%rbx %bx %b1 Callee save %rcx %cx %c1 4th argum %rdx %edx %dx %d1 3rd argum %rdx %edx %dx %d1 3rd argum %rsi %si %si %si 1st argume %rdi %edi %di %di 1st argume %rbp %bp %bp %bp1 Callee sav %rsp %sp1 Stack point %r8 %r8 %r8 5th argume %r9 %r9d %r9d %r9b 6th argume %r9 %r9b 6th argume %r10 %r10d %r10w %r10b Caller save %r11 %r11d %r11b Callee save %r12 %r12d %r12w %r12b Callee save %r13 %r13d %r14w %r14b Callee save	63	31	15	7	o W
%rcx %ecx %cx %c1 4th argum %rdx %edx %dx %d1 3rd argum %rsi %esi %si %si1 2nd argum %rsi %esi %si %si1 1st argume %rdi %edi %di %di1 1st argume %rbp %bp1 %bp1 Callee sav %rsp %sp1 %sp1 Stack point %r8 %r8d %r8w %r8b 5th argume %r9 %r9d %r9w %r9b 6th argume %r10 %r10d %r10w %r10b Caller save %r11 %r11d %r11w %r10b Caller save %r12 %r12d %r12w %r12b Callee save %r13 %r13d %r13w %r14b Callee save	%rax	%eax	%ax	%al	Return value
%rdx %edx %dx %d1 3rd argum %rsi %esi %si %si1 2nd argum %rdi %edi %di %di1 1st argume %rbp %bp %bp1 Callee sav %rsp %sp1 Stack point %rs %rsd %rsb 5th argume %rs %rsd %rsb 5th argume %rs %rsd %rsb 6th argume %rs %rsd %rsb 6th argume %rs %rsd %rsb 6th argume %rsd %rsd %rsd %rs	%rbx	%ebx	%bx	%bl	Callee saved
%rsi %esi %si %sil 2nd argum %rdi %edi %di %dil 1st argume %rbp %bp %bpl Callee sav %rsp %sp %spl Stack point %r8 %r8d %r8w %r8b 5th argume %r9 %r9d %r9w %r9b 6th argume %r10 %r10d %r10w %r10b Caller save %r11 %r11d %r11w %r11b Callee save %r12 %r12d %r12w %r12b Callee save %r13 %r13d %r13w %r13b Callee save %r14 %r14d %r14w %r14b Callee save	%rcx	%есх	%сх	%cl	4th argument
%rdi %edi %di %dil 1st argume %rbp %bp %bp %bpl Callee save %rsp %sp %spl Stack point %r8 %r8d %r8w %r8b 5th argume %r9 %r9d %r9w %r9b 6th argume %r10 %r1od %r1ow %r1ob Caller save %r11 %r11d %r11w %r11b Callee save %r12 %r12d %r12w %r12b Callee save %r13 %r13d %r13w %r13b Callee save %r14 %r14d %r14w %r14b Callee save	%rdx	%edx	%dx	%dl	3rd argument
%rbp %bp %bpl Callee save %rsp %sp %spl Stack point %rsp %spl Stack point %rsp %spl Stack point %rsp %spl Sth argume %rsp %rsp %rsp %rsp %rsp Sth argume %rsp %rsp %rsp %rsp %rsp %rsp <tr< td=""><td>%rsi</td><td>%esi</td><td>%si</td><td>%sil</td><td>2nd argument</td></tr<>	%rsi	%esi	%si	%sil	2nd argument
%rsp %sp %sp1 Stack point %r8 %r8d %r8w %r8b 5th argume %r9 %r9d %r9w %r9b 6th argume %r10 %r10d %r10w %r10b Caller save %r11 %r11d %r11w %r11b Caller save %r12 %r12d %r12w %r12b Callee save %r13 %r13d %r13w %r13b Callee save %r14 %r14d %r14w %r14b Callee save	%rdi	%edi	%di	%dil	1st argument
%r8 %r8d %r8w %r8b 5th argume %r9 %r9d %r9w %r9b 6th argume %r10 %r10d %r10w %r10b Caller save %r11 %r11d %r11w %r11b Caller save %r12 %r12d %r12w %r12b Callee save %r13 %r13d %r13w %r14b Callee save %r14 %r14d %r14w %r14b Callee save	%rbp	%ebp	%bp	%bpl	Callee saved
%r9 %r9d %r9w %r9b 6th argume %r10 %r10d %r10w %r10b Caller save %r11 %r11d %r11w %r11b Caller save %r12 %r12d %r12w %r12b Callee save %r13 %r13d %r13w %r13b Callee save %r14 %r14d %r14w %r14b Callee save	%rsp	%esp	%sp	%spl	Stack pointer
%r10 %r10d %r10w %r10b Caller save %r11 %r11d %r11w %r11b Caller save %r12 %r12d %r12w %r12b Callee save %r13 %r13d %r13w %r13b Callee save %r14 %r14d %r14w %r14b Callee save	%r8	%r8d	%r8w	%r8b	5th argument
%r11 %r11d %r11w %r11b Caller save %r12 %r12d %r12w %r12b Callee save %r13 %r13d %r13w %r13b Callee save %r14 %r14d %r14w %r14b Callee save	%r9	%r9d	%r9w	%r9b	6th argument
%r12 %r12d %r12w %r12b Callee save %r13 %r13d %r13w %r13b Callee save %r14 %r14d %r14w %r14b Callee save	(r10	%r10d	%r10w	%r10b	Caller saved
%r13 %r13d %r13w %r13b Callee save %r14 %r14d %r14w %r14b Callee save	(r11	%r11d	%r11w	%r11b	Caller saved
%r14 %r14d %r14w %r14b Callee save	r12	%r12d	%r12w	%r12b	Callee saved
IN THE PROPERTY OF THE PROPERT	r13	%r13d	%r13w	%r13b	Callee saved
%r15 %r15d %r15w %r15b Callee save	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.

			7,000
Туре	Form	Operand value	Name
/ Immediate	\$Imm	Imm	Immediate
2 Register	r_d	$R[r_a]$	Register
B Memory Memory Memory Memory Memory Memory Memory Memory Memory	$Imm (r_a) \\ Imm(r_b) \\ (r_b, r_i) \\ Imm(r_b, r_i) \\ (, r_i, s) \\ Imm(, r_i, s)$	$M[Imm]$ $M[R[r_a]]$ $M[Imm + R[r_b]]$ $M[R[r_b] + R[r_i]]$ $M[Imm + R[r_b] + R[r_i]]$ $M[R[r_i] \cdot s]$ $M[Imm + R[r_i] \cdot s]$	Absolute Indirect Base + displacement Indexed Indexed Scaled indexed
Memory Memory	$(\mathbf{r}_b, \mathbf{r}_i, s)$ $Imm(\mathbf{r}_b, \mathbf{r}_i, s)$	$M[R[r_b] + R[r_i] \cdot s]$ $M[Imm + R[r_b] + R[r_i] \cdot s]$	Scaled indexed Scaled indexed 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	L	Effect	Description
MOV	S, D	$D \leftarrow S$	Move
movb			Move byte
movw			Move word
movl			Move double word Zero
movq			Move quad word
movabsq	I, R	$R \leftarrow I$	Move absolute quad word

Instruction	Effect	Description
pushq S	$R[\%rsp] \leftarrow R[\%rsp] - 8;$	Push quad word
	$M[R[\%rsp]] \leftarrow S$	3
$\mathtt{popq} D$	$D \leftarrow M[R[\%rsp]];$	Pop quad word
	$R[\%rsp] \leftarrow R[\%rsp] + 8$	

Figure 3.8 Push and pop instructions.

Figure 3.4 Simple data movement instructions.

Instruction	Effect	Description
MOVS S, R	$R \leftarrow \text{SignExtend}(S)$	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
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.

Instru	ction	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	$D \leftarrow \neg D$	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	$D \leftarrow D^S$	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

Figure 3.10 Integer arithmetic operations. The load effective address (1eaq) instruction is commonly used to perform simple arithmetic. The remaining ones are more standard unary or binary operations. We use the notation $>>_A$ and $>>_L$ to denote arithmetic and logical right shift, respectively. Note the nonintuitive ordering of the operands with ATT-format assembly code.

Instruct	tion	Effect	Description
imulq mulq	S S	$ \begin{array}{lll} R[\mbox{\ensuremath{\mbox{R[\mbox{\ensuremath{\mbox{\sc R[\mbox{\sc R[\sc R[\mbox{\sc R[\mbox{\sc R[\sc R[\mbox{\sc R[\sc R[\s]\sc R[\sc R[\s]\s]\sc R[\sc R[\s]\sc R[\sc R[\sc R[\sc R[\s]\s]}\s]}}]}]}} }})}}}}$	Signed full multiply Unsigned full multiply
cqto		$R[\rdx]:R[\rdx] \leftarrow SignExtend(R[\rdx])$	Convert to oct word
idivq	S	$\begin{aligned} & R[\%rdx] \; \leftarrow \; & R[\%rdx] : R[\%rax] \; mod \; S; \\ & R[\%rax] \; \leftarrow \; & R[\%rdx] : R[\%rax] \; \div \; S \end{aligned}$	Signed divide
divq	S	$\begin{array}{ll} R[\%rdx] \leftarrow R[\%rdx] : R[\%rax] \bmod S; \\ R[\%rax] \leftarrow R[\%rdx] : R[\%rax] \div S \end{array}$	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	ion	Synonym	Effect	H	TEST FOR	Set condition
sete	D	setz	$D \leftarrow \text{ZF}$		ZF CONS	Equal / zero
setne	D	setnz	$D \leftrightarrow$	1	← ~ ZF	Not equal / not zero
sets	D		$D \uparrow$	1	SF	Negative
setns	D		$D \downarrow$	1	~ SF	Nonnegative
g etg	D	setnle	$D \downarrow$	1	~ (SF ^ OF) & ~ZF	Greater (signed >)
setge	D	setnl	$D \star$	1	← ~ (SF ^ OF)	Greater or equal (signed >=)
set1	D	setnge	D ⋆	↑	SF ~ OF	Less (signed <)
setle	D	setng	$D \star$	T,	← (SF ^ OF) ZF	Less or equal (signed <=)
seta	D	setnbe	D +	1	~ CF & ~ZF	Above (unsigned >)
setae	D	setnb	D .	Ť	~ CF	Above or equal (unsigned >=)
setb	D	setnae	D 💠	1	CF	Below (unsigned <)
setbe	D	setna	D .	T	← CF ZF	Below or equal (unsigned <=)

Figure 3.34 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.

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	o B	Based on	Description
стрь			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	ction	Synonym	Jump condition	Description
dani	Label		1	Direct jump
daj.	*Operand		1	Indirect jump
	Label	jz	ZF	Equal / zero
jne	Label	jnz	~ZF	Not equal / not zero
ی. ب. خ ده ه	Label Lahel		~SF	Negative Nonnegative
ப. ஒ	Label	jnle	~(SF ~ OF) & ~ZF	Greater (signed >)
jge	Label	jpl	~(SF ^ OF)	Test (signed <)
j	Label	jnge ing	$(SF \sim OF) \mid ZF$	Less (signed <) Less or equal (signed <=)
ن. م	Label	jnbe	~CF & ~ZF	Above (unsigned >)
jae	Label	jnb	ÇF.	Above or equal (unsigned >=
jb	Label	jnae	CF.	Below (unsigned <)
jbe	Label	jna	CF ZF	Below or equal (unsigned \-)

Migrare 3.18 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.