```
8     17:   48 89 4c d0 10          mov    %rcx,0x10(%rax,%rdx,8)
9     1c:   c3                      retq
```

运用你的逆向工程技术，推断出下列内容：

A. CNT 的值。

B. 结构 a_struct 的完整声明。假设这个结构中只有字段 idx 和 x，并且这两个字段保存的都是有符号值。

** 3.70 考虑下面的联合声明：

```
1   union ele {
2       struct {
3           long *p;
4           long y;
5       } e1;
6       struct {
7           long x;
8           union ele *next;
9       } e2;
10  };
```

这个声明说明联合中可以嵌套结构。

下面的函数(省略了一些表达式)对一个链表进行操作，链表是以上述联合作为元素的：

```
1   void proc (union ele *up) {
2       up->_____ = *(_____) - _____;
3   }
```

A. 下列字段的偏移量是多少(以字节为单位)：

```
e1.p      _____
e1.y      _____
e2.x      _____
e2.next   _____
```

B. 这个结构总共需要多少个字节？

C. 编译器为 proc 产生下面的汇编代码：

```
    void proc (union ele *up)
    up in %rdi
1   proc:
2     movq    8(%rdi), %rax
3     movq    (%rax), %rdx
4     movq    (%rdx), %rdx
5     subq    8(%rax), %rdx
6     movq    %rdx, (%rdi)
7     ret
```

在这些信息的基础上，填写 proc 代码中缺失的表达式。**提示**：有些联合引用的解释可以有歧义。当你清楚引用指引到哪里的时候，就能够澄清这些歧义。只有一个答案，不需要进行强制类型转换，且不违反任何类型限制。

* 3.71 写一个函数 good_echo，它从标准输入读取一行，再把它写到标准输出。你的实现应该对任意长度的输入行都能工作。可以使用库函数 fgets，但是你必须确保即使当输入行要求比你已经为缓冲区分配的更多的空间时，你的函数也能正确地工作。你的代码还应该检查错误条件，要在遇到错误条件时返回。参考标准 I/O 函数的定义文档[45, 61]。

** 3.72 图 3-54a 给出了一个函数的代码，该函数类似于函数 vfunct(图 3-43a)。我们用 vfunct 来说明过帧指针在管理变长栈帧中的使用情况。这里的新函数 aframe 调用库函数 alloca 为局部数组 p 分配空间。alloca 类似于更常用的函数 malloc，区别在于它在运行时栈上分配空间。当正在执行的过程返回时，该空间会自动释放。

图 3-54b 给出了部分的汇编代码，建立帧指针，为局部变量 i 和 p 分配空间。非常类似于