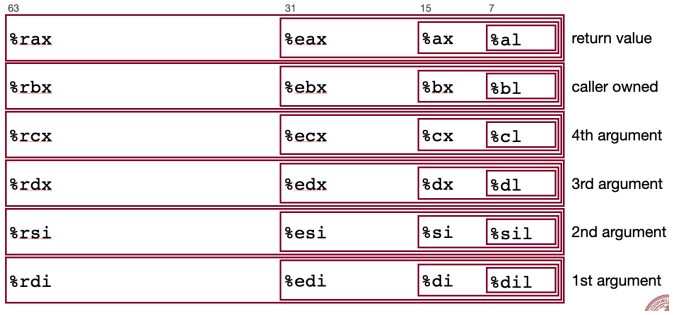
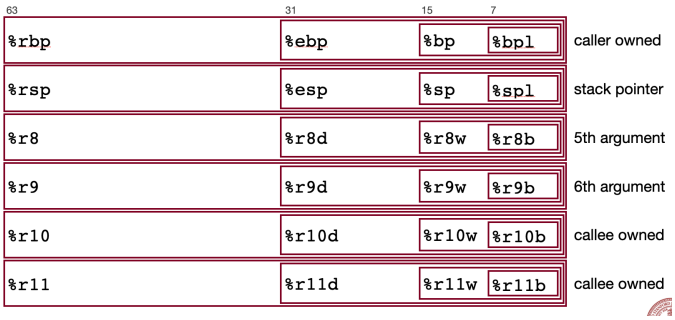
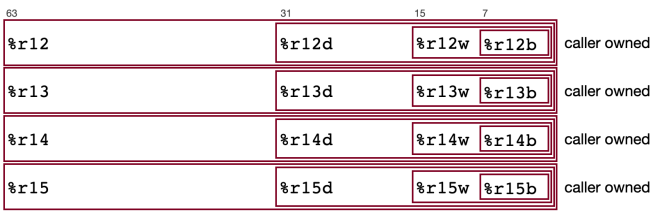
Type	Form	Operand value	Name
Immediate	$\$Imm$	Imm	Immediate
Register	r_a	$R[r_a]$	Register
Memory	Imm	$M[Imm]$	Absolute
Memory	(r_a)	$M[R[r_a]]$	Indirect
Memory	$Imm(r_b)$	$M[Imm + R[r_b]]$	Base + displacement
Memory	(r_b, r_i)	$M[R[r_b] + R[r_i]]$	Indexed
Memory	$Imm(r_b, r_i)$	$M[Imm + R[r_b] + R[r_i]]$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







The last column designates the standard programming conventions — we will get to that later, but it denotes how registers manage the stack, passing function arguments, returning from function calls, and storing local and temporary data.

- Because of its 16-bit origins, Intel uses "word" to mean 16-bits (two bytes)
- 32-bit words are referred to as "double words" ("`l`" suffix)
- 64-bit quantities are referred to as "quad words" ("`q`" suffix)
- This table shows the x86 primitive data types of C (Figure 3.1 in the textbook)

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

- Notice:
  - Pointers are 8-byte quad words
  - The suffixes will become important very soon

```
mov____ %eax, (%rsp)

mov____ (%rax), %dx

mov____ $0xFF, %bl

mov____ (%rsp,%rdx,4), %dl

mov____ (%rdx), %rax

mov____ %dx, (%rax)
```

Answers:

Instruction	Description
`movl %eax, (%rsp)`	copy 4 bytes from %eax into memory at (%rsp)
`movw (%rax), %dx`	copy 2 bytes from memory at (%rax) to %dx
`movb $0xFF, %bl`	set %bl to hold the 1-byte value $0xFF
`movb (%rsp,%rdx,4), %dl`	copy 1 byte from memory at (%rsp + 4*%rdx) to %dl
`movq (%rdx), %rax`	copy 8 bytes from memory at (%rdx) to %rax
`movw %dx, (%rax)`	copy 2 bytes from %dx to memory at (%rax)