```
    } t1;
    struct {
        int a[2];
        char  *p;
    } t2;
} u_type;
```

你写了一组具有下面这种形式的函数：

```
void get(u_type *up, type *dest) {
    *dest =  expr;
}
```

这组函数有不一样的访问表达式 *expr*，而且根据 *expr* 的类型来设置目的数据类型 *type*。然后再检查编译这些函数时产生的代码，看看它们是否与你预期的一样。

假设在这些函数中，up 和 dest 分别被加载到寄存器%rdi 和%rsi 中。填写下表中的数据类型 *type*，并用 1～3 条指令序列来计算表达式，并将结果存储到 dest 中。

expr	*type*	代码
up->t1.u	long	movq(%rdi),%rax movq %rax,(%rsi)
up->t1.v		
&up->t1.w		
up->t2.a		
up->t2.a[up->t1.u]		
*up->t2.p		

3.9.3　数据对齐

许多计算机系统对基本数据类型的合法地址做出了一些限制，要求某种类型对象的地址必须是某个值 K(通常是 2、4 或 8)的倍数。这种对齐限制简化了形成处理器和内存系统之间接口的硬件设计。例如，假设一个处理器总是从内存中取 8 个字节，则地址必须为 8 的倍数。如果我们能保证将所有的 double 类型数据的地址对齐成 8 的倍数，那么就可以用一个内存操作来读或者写值了。否则，我们可能需要执行两次内存访问，因为对象可能被分放在两个 8 字节内存块中。

无论数据是否对齐，x86-64 硬件都能正确工作。不过，Intel 还是建议要对齐数据以提高内存系统的性能。对齐原则是任何 K 字节的基本对象的地址必须是 K 的倍数。可以看到这条原则会得到如下对齐：