



分配9个字节,不可能满足d的每个元素的对齐要求,因为这些元素的地址分别为  $x_d$ 、 $x_d+9$ 、 $x_d+18$  和  $x_d+27$ 。相反,编译器会为结构 S2 分配12个字节,最后3个字节是浪费的空间:

偏移	0	4	8	9	12
内容	i		j		c

这样一来,d 的元素的地址分别为  $x_d$ 、 $x_d+12$ 、 $x_d+24$  和  $x_d+36$ 。只要  $x_d$  是4的倍数,所有的对齐限制就都可以满足了。

 **练习题 3.44** 对下面每个结构声明,确定每个字段的偏移量、结构总的大小,以及在 x86-64 下它的对齐要求:

- struct P1 { int i; char c; int j; char d; };
- struct P2 { int i; char c; char d; long j; };
- struct P3 { short w[3]; char c[3] };
- struct P4 { short w[5]; char \*c[3] };
- struct P5 { struct P3 a[2]; struct P2 t };

 **练习题 3.45** 对于下列结构声明回答后续问题:

```
struct {
    char    *a;
    short   b;
    double  c;
    char    d;
    float   e;
    char    f;
    long    g;
    int     h;
} rec;
```

- 这个结构中所有的字段的字节偏移量是多少?
- 这个结构总的大小是多少?
- 重新排列这个结构中的字段,以最小化浪费的空间,然后再给出重排过的结构的字节偏移量和总的大小。

### 旁注 强制对齐的情况

对于大多数 x86-64 指令来说,保持数据对齐能够提高效率,但是它不会影响程序的行为。另一方面,如果数据没有对齐,某些型号的 Intel 和 AMD 处理器对于有些实现多媒体操作的 SSE 指令,就无法正确执行。这些指令对 16 字节数据块进行操作,在 SSE 单元和内存之间传送数据的指令要求内存地址必须是 16 的倍数。任何试图以不满足对齐要求的地址来访问内存都会导致异常(参见 8.1 节),默认的行为是程序终止。

因此,任何针对 x86-64 处理器的编译器和运行时系统都必须保证分配用来保存可能会被 SSE 寄存器读或写的数据结构的内存,都必须满足 16 字节对齐。这个要求有两个后果:

- 任何内存分配函数(alloca、malloc、calloc 或 realloc)生成的块的起始地址都必须是 16 的倍数。
- 大多数函数的栈帧的边界都必须是 16 字节的倍数。(这个要求有一些例外。)

较近版本的 x86-64 处理器实现了 AVX 多媒体指令。除了提供 SSE 指令的超集,支持 AVX 的指令并没有强制性的对齐要求。