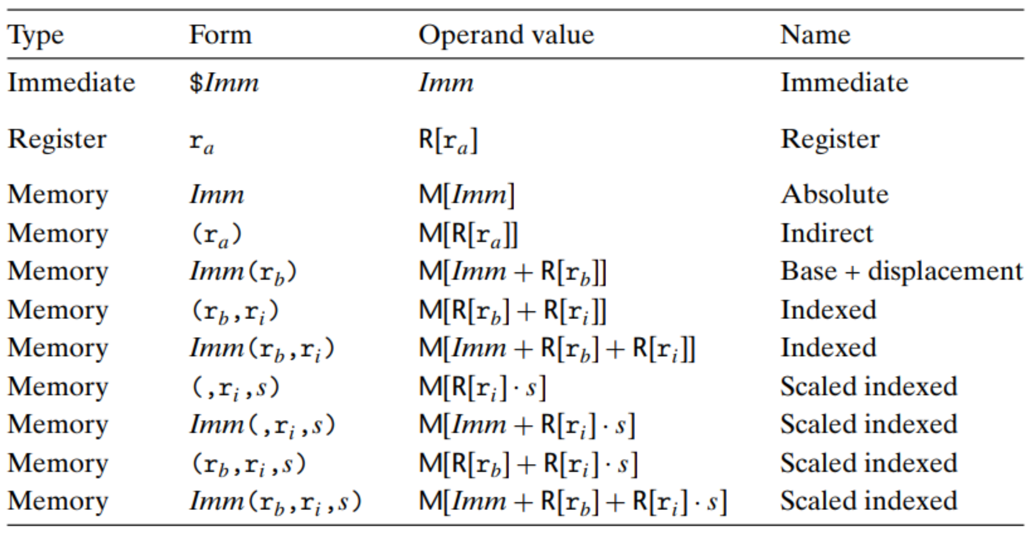
X86 大杂烩

X86的操作数



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.

X86一般采用了INTEL的格式

- Intel代码用不同的方式来描述内存中的位置,例如是‘QWORD PTR [rbx]’而不是 ‘(%rbx)’。
- INTEL格式的目的操作数在前，源操作数在后。

程序员可见状态

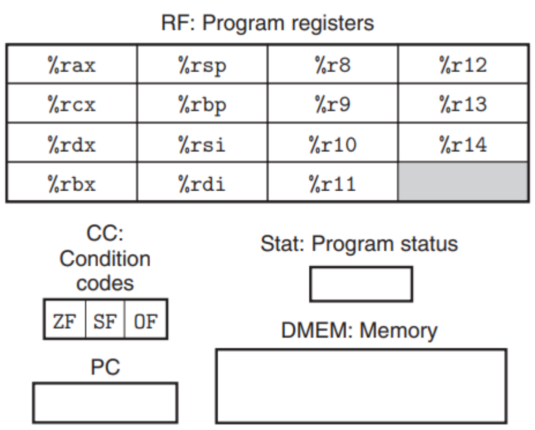


Figure 4.1
Y86-64 programmer-visible state. As with x86-64, programs for Y86-64 access and modify the program registers, the condition codes, the program counter (PC), and the memory. The status code indicates whether the program is running normally or some special event has occurred.

调用约定

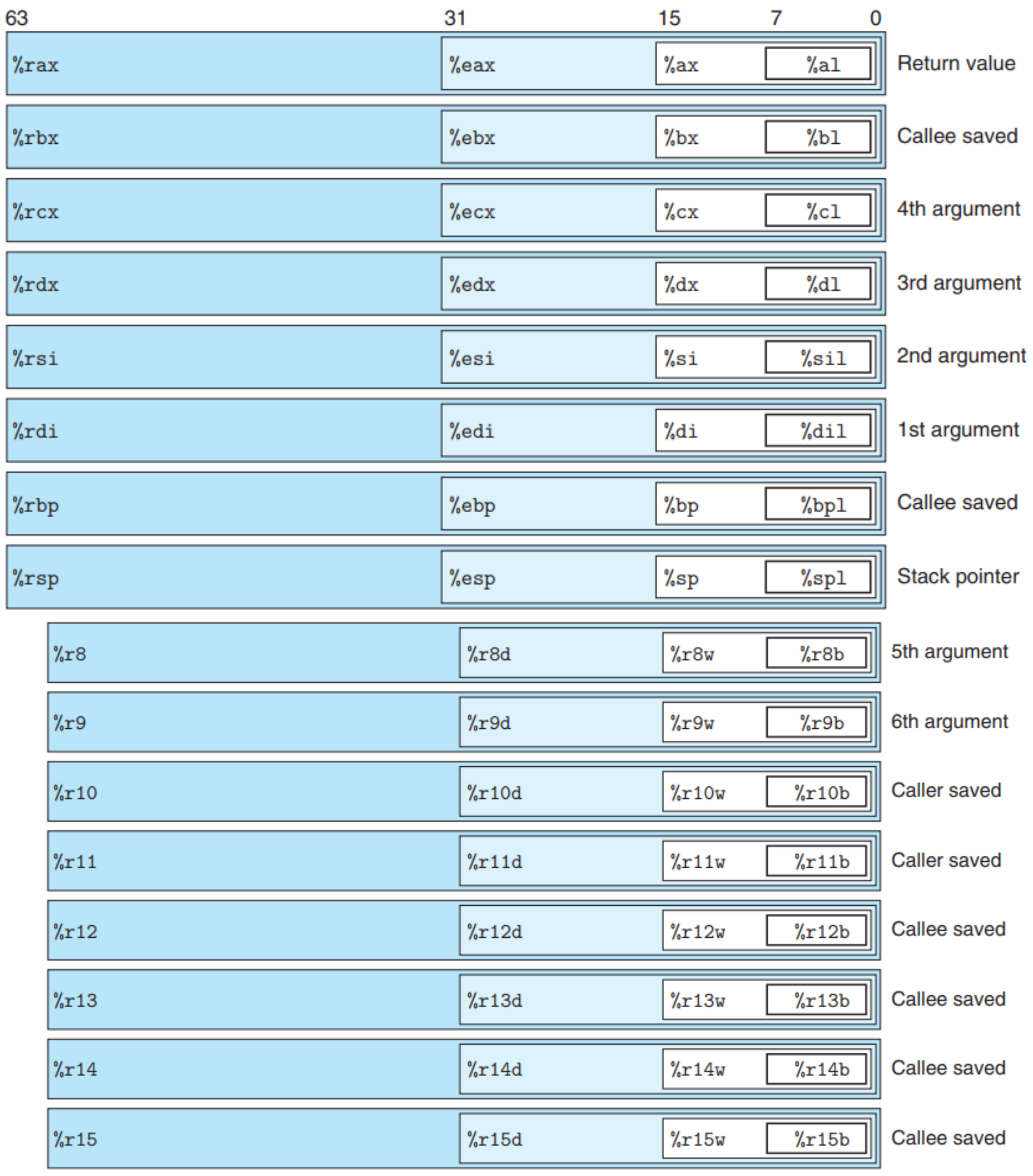


Figure 3.2 **Integer registers.** The low-order portions of all 16 registers can be accessed as byte, word (16-bit), double word (32-bit), and quad word (64-bit) quantities.

Stack

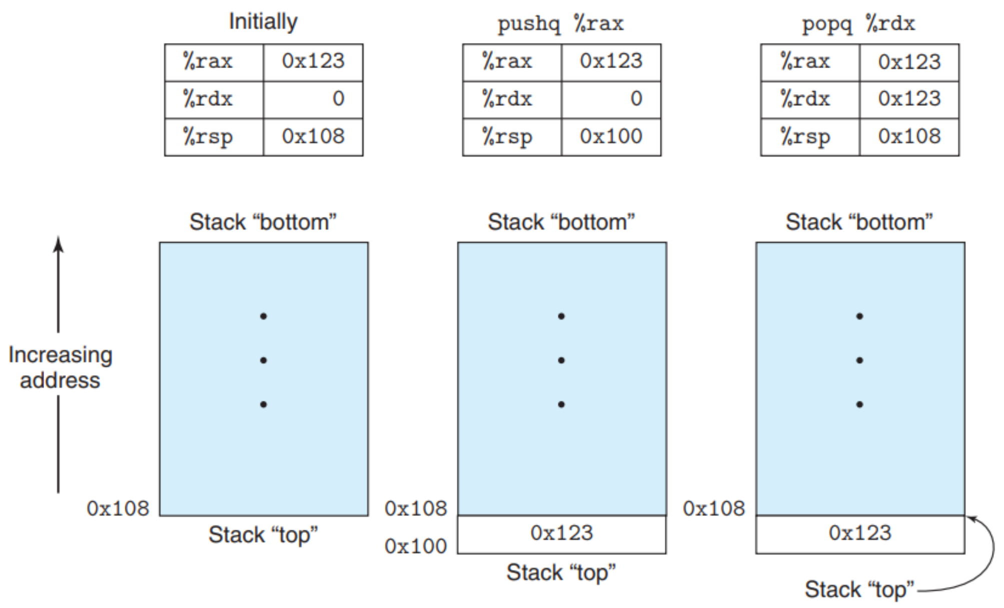


Figure 3.9 **Illustration of stack operation.** By convention, we draw stacks upside down, so that the “top” of the stack is shown at the bottom. With x86-64, stacks grow toward lower addresses, so pushing involves decrementing the stack pointer (register %rsp) and storing to memory, while popping involves reading from memory and incrementing the stack pointer.

Therefore, the behavior of the instruction `pushq %rbp` is equivalent to that of the pair of instructions

```
subq $8,%rsp            Decrement stack pointer
movq %rbp,(%rsp)        Store %rbp on stack
```

Popping a quad word involves reading from the top-of-stack location and then incrementing the stack pointer by 8. Therefore, the instruction `popq %rax` is equivalent to the following pair of instructions:

```
movq (%rsp),%rax        Read %rax from stack
addq $8,%rsp            Increment stack pointer
```

条件码寄存器

unsigned:

$$\begin{gathered}
a-b<0 \longrightarrow Set\ CF=1 \\
a-b=0 \longrightarrow Set\ ZF=1 \\
a-b\leq 0 \longrightarrow CF \mid ZF \\
a-b>0 \longrightarrow \overline{CF \mid ZF} \longrightarrow \overline{CF}\&\overline{ZF} \\
a-b\geq 0 \longrightarrow \overline{CF}\&\overline{ZF} \mid ZF \longrightarrow \overline{CF}
\end{gathered} \tag{1}$$

signed:

$$\begin{gathered}
a-b<0 \longrightarrow (SF \oplus OF) \\
a-b=0 \longrightarrow Set\ ZF=1 \\
a-b\leq 0 \longrightarrow (SF \oplus OF) \mid ZF \\
a-b>0 \longrightarrow \overline{(SF \oplus OF) \mid ZF} \longrightarrow \overline{(SF \oplus OF)}\&\overline{ZF} \\
a-b\geq 0 \longrightarrow \overline{(SF \oplus OF)}\&\overline{ZF} \mid ZF \longrightarrow \overline{(SF \oplus OF)}
\end{gathered} \tag{2}$$

$$\oplus : XOR$$

CF: 进位标志。最近的操作使最高位产生了进位。可用来检查无符号操作的溢出。

ZF: 零标志。最近的操作得出的结果为0。

SF: 符号标志。最近的操作得到的结果为负数。

OF: 溢出标志。最近的操作导致一个补码溢出—正溢出或负溢出。