32-bit	16-bit	8-bit	Preserved	Usage
Accumulator	`rax`	`eax`	`ax`	`ah:al`		return
Base	`rbx`	`ebx`	`bx`	`bh:bl`	✓	
Counter	`rcx`	`ecx`	`cx`	`ch:cl`		arg4
Data	`rdx`	`edx`	`dx`	`dh:dl`		arg3
Source	`rsi`	`esi`	`si`	`sil`		arg2
Destination	`rdi`	`edi`	`di`	`dil`		arg1
	`r8`	`r8d`	`r8w`	`r8b`		arg5
	`r9`	`r9d`	`r9w`	`r9b`		arg6
	`r10`	`r10d`	`r10w`	`r10b`		
	`r11`	`r11d`	`r11w`	`r11b`		
	`r12`	`r12d`	`r12w`	`r12b`	✓	
	`r13`	`r13d`	`r13w`	`r13b`	✓	
	`r14`	`r14d`	`r14w`	`r14b`	✓	
	`r15`	`r15d`	`r15w`	`r15b`	✓	
Base Pointer	`rbp`	`ebp`	`bp`	`bpl`	✓	function stack base (optional)
Stack Pointer	`rsp`	`esp`	`sp`	`spl`	✓	top of stack *
Instruction Pointer	`rip`	`eip`	`ip`			

* The `rsp` register must be 16-byte aligned (i.e. divisible by 16) before making a `call`.

Flags

Label	Name	Description
`CF`	carry flag	last arithmetic operation carried/borrowed
`ZF`	zero flag	result of last operation was zero
`SF`	sign flag	result of last operation was negative
`OF`	overflow flag	last result had a signed overflow

Jumps

Instruction	Test	After `TEST x, x`	After `CMP y, x`
`je`	`ZF`	x = 0	x = y
`jne`	`~ZF`	x ≠ 0	x ≠ y
`ja`	`~CF & ~ZF`		x > y (unsigned)
`jae`	`~CF`		x ≥ y (unsigned)

Instruction	Test	After `TEST x, x`	After `CMP y, x`
`jb`	`CF`		x < y (unsigned)
`jbe`	`CF | ZF`		x ≤ y (unsigned)
`jg`	`~ZF & SF=OF`	x > 0 (signed)	x > y (signed)
`jge`	`SF=OF`	x ≥ 0 (signed)	x ≥ y (signed)
`jl`	`SF≠OF`	x < 0 (signed)	x < y (signed)
`jle`	`ZF | SF≠OF`	x ≤ 0 (signed)	x ≤ y (signed)
`jc`	`CF`		
`jnc`	`~CF`		
`jo`	`OF`		
`jno`	`~OF`		
`js`	`SF`		
`jns`	`~SF`		

Floating Point

Floating point registers `st(0)` to `st(7)` are manipulated by the x87 instructions, which we won't be using.

SIMD: MMX, SSE, AVX

We will be using these registers and the instructions that work on them for floating point operations:

512-bit	256-bit	128-bit	Preserved	Usage
`zmm0`	`ymm0`	`xmm0`		arg1, return
`zmm1`	`ymm1`	`xmm1`		arg2
⋮				⋮
`zmm7`	`ymm7`	`xmm7`		arg8
`zmm8`	`ymm8`	`xmm8`		
⋮				
`zmm15`	`ymm15`	`xmm15`		

Addressing

Memory access in an instruction: `displacement(base, index, scale)`

e.g. `-16(%rbp, %rdx, 8)` == `rbp + (rdx * 8) - 16`. In C, that might be `rbp[rdx-2]` if you're working with elements of 8 bytes.

Operand Size

Name	Size	Suffix	Data

Name	Size	Suffix	Data
byte	8-bits	`b`	`.byte`
word	16-bits	`w`	`.word`
long word	32-bits	`l`	`.long`
quad word	64-bits	`q`	`.quad`

Instructions

- Integer movement: `mov`, `push`, `pop`, `movabs`, `movsx`, `movzx`.
- Unsigned integer arithmetic: `add`, `sub`, `mul`, `div`, `shl`, `shr`.
- Signed integer arithmetic: `add`, `sub`, `imul`, `idiv`, `sal`, `sar`, `neg`.
- Bit manipulation: `and`, `or`, `not`.
- Integer comparison: `cmp`, `test`
- Jumps: `jmp`, `j`*X*, `jn`*X*, `call`, `ret`.
- Pointer creation: `lea`.
- Conditional move: `cmov`*X*.
- Double-precision scalars: `movq`, `addsd`, `subsd`, `mulsd`, `divsd`, `comisd`, `pxor`.
- Single-precision scalars: `movd`, `addss`, `subss`, `mulss`, `divss`, `comiss`, `pxor`.
- Double-precision vectors: `vmovupd`, `vaddpd`, `vsubpd`, `vmulpd`, `vdivpd`, `vxorpd`, `vbroadcastsd`.
- Single-precision vectors: `vmovups`, `vaddps`, `vsubps`, `vmulps`, `vdivps`, `vxorps`, `vbroadcastss`.

References: Intel 64 and IA-32 Architectures Software Developer's Manuals; System V Application Binary Interface: AMD64 Architecture Processor Supplement.