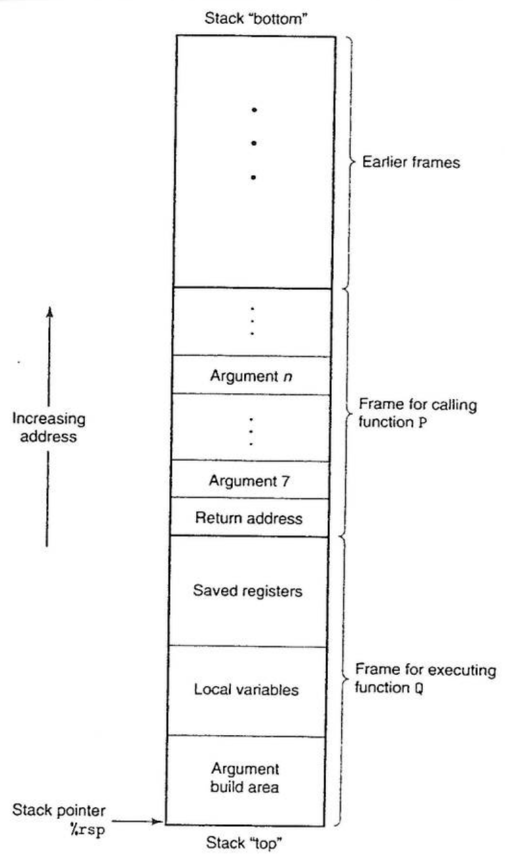
63	31	15	7 0	
%rax	%eax	%ax	%al	Return value
%rbx	%ebx	%bx	%bl	Callee saved
%rcx	%ecx	%cx	%cl	4th argument
%rdx	%edx	%dx	%dl	3rd argument
%rsi	%esi	%si	%sil	2nd argument
%rdi	%edi	%di	%dil	1st argument
%rbp	%ebp	%bp	%bpl	Callee saved
%rsp	%esp	%sp	%spl	Stack pointer
%r8	%r8d	%r8w	%r8b	5th argument
%r9	%r9d	%r9w	%r9b	6th argument
%r10	%r10d	%r10w	%r10b	Caller saved
%r11	%r11d	%r11w	%r11b	Caller saved
%r12	%r12d	%r12w	%r12b	Callee saved
%r13	%r13d	%r13w	%r13b	Callee saved
%r14	%r14d	%r14w	%r14b	Callee saved
%r15	%r15d	%r15w	%r15b	Callee saved

Figure 3.2 **Integer registers.** The low-order portions of all 16 registers can be accessed as byte, word (16-bit), double word (32-bit), and quad word (64-bit) quantities.



Type	Form	Operand value	Name
Immediate	$\$Imm$	Imm	Immediate
Register	r_a	$R[r_a]$	Register
Memory	Imm	$M[Imm]$	Absolute
Memory	(r_a)	$M[R[r_a]]$	Indirect
Memory	$Imm(r_b)$	$M[Imm + R[r_b]]$	Base + displacement
Memory	(r_b,r_i)	$M[R[r_b] + R[r_i]]$	Indexed
Memory	$Imm(r_b,r_i)$	$M[Imm + R[r_b] + R[r_i]]$	Indexed
Memory	$(,r_i,s)$	$M[R[r_i] \cdot s]$	Scaled indexed
Memory	$Imm(,r_i,s)$	$M[Imm + R[r_i] \cdot s]$	Scaled indexed
Memory	(r_b,r_i,s)	$M[R[r_b] + R[r_i] \cdot s]$	Scaled indexed
Memory	$Imm(r_b,r_i,s)$	$M[Imm + R[r_b] + R[r_i] \cdot s]$	Scaled indexed

Figure 3.3 **Operand forms.** Operands can denote immediate (constant) values, register values, or values from memory. The scaling factor s must be either 1, 2, 4, or 8.

Operand size (bits)	Argument number 1	2	3	4	5	6
64	%rdi	%rsi	%rdx	%rcx	%r8	%r9
32	%edi	%esi	%edx	%ecx	%r8d	%r9d
16	%di	%si	%dx	%cx	%r8w	%r9w
8	%dil	%sil	%dl	%cl	%r8b	%r9b

Instruction		Effect	Description
MOV	S, D	$D \leftarrow S$	Move
movb			Move byte
movw			Move word
movl			Move double word
movq			Move quad word
movabsq	I, R	$R \leftarrow I$	Move absolute quad word

C declaration	Intel data type	Assembly-code suffix	Size (bytes)
char	Byte	b	1
short	Word	w	2
int	Double word	l	4
long	Quad word	q	8
char *	Quad word	q	8
float	Single precision	s	4
double	Double precision	l	8

Instruction	Effect	Description
pushq S	R[%rsp] $\leftarrow$ R[%rsp] − 8; M[R[%rsp]] $\leftarrow S$	Push quad word
popq D	$D \leftarrow$ M[R[%rsp]]; R[%rsp] $\leftarrow$ R[%rsp] + 8	Pop quad word

Instruction		Effect	Description
`leaq`	S, D	$D \leftarrow \&S$	Load effective address
INC	D	$D \leftarrow D+1$	Increment
DEC	D	$D \leftarrow D-1$	Decrement
NEG	D	$D \leftarrow -D$	Negate
NOT	D	$D \leftarrow \sim D$	Complement
ADD	S, D	$D \leftarrow D + S$	Add
SUB	S, D	$D \leftarrow D - S$	Subtract
IMUL	S, D	$D \leftarrow D * S$	Multiply
XOR	S, D	$D \leftarrow D \verb|^| S$	Exclusive-or
OR	S, D	$D \leftarrow D \mid S$	Or
AND	S, D	$D \leftarrow D \& S$	And
SAL	k, D	$D \leftarrow D << k$	Left shift
SHL	k, D	$D \leftarrow D << k$	Left shift (same as SAL)
SAR	k, D	$D \leftarrow D >>_A k$	Arithmetic right shift
SHR	k, D	$D \leftarrow D >>_L k$	Logical right shift

Instruction		Synonym	Jump condition	Description
`jmp`	*Label*		1	Direct jump
`jmp`	**Operand*		1	Indirect jump
`je`	*Label*	`jz`	ZF	Equal / zero
`jne`	*Label*	`jnz`	~ZF	Not equal / not zero
`js`	*Label*		SF	Negative
`jns`	*Label*		~SF	Nonnegative
`jg`	*Label*	`jnle`	~(SF ^ OF) & ~ZF	Greater (signed >)
`jge`	*Label*	`jnl`	~(SF ^ OF)	Greater or equal (signed >=)
`jl`	*Label*	`jnge`	SF ^ OF	Less (signed <)
`jle`	*Label*	`jng`	(SF ^ OF) | ZF	Less or equal (signed <=)
`ja`	*Label*	`jnbe`	~CF & ~ZF	Above (unsigned >)
`jae`	*Label*	`jnb`	~CF	Above or equal (unsigned >=)
`jb`	*Label*	`jnae`	CF	Below (unsigned <)
`jbe`	*Label*	`jna`	CF | ZF	Below or equal (unsigned <=)

Instruction		Based on	Description
CMP	S_1, S_2	$S_2 - S_1$	Compare
`cmpb`			Compare byte
`cmpw`			Compare word
`cmpl`			Compare double word
`cmpq`			Compare quad word
TEST	S_1, S_2	$S_1 \& S_2$	Test
`testb`			Test byte
`testw`			Test word
`testl`			Test double word
`testq`			Test quad word

Instruction		Synonym	Move condition	Description
`cmove`	S, R	`cmovz`	ZF	Equal / zero
`cmovne`	S, R	`cmovnz`	~ZF	Not equal / not zero
`cmovs`	S, R		SF	Negative
`cmovns`	S, R		~SF	Nonnegative
`cmovg`	S, R	`cmovnle`	~(SF ^ OF) & ~ZF	Greater (signed >)
`cmovge`	S, R	`cmovnl`	~(SF ^ OF)	Greater or equal (signed >=)
`cmovl`	S, R	`cmovnge`	SF ^ OF	Less (signed <)
`cmovle`	S, R	`cmovng`	(SF ^ OF) | ZF	Less or equal (signed <=)
`cmova`	S, R	`cmovnbe`	~CF & ~ZF	Above (unsigned >)
`cmovae`	S, R	`cmovnb`	~CF	Above or equal (Unsigned >=)
`cmovb`	S, R	`cmovnae`	CF	Below (unsigned <)
`cmovbe`	S, R	`cmovna`	CF | ZF	Below or equal (unsigned <=)

Figure 3.18 **The conditional move instructions.** These instructions copy the source value S to its destination R when the move condition holds. Some instructions have "synonyms," alternate names for the same machine instruction.

Instruction		Effect	Description
MOVS	S, R	$R \leftarrow$ SignExtend(S)	Move with sign extension
`movsbw`			Move sign-extended byte to word
`movsbl`			Move sign-extended byte to double word
`movswl`			Move sign-extended word to double word
`movsbq`			Move sign-extended byte to quad word
`movswq`			Move sign-extended word to quad word
`movslq`			Move sign-extended double word to quad word
`cltq`		`%rax` $\leftarrow$ SignExtend(`%eax`)	Sign-extend `%eax` to `%rax`

Instruction		Effect	Description
MOVZ	S, R	$R \leftarrow$ ZeroExtend(S)	Move with zero extension
`movzbw`			Move zero-extended byte to word
`movzbl`			Move zero-extended byte to double word
`movzwl`			Move zero-extended word to double word
`movzbq`			Move zero-extended byte to quad word
`movzwq`			Move zero-extended word to quad word

Instruction		Synonym	Effect	Set condition
`sete`	D	`setz`	$D \leftarrow$ ZF	Equal / zero
`setne`	D	`setnz`	$D \leftarrow$ ~ZF	Not equal / not zero
`sets`	D		$D \leftarrow$ SF	Negative
`setns`	D		$D \leftarrow$ ~SF	Nonnegative
`setg`	D	`setnle`	$D \leftarrow$ ~(SF ^ OF) & ~ZF	Greater (signed >)
`setge`	D	`setnl`	$D \leftarrow$ ~(SF ^ OF)	Greater or equal (signed >=)
`setl`	D	`setnge`	$D \leftarrow$ SF ^ OF	Less (signed <)
`setle`	D	`setng`	$D \leftarrow$ (SF ^ OF) | ZF	Less or equal (signed <=)
`seta`	D	`setnbe`	$D \leftarrow$ ~CF & ~ZF	Above (unsigned >)
`setae`	D	`setnb`	$D \leftarrow$ ~CF	Above or equal (unsigned >=)
`setb`	D	`setnae`	$D \leftarrow$ CF	Below (unsigned <)
`setbe`	D	`setna`	$D \leftarrow$ CF | ZF	Below or equal (unsigned <=)

Figure 3.14 **The SET instructions.** Each instruction sets a single byte to 0 or 1 based on some combination of the condition codes. Some instructions have "synonyms," that is, alternate names for the same machine instruction.