Type	From	Operand Value	Name
Immediate	*$Imm*	*Imm*	Immediate
Register	E_a	*R[E_a]*	Register
Memory	*Imm*	*M[Imm]*	Absolute
Memory	*(E_a)*	*M[R[E_b]]*	Absolute
Memory	*Imm(E_b, E_i, s)*	*M[Imm + R[E_b] + (R[E_i] x s)]*	Scaled indexed

More information about operand specifiers can be found on pages 169-170 of the textbook.

3. x64 Instructions

In the following tables,

- "byte" refers to a one-byte integer (suffix **`b`**),
- "word" refers to a two-byte integer (suffix **`w`**),
- "doubleword" refers to a four-byte integer (suffix **`l`**), and
- "quadword" refers to an eight-byte value (suffix **`q`**).

Most instructions, like **`mov`**, use a suffix to show how large the operands are going to be. For example, moving a quadword from **`%rax`** to **`%rbx`** results in the instruction **`movq %rax, %rbx`**. Some instructions, like `ret`, do not use suffixes because there is no need. Others, such as **`movs`** and **`movz`** will use two suffixes, as they convert operands of the type of the first suffix to that of the second. Thus, assembly to convert the byte in **`%al`** to a doubleword in **`%ebx`** with zero-extension would be **`movzbl %al, %ebx`**.

In the tables below, instructions have one suffix unless otherwise stated.

3.1 Data Movement

Instruction		Description	Page #
		Instructions with one suffix	
`mov`	*S, D*	Move source to destination	171
`push`	*S*	Push source onto stack	171
`pop`	*D*	Pop top of stack into destination	171
		Instructions with two suffixes	
`mov`	*S, D*	Move byte to word (sign extended)	171
`push`	*S*	Move byte to word (zero extended)	171
		Instructions with no suffixes	
`cwtl`		Convert word in **`%ax`** to doubleword in **`%eax`** (sign-extended)	182
`cltq`		Convert doubleword in **`%eax`** to quadword in **`%rax`** (sign-extended)	182
`cqto`		Convert quadword in **`%rax`** to octoword in **`%rdx:%rax`**	182

3.2 Arithmetic Operations

Unless otherwise specified, all arithmetic operation instructions have one suffix.

3.2.1 Unary Operations

Instruction		Description	Page #
inc	*D*	Increment by 1	178
dec	*D*	Decrement by 1	178
neg	*D*	Arithmetic negation	178
not	*D*	Bitwise complement	178

3.2.2 Binary Operations

Instruction		Description	Page #
leaq	*S, D*	Load effective address of source into destination	178
add	*S, D*	Add source to destination	178
sub	*S, D*	Subtract source from destination	178
imul	*S, D*	Multiply destination by source	178
xor	*S, D*	Bitwise XOR destination by source	178
or	*S, D*	Bitwise OR destination by source	178
and	*S, D*	Bitwise AND destination by source	178

3.2.3 Shift Operations

Instruction		Description	Page #
sal / shl	*k, D*	Left shift destination by *k* bits	179
sar	*k, D*	Arithmetic right shift destination by *k* bits	179
shr	*k, D*	Logical right shift destination by *k* bits	179

3.2.4 Special Arithmetic Operations

Instruction		Description	Page #
imulq	*S*	Signed full multiply of %rax by S Result stored in %rdx:%rax	182

mulq	*S*	Unsigned full multiply of **%rax** by S Result stored in **%rdx:%rax**	182
idivq	*S*	Signed divide **%rdx:%rax** by S Quotient stored in **%rax** Remainder stored in **%rdx**	182
divq	*S*	Unsigned divide **%rdx:%rax** by S Quotient stored in **%rax** Remainder stored in **%rdx**	182

3.3 Comparison and Test Instructions

Comparison instructions also have one suffix.

Instruction		Description	Page #
cmp	S_2, S_1	Set condition codes according to S_1 - S_2	185
test	S_2, S_1	Set condition codes according to S_1 & S_2	185

3.4 Accessing Condition Codes

None of the following instructions have any suffixes.

3.4.1 Conditional Set Instructions

Instruction		Description	Condition Code	Page #
sete / **setz**	*D*	Set if equal/zero	**ZF**	187
setne / **setnz**	*D*	Set if not equal/nonzero	**~ZF**	187
sets	*D*	Set if negative	**SF**	187
setns	*D*	Set if nonnegative	**~SF**	187
setg / **setnle**	*D*	Set if greater (signed)	**~(SF^0F)&~ZF**	187
setge / **setnl**	*D*	Set if greater or equal (signed)	**~(SF^0F)**	187
setl / **setnge**	*D*	Set if less (signed)	**SF^0F**	187
setle / **setng**	*D*	Set if less or equal	**(SF^OF)|ZF**	187
seta / setnbe	*D*	Set if above (unsigned)	**~CF&~ZF**	187
setae / **setnb**	*D*	Set if above or equal (unsigned)	**~CF**	187
setb / **setnae**	*D*	Set if below (unsigned)	**CF**	187
setbe / **setna**	*D*	Set if below or equal (unsigned)	**CF|ZF**	187

3.4.2 Jump Instructions

Instruction		Description	Condition Code	Page #
jmp	*Label*	Jump to label		189
jmp	**Operand*	Jump to specified location		189
je / **jz**	*Label*	Jump if equal/zero	**ZF**	189
jne / **jnz**	*Label*	Jump if not equal/nonzero	**~ZF**	189
js	*Label*	Jump if negative	**SF**	189
jns	*Label*	Jump if nonnegative	**~SF**	189
jg / **jnle**	*Label*	Jump if greater (signed)	**~(SF^0F)&~ZF**	189
jge / **jnl**	*Label*	Jump if greater or equal (signed)	**~(SF^0F)**	189
jl / **jnge**	*Label*	Jump if less (signed)	**SF^0F**	189
jle / **jng**	*Label*	Jump if less or equal	**(SF^OF)|ZF**	189
ja / **jnbe**	*Label*	Jump if above (unsigned)	**~CF&~ZF**	189
jae / **jnb**	*Label*	Jump if above or equal (unsigned)	**~CF**	189
jb / **jnae**	*Label*	Jump if below (unsigned)	**CF**	189
jbe / **jna**	*Label*	Jump if below or equal (unsigned)	**CF|ZF**	189

3.4.3 Conditional Move Instructions

Conditional move instructions do not have any suffixes, but their source and destination operands must have the same size.

Instruction		Description	Condition Code	Page #
cmove / **cmovz**	*S, D*	Move if equal/zero	**ZF**	206
cmovne / **cmovnz**	*S, D*	Move if not equal/nonzero	**~ZF**	206
cmovs	*S, D*	Move if negative	**SF**	206
cmovns	*S, D*	Move if nonnegative	**~SF**	206
cmovg / **cmovnle**	*S, D*	Move if greater (signed)	**~(SF^0F)&~ZF**	206
cmovge / **cmovnl**	*S, D*	Move if greater or equal (signed)	**~(SF^0F)**	206
cmovl / **cmovnge**	*S, D*	Move if less (signed)	**SF^0F**	206
cmovle / **cmovng**	*S, D*	Move if less or equal	**(SF^OF)|ZF**	206
cmova / **cmovnbe**	*S, D*	Move if above (unsigned)	**~CF&~ZF**	206
cmovae / **cmovnb**	*S, D*	Move if above or equal (unsigned)	**~CF**	206
cmovb / **cmovnae**	*S, D*	Move if below (unsigned)	**CF**	206
cmovbe / **cmovna**	*S, D*	Move if below or equal (unsigned)	**CF|ZF**	206

3.5 Procedure Call Instruction

Procedure call instructions do not have any suffixes.

Instruction	Description	Page #
`call` *Label*	Push return address and jump to label	221
`call` **Operand*	Push return address and jump to specified location	221
`leave`	Set **`%rsp`** to **`%rbp`**, then pop top of stack into **`%rbp`**	221
`ret`	Pop return address from stack and jump there	221

4. Coding Practices

4.1 Commenting

Each function you write should have a comment at the beginning describing what the function does and any arguments it accepts. In addition, we strongly recommend putting comments alongside your assembly code stating what each set of instructions does in pseudocode or some higher level language. Line breaks are also helpful to group statements into logical blocks for improved readability.

4.2 Arrays

Arrays are stored in memory as contiguous blocks of data. Typically an array variable acts as a pointer to the first element of the array in memory. To access a given array element, the index value is multiplied by the element size and added to the array pointer. For instance, if `arr` is an array of `int`s, the statement:

```
arr[i] = 3;
```

can be expressed in x86-64 as follows (assuming the address of `arr` is stored in `%rax` and the index `i` is stored in `%rcx`):

```
movq $3, (%rax, %rcx, 8)
```

More information about arrays can be found on pages 232-241 of the textbook.