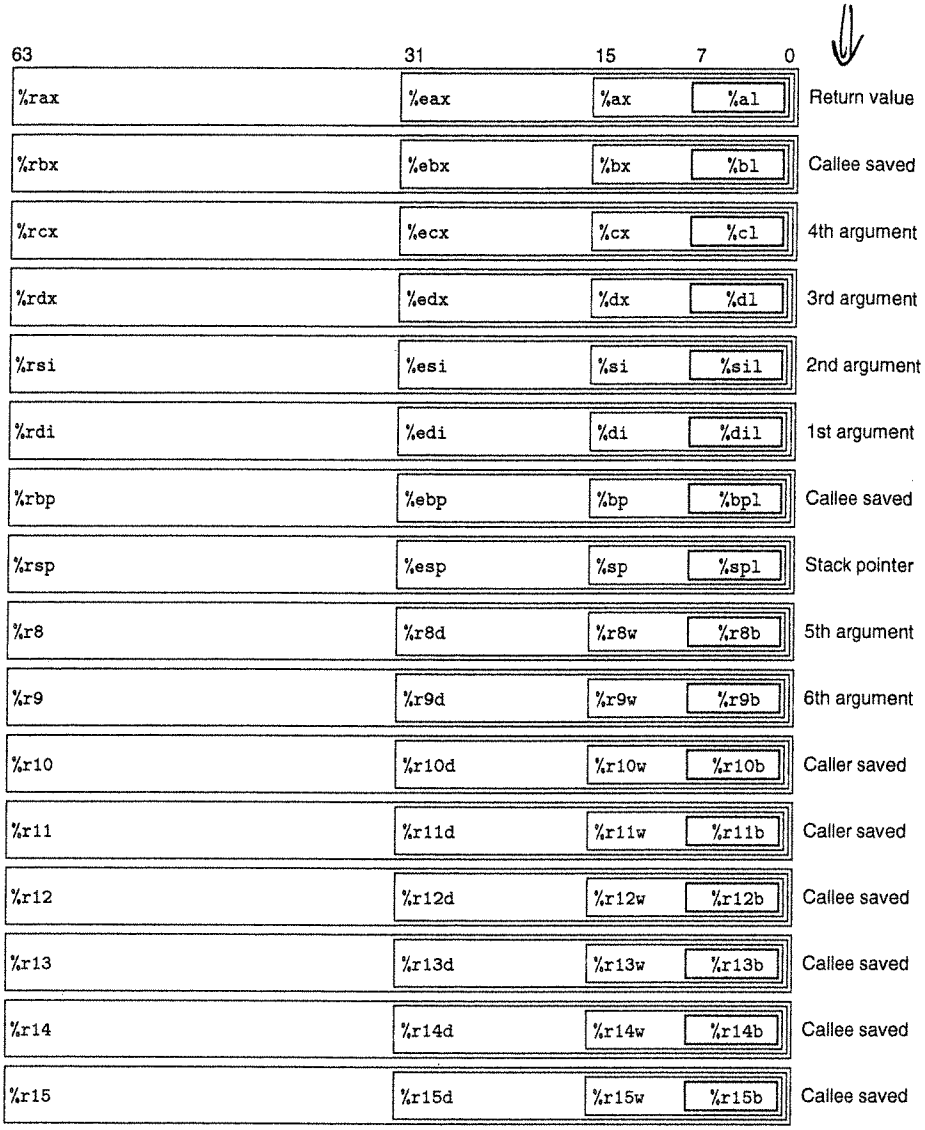

Figure 3.2 **Integer registers.** The low-order portions of all 16 registers can be accessed as byte, word (16-bit), double word (32-bit), and quad word (64-bit) quantities.

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.

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

Figure 3.4 **Simple data movement instructions.**

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

Figure 3.5 **Zero-extending data movement instructions.** These instructions have a register or memory location as the source and a register as the destination.

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

Figure 3.6 **Sign-extending data movement instructions.** The MOVS instructions have a register or memory location as the source and a register as the destination. The `cltq` instruction is specific to registers %eax and %rax.

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

Figure 3.8 **Push and pop instructions.**

Instruction		Effect	Description
`imulq`	S	R[%rdx]:R[%rax] $\leftarrow$ $S \times$ R[%rax]	Signed full multiply
`mulq`	S	R[%rdx]:R[%rax] $\leftarrow$ $S \times$ R[%rax]	Unsigned full multiply
`cqto`		R[%rdx]:R[%rax] $\leftarrow$ SignExtend(R[%rax])	Convert to oct word
`idivq`	S	R[%rdx] $\leftarrow$ R[%rdx]:R[%rax] mod S; R[%rax] $\leftarrow$ R[%rdx]:R[%rax] $\div S$	Signed divide
`divq`	S	R[%rdx] $\leftarrow$ R[%rdx]:R[%rax] mod S; R[%rax] $\leftarrow$ R[%rdx]:R[%rax] $\div S$	Unsigned divide

Figure 3.12 **Special arithmetic operations.** These operations provide full 128-bit multiplication and division, for both signed and unsigned numbers. The pair of registers %rdx and %rax are viewed as forming a single 128-bit oct 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	$D \leftarrow D \hat{} S$	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	$D \leftarrow D >>_{\mathrm{A}} k$	Arithmetic right shift
SHR	k, D	$D \leftarrow D >>_{\mathrm{L}} k$	Logical right shift

Figure 3.10 **Integer arithmetic operations.** The load effective address (`leaq`) instruction is commonly used to perform simple arithmetic. The remaining ones are more standard unary or binary operations. We use the notation $>>_{\mathrm{A}}$ and $>>_{\mathrm{L}}$ to denote arithmetic and logical right shift, respectively. Note the nonintuitive ordering of the operands with ATT-format assembly code.

CF: Carry flag. The most recent operation generated a carry out of the most significant bit. Used to detect overflow for unsigned operations.

ZF: Zero flag. The most recent operation yielded zero.

SF: Sign flag. The most recent operation yielded a negative value.

OF: Overflow flag. The most recent operation caused a two's-complement overflow—either negative or positive.

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

Figure 3.13 **Comparison and test instructions.** These instructions set the condition codes without updating any other registers.

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 <=)

Figure 3.14 **The jump instructions.** These instructions jump to a labeled destination when the jump condition holds. Some instructions have "synonyms," alternate names for the same machine instruction.

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