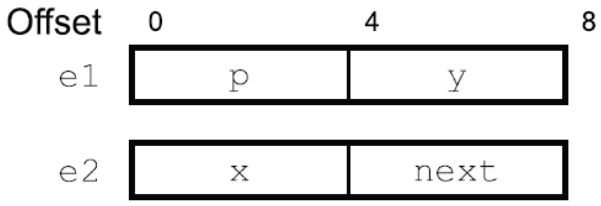
A. The layout of the union is shown in the table that follows. As the table illustrates, the union can have either its “e1” interpretation (having fields e1.p and e1.y), or it can have its “e2” interpretation (having fields e2.x and e2.next).



B. It uses 8 bytes.

C. As always, we start by annotating the assembly code. In our annotations, we show multiple possible interpretations for some of the instructions, and then indicate which interpretation later gets discarded. For example, line 2 could be interpreted as either getting element e1.y or e2.next. In line 3, we see that the value gets used in an indirect memory reference, for which only the second interpretation of line 2 is possible.

```
1 movl 8(%ebp), %edx Get up
2 movl 4(%edx), %ecx up->e1.y (no) or up->e2.next
3 movl (%ecx), %eax up->e2.next->e1.p or up->e2.next->e2.x (no)
4 movl (%eax), %eax *(up->e2.next->e1.p)
5 subl (%edx), %eax *(up->e2.next->e1.p) - up->e2.x
6 movl %eax, 4(%ecx) Store in up->e2.next->e1.y
```

From this, we can generate C code as follows:

```
1 void proc (union ele *up)
2 {
3 up->e2.next->e1.y = *(up->e2.next->e1.p) - up->e2.x;
4 }
```