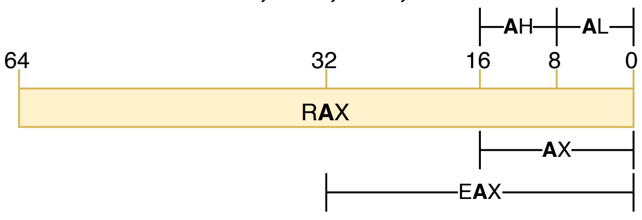
x86 Cheat Sheet

General Purpose Registers (GPR)

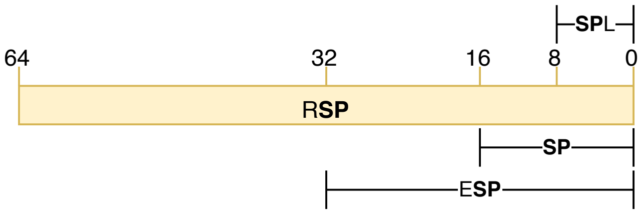
`EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP`	32-bit mode
`RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8, R9, R10, R11, R12, R13, R14, R15`	64-bit mode

Sub Registers

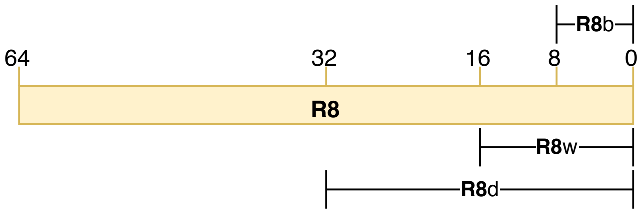
For `RAX, RBX, RCX, RDX`



For `RSP, RBP, RSI, RDI`



For `R8, R9, R10, R11, R12, R13, R14, R15`



Fundamental Datatypes

`BYTE`	8 bits
`WORD`	16 bits (2 bytes)
`DWORD`	32 bits (4 bytes)
`QWORD`	64 bits (8 bytes)

RFLAGS - Status Flags

`CF`	Carry flag, e.g. $2^{64-1} + 2^{64-1}$ sets CF
`PF`	Parity flag,
`AF`	Adjust flag
`ZF`	Zero flag, e.g. $2 - 2$ sets ZF
`SF`	Sign flag, indicates the MSB is set
`OF`	Overflow flag

Status flags are set by all arithmetic instructions.

RFLAGS - Control Flags

`TF`	Trap flag, for single-step debugging
`IF`	Interrupt enable (set by OS kernel)

Instruction Syntax

`instr op1, op2, op3`	Instructions can have up to three operands
`<SIZE> PTR[<expr>]`	Dereference <expr>

Calling Conventions (Linux)

Integer arguments are passed left to right to `RDI, RSI, RDX, RCX, R8` and `R9` then on the stack

Return values `EDX.EAX`
Volatile `RAX, RCX, RDX, RDI, RSI, R8, R9, R10, R11`
Non-volatile `RBX, RBP, RSP, R12, R13, R14, R15`

Data Transfer Instructions

`CMOVxx dst, src`	Conditional moves
`MOV dst, src`	Move data or immediate to memory or register
`PUSH op`	Push operand `op` onto the stack
`POP op`	Pop value from the stack to `op` (register or memory)

Arithmetic Instructions

`ADD dst, src`	`dst` = `dst` + `src`	
`SUB dst, src`	`dst` = `dst` - `src`	
`MUL dst, src`	`dst` = `dst` · `src`	(unsigned)
`IMUL dst, src`	`dst` = `dst` · `src`	(signed)
`DIV dst, src`	`dst` = `dst` ÷ `src`	(unsigned)
`IDIV dst, src`	`dst` = `dst` ÷ `src`	(signed)
`INC op`	Increments `op` (register or memory) by 1 without changing CF	
`DEC op`	Decrements `op` (register or memory) by 1 without changing CF	
`CMP op1, op2`	Calculates `op1 - op2` and sets status flags (result is discarded)	

Logical Instructions

`AND dst, src`	`dst` = bitwise AND of `dst` and `src`
`OR dst, src`	`dst` = bitwise OR of `dst` and `src`
`XOR dst, src`	`dst` = bitwise XOR of `dst` and `src`
`NOT op`	`op` = bitwise NOT of `op`
`TEST op1, op2`	Calculates `op1 & op2` and sets status flags (result is discarded)

Control Transfer Instructions

`JMP op`	Jumps to target address
`Jxx op`	Conditional jump
`CALL op`	Call procedure (at taregt address)
`RET`	Return from procedure call

Shift and Rotate Instructions

`SHR dst, src`
`SHL dst, src`
	Shift bits of dst by src bits to the right (left) and stores in dst

`ROR dst, src`
`ROL dst, src`
	Rotates bits of dst by src bits to the right (left) and stores in dst

`RCR dst, src`
`RCL dst, src`
	Rotates bits of dst by src bits to the right (left) through carry and stores in dst

Miscellaneous Instructions

`LEA dst, src`	Calculates effective address of src and stores it in dst
`NOP`	No operation

References

"Malware Analysis and x86 Reverse Engineering" J. Wichelmann
"Intel® 64 and IA-32 Architectures Software Developer Manuals" https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html