Instruction	**Operation**
`addq` A, B	$A + B \rightarrow B$
`negq` A	$-A \rightarrow A$
`subq` A, B	$B - A \rightarrow B$
`callq` L	Pushes the return address and jumps to label L
`callq` $*A$	Calls the function at the address A.
`retq`	Pops the return address and jumps to it
`popq` A	$*\texttt{rsp} \rightarrow A; \texttt{rsp} + 8 \rightarrow \texttt{rsp}$
`pushq` A	$\texttt{rsp} - 8 \rightarrow \texttt{rsp}; A \rightarrow *\texttt{rsp}$
`leaq` A,B	$A \rightarrow B$ (C must be a register)
`cmpq` A, B	compare A and B and set flag
`je` L `jl` L `jle` L `jg` L `jge` L	Jump to label L if the flag matches the condition code, otherwise go to the next instructions. The condition codes are `e` for "equal", `l` for "less", `le` for "less or equal", `g` for "greater", and `ge` for "greater or equal".
`jmp` L	Jump to label L
`movq` A, B	$A \rightarrow B$
`movzbq` A, B	$A \rightarrow B$, where A is a single-byte register (e.g., `al` or `cl`), B is a 8-byte register, and the extra bytes of B are set to zero.
`notq` A	$\sim A \rightarrow A$ (bitwise complement)
`orq` A, B	$A \mid B \rightarrow B$ (bitwise-or)
`andq` A, B	$A \& B \rightarrow B$ (bitwise-and)
`salq` A, B	$B \ll A \rightarrow B$ (arithmetic shift left, where A is a constant)
`sarq` A, B	$B \gg A \rightarrow B$ (arithmetic shift right, where A is a constant)
`sete` A `setl` A `setle` A `setg` A `setge` A	If the flag matches the condition code, then $1 \rightarrow A$, else $0 \rightarrow A$. Refer to `je` above for the description of the condition codes. A must be a single byte register (e.g., `al` or `cl`).

Table 12.1: Quick-reference for the x86 instructions used in this book.