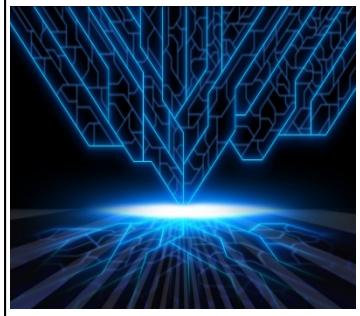
第7讲
数据传送操作



吴海军
南京大学计算机科学与技术系

数据格式

- C语言数据类型及在x86-64中的大小

C声明	Intel数据类型	汇编代码后缀	字节大小
Char	字节	b	1
Short	字	w	2
Int	双字	l	4
Long	四字	q	8
Char *	四字	q	8
float	单精度	s	4
double	双精度	l	8

X86-64通用寄存器的功能

%rax	%eax	%ax	%al	返回值
%rbx	%ebx	%bx	%bl	被调用者保存
%rcx	%ecx	%cx	%cl	第4个参数
%rdx	%edx	%dx	%dl	第3个参数
%rsi	%esi	%si	%sil	第2个参数
%rdi	%edi	%di	%dil	第1个参数
%rbp	%ebp	%bp	%bpl	被调用者保存
%rsp	%esp	%sp	%spl	栈指针
%r8	%r8d	%r8w	%r8b	第5个参数
%r9	%r9d	%r9w	%r9b	第6个参数
%r10	%r10d	%r10w	%r10b	调用者保存
%r11	%r11d	%r11w	%r11b	调用者保存
%r12	%r12d	%r12w	%r12b	被调用者保存
%r13	%r13d	%r13w	%r13b	被调用者保存
%r14	%r14d	%r14w	%r14b	被调用者保存
%r15	%r15d	%r15w	%r15b	被调用者保存

指令可以对这 16 个寄存器的低位字节中存放的不同大小的数据进行操作。

生成1字节和2字节数字的指令会保持剩下的字节不变；生成4字节数字的指令会把高位4个字节置为0。

数据传送指令

- 通用数据传送指令：将数据从一个位置复制到另一个位置的指令。

  MOV：一般传送指令，指令后缀为b、w、l、q。

  MOVABSQ：传送64位的绝对值到寄存器。

  MOVS：符号扩展传送，如movsbw、movswl等

  MOVZ：零扩展传送，如movzwl、movzbl等

  XCHG：数据交换

  PUSHL S：压栈

  POPL D：出栈

- 输入输出指令：IN和OUT：I/O端口与寄存器之间的交换
- 标志传送指令

  PUSHF：将状态寄存器RFLAG内容压栈

  POPF：将栈顶内容传送状态寄存器RFLAG。

传送数据

- 源操作数三种类型：
  - **立即数Immediate**: 整形常量
    - 有‘**$**’前缀，如 `$0x400`, `$-533`
    - 不同的指令允许的立即数值范围不同
  - **寄存器Register**:表示某个通用寄存器的内容
    - 有‘**%**’前缀，如%rax，%r8d，%bx
    - 16个寄存器的低位 1、2、4或8字节中的一个作为操作数
    - 符号r_a 来表示任意寄存器 a, 用引用 R[r_a]来表示它的值。
  - **内存引用**：根据计算出来的有效地址访问某个内存位置。
    - M_b [Addr]表示对存储在内存中从地址 Addr 开始的b 个字节值的引用，为了简便通常省去下标b。

操作数寻址模式

- 有多种不同的寻址模式

类型	格式	操作数值	名称
立即数	$*Imm*	*Imm*	立即数寻址
寄存器	r_a	R[r_a]	寄存器寻址
存储器	*Imm*	M[*Imm*]	绝对寻址
存储器	(r_a)	M[R[r_a]]	间接寻址
存储器	*Imm*(r_b)	M[*Imm*+R[r_b]]	(基址+偏移量)寻址
存储器	(r_b+r_i)	M[R[r_b]+R[r_i]]	变址寻址
存储器	*Imm*(r_b+r_i)	M[*Imm*+R[r_b]+R[r_i]]	变址寻址
存储器	(, r_i,*s*)	M[R[r_i]·*s*]	比例变址寻址
存储器	*Imm*(, r_i,*s*)	M[*Imm*+R[r_i]·*s*]	比例变址寻址
存储器	(r_b, r_i,*s*)	M[R[r_b]+R[r_i] ·*s*]	比例变址寻址
存储器	*Imm*(r_b, r_i,*s*)	M[*Imm*+R[r_b]+R[r_i] ·*s*]	比例变址寻址

多种寻址模式

- rb:基址寄存器，ri：变址寄存器，s：比例因子，取值1、2、4、8。

movl 8(%ebx,%eax,4), %eax

等价

```
imul $4, %eax
addl %ebx, %eax
addl $8, %eax
movl (%eax), %eax
```

体现了CISC的“复杂性”

数据传送指令

	源	目的	Src,Dest	C 类似
Mov{q,l,w,b}	立即数 Imm	Reg	`movl $0x4050,%eax`	`temp = 0x4050;`
		Mem	`movb $-17,(%rsp)`	`*p = -17;`
	寄存器 Reg	Reg	`movw %bp,%sp`	`temp2 = temp1;`
		Mem	`movq %rax,-12(%rbp)`	`*p = temp;`
	存储器 Mem	Reg	`movb (%rdi,%rcx),%al`	`temp = *p;`

不能在一条指令里实现存储器到存储器传送！

指令的后缀、操作数的长度、寄存器的宽度要一致！

数据移动指令

- 在将较小的源值复制到较大的目的时使用。

指令	效果	描述
MOVZ S，R	R←零扩展(S)	以零扩展进行传送
movzbw		将做了零扩展的字节传送到字
movzbl		将做了零扩展的字节传送到双字
movzwl		将做了零扩展的字传送到双字
movzbq		将做了零扩展的字节传送到四字
movzwq		将做了零扩展的字传送到四字

指令	效果	描述
MOVS S，R	R←符号扩展(S)	传送符号扩展的字节
movsbw		将做了符号扩展的字节传送到字
movsbl		将做了符号扩展的字节传送到双字
movswl		将做了符号扩展的字传送到双字
movsbq		将做了符号扩展的字节传送到四字
movswq		将做了符号扩展的字传送到四字
movslq		将做了符号扩展的双字传送到四字
cltq	%rax ←符号扩展(%eax)	把%eax 符号扩展到%rax

数据移动指令

- 数据传送指令如何修改目的寄存器的高位字节

movabsq $0x0011223344556677, %rax	%rax = 0011223344556677
movb $-1，%al	%rax = 0011223344556FF
movw $-1，%ax	%rax = 001122334455FFFF
movl $-1，%eax	%rax = 00000000FFFFFFFF
movq $-1，%rax	%rax = FFFFFFFFFFFFFFFF

- Cltq指令，没有操作数，以寄存器%eax作为源，%rax作为符号扩展结果的目的寄存器。
- 效果与指令movslq %eax, %rax一致，但编码更紧凑。

数据移动指令

- 确定适当的指令后缀
  - mov w %eax, (%rsp)
  - mov w (%rax), %dx
  - mov b $0xFF, %bl
  - mov b (%rsp,%rdx,4), %dl
  - mov q (%rdx), %rax
  - mov w %dx, (%rax)
- 执行下列指令后，寄存器中数值
  - movabsq $0x0011223344556677, %rax
  - movb $0xAA, %dl
  - movb %dl,%al
  - movsbq %dl,%rax
  - movzbq %dl,%rax

%rax = 0011223344556677
%dl = AA
%rax = 0011223344556AA
%rax = FFFFFFFFFFFFFFAA
%rax = 00000000000000AA

寻址运算例子

%eax	0xf000
%ecx	0x100

表达式	地址运算
0x8(%eax)	0xf000 + 0x8=0xf008
(%eax,%ecx)	0xf000 + 0x100=0xf100
(%eax,%ecx,4)	0xf000 + 4*0x100=0xf400
0x80(,%eax,2)	2*0xf000 + 0x80=0x1e080

存储器地址：

0	4	8	12	16	20	24	28	32	36	40	44	48
2	0	1	8	0	7	0	6	0	9	4	5	6

指令	寄存器eax中的数值
movl $12, %eax	12
movl (%eax), %eax	8
movl 12(%eax), %eax	7

存储器操作数的寻址方式

```
int x;
float a[100];
short b[4][4];
char c;
double d[10];
```

d[i]为什么不从540开始？

a[i]的地址如何计算?

104+i×4

i=99时，104+99×4=500

b[i][j]的地址如何计算?

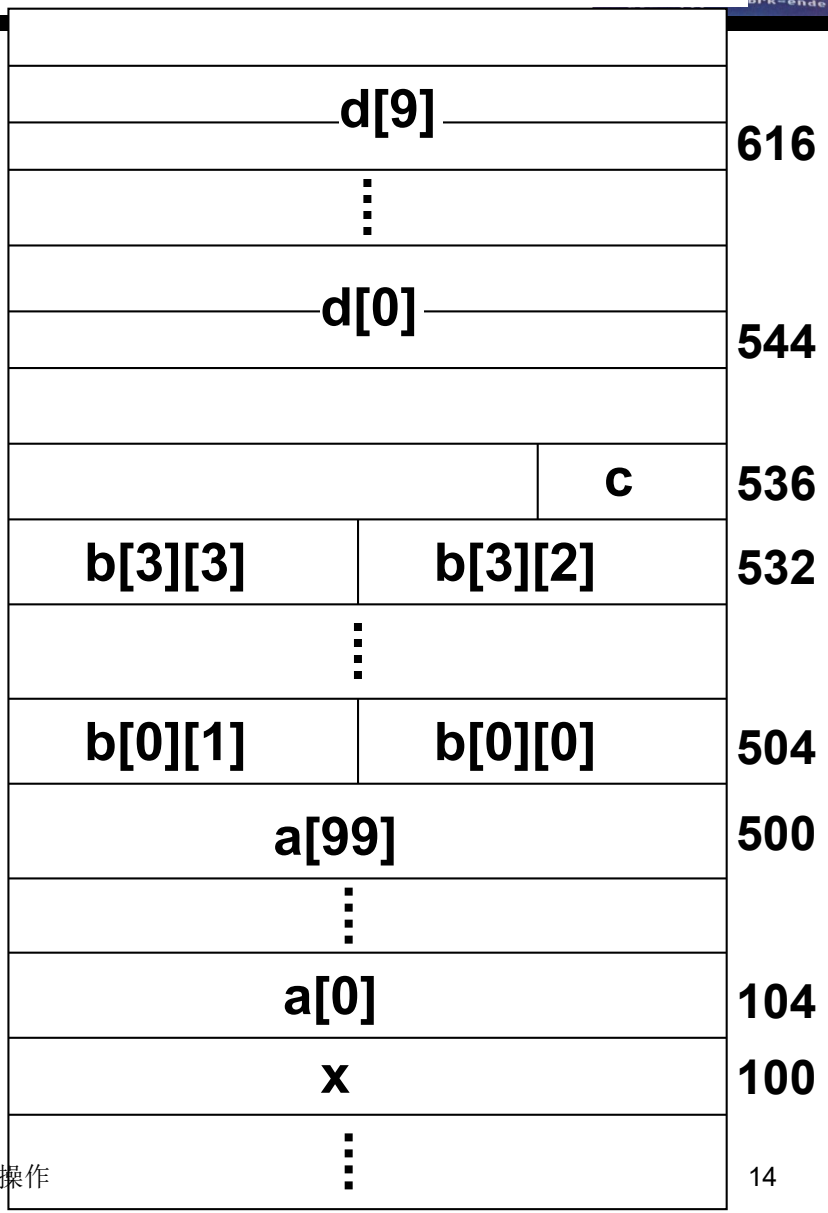
504+i×8+j×2

i=3、j=2时，504+24+4=532

d[i]的地址如何计算?

544+i×8

i=9时，544+9×8=616



存储器操作数的寻址方式

```
int x;
float a[100];
short b[4][4];
char c;
double d[10];
```

各变量应采用什么寻址方式?

x、c：位移 / 基址

a[i]：104+i×4，比例变址+位移

d[i]：544+i×8，比例变址+位移

b[i][j]： 504+i×8+j×2，

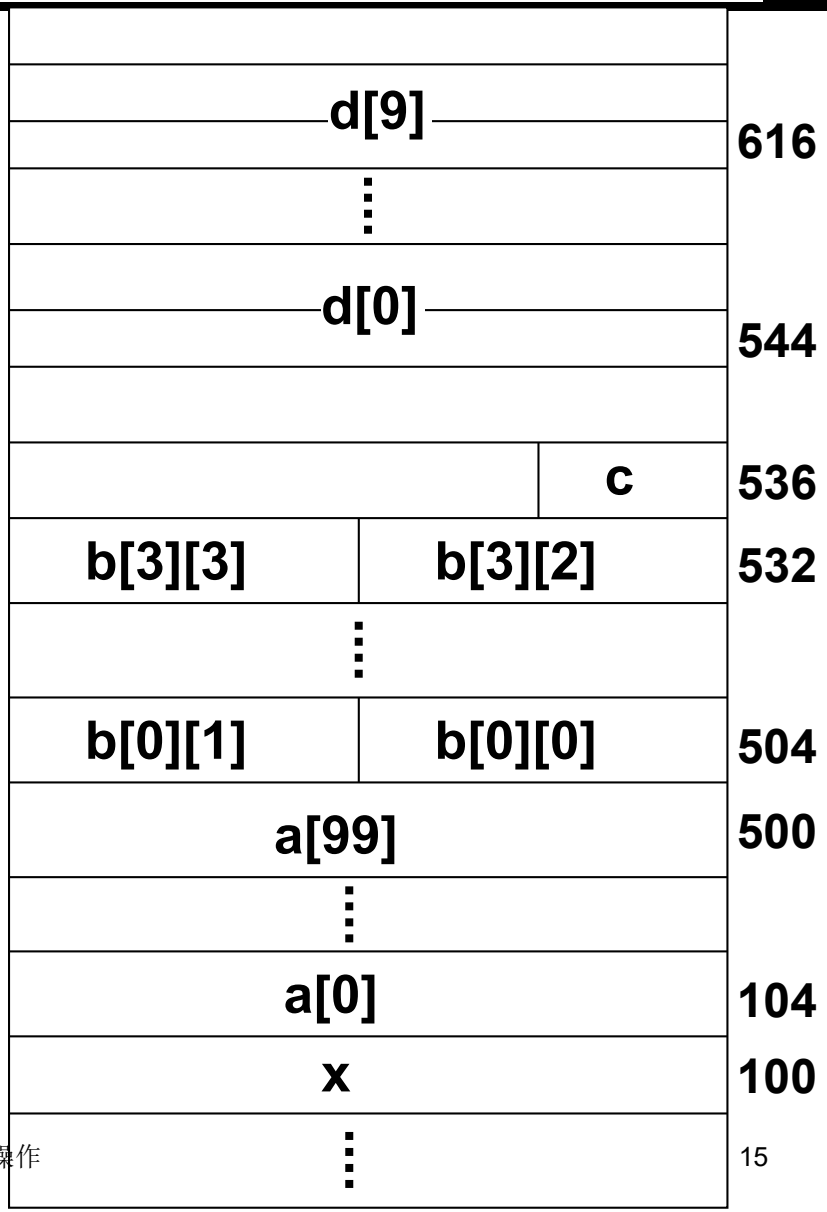
基址+比例变址+位移

将b[i][j]取到AX中的指令可以是：

“movw 504(%ebp,%esi,2), %ax”

其中， i×8在EBP中，j在ESI中，

2为比例因子



x86-64使用寄存器传递参数

- 使用寄存器传递参数
  - 第1个参数 **xp**存放在**%rdi**, 第2个参数**y**存放在**%rsi**
  - 函数通过寄存器%rax 返回数值。

```
long exchg(long *xp, long y)
{
  long x = *xp;
     *xp=y;
   return x;
}
```

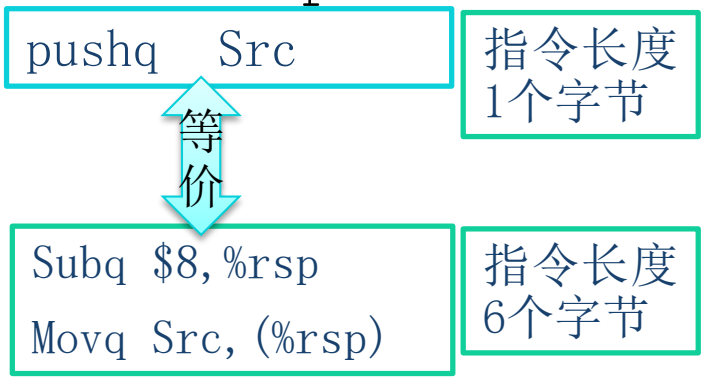
```
exchg:
   movq  (%rdi), %rax
   movq  %rsi, (%rdi)

   ret
```

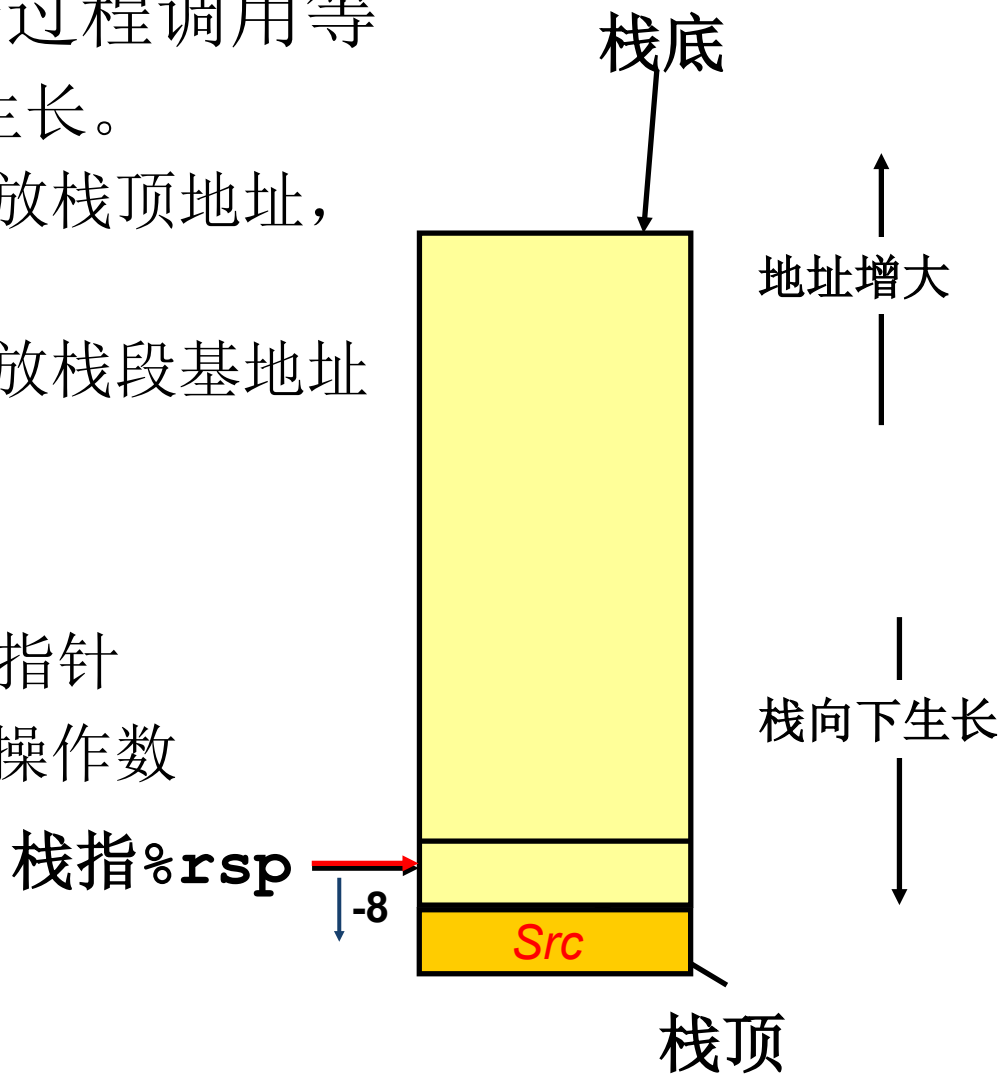
C语言中指针类型其实就是地址。间接引用指针就是将该指针放在一个寄存器中。

局部变量x通常是保存在寄存器中，而不是内存中。 访问寄存器比访 问内存要快得多。

数据传送指令-压栈

- 一块特殊的内存区，用于过程调用等
  - 后进先出、向低地址空间生长。
  - %rsp堆栈指针寄存器：存放栈顶地址，栈的最低地址。
  - %rbp基址指针寄存器：存放栈段基地址
- 压栈push
  - pushq *Src*
  - %rsp 减8 ： 向下移动栈顶指针
  - 在%rsp给出的地址处写入操作数





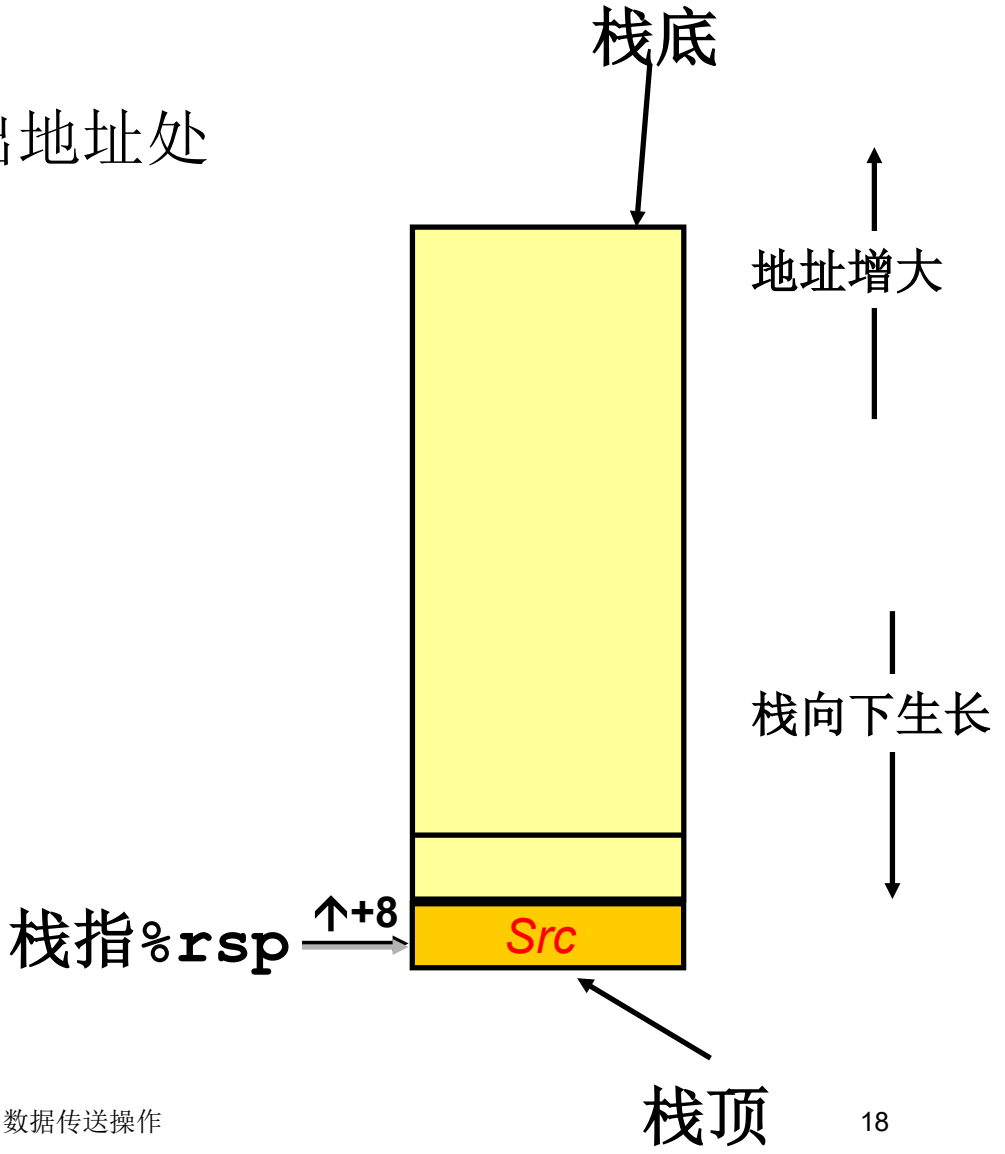
数据传送指令–出栈

- 出栈pop
  - popl *Dest*：从%rsp给出地址处读取操作数
  - *将操作数写到Dest寄存器*
  - %rsp 加8

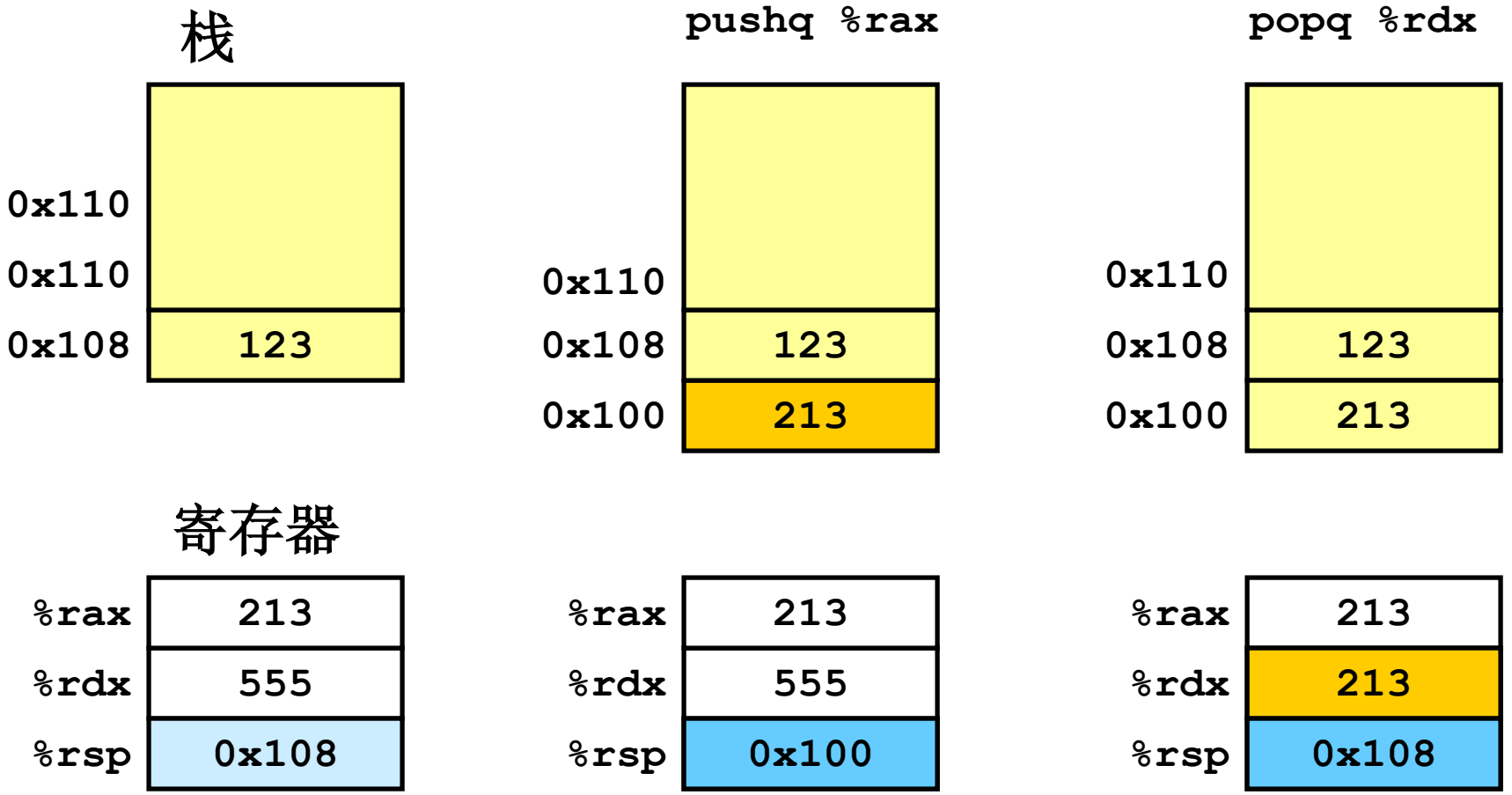
```
Popq Dest
```

等价

```
Movq (%rsp), Dest
addq $8,%rsp
```



栈操作例子


栈
0x110
0x110
0x108
123
pushq %rax
0x110
0x108
0x100
123
213
popq %rdx
0x110
0x108
0x100
123
213
寄存器
%rax
%rdx
%rsp
213
555
0x108
%rax
%rdx
%rsp
213
555
0x100
%rax
%rdx
%rsp
213
213
0x108