A. 可以看到这段代码使用的是跳转到中间翻译方法，在第 3 行使用了 jmp 指令。

B. 下面是原始的 C 代码：

```
long fun_a(unsigned long x) {
    long val = 0;
    while (x) {
        val ^= x;
        x >>= 1;
    }
    return val & 0x1;
}
```

C. 这个代码计算参数 x 的奇偶性。也就是，如果 x 中有奇数个 1，就返回 1，如果有偶数个 1，就返回 0。

3.27 这道练习题意在加强你对如何实现循环的理解。

```
long fact_for_gd_goto(long n)
{
    long i = 2;
    long result = 1;
    if (n <= 1)
        goto done;
loop:
    result *= i;
    i++;
    if (i <= n)
        goto loop;
done:
    return result;
}
```

3.28 这个问题比练习题 3.26 要难一些，因为循环中的代码更复杂，而整个操作也不那么熟悉。

A. 以下是原始的 C 代码：

```
long fun_b(unsigned long x) {
    long val = 0;
    long i;
    for (i = 64; i != 0; i--) {
        val = (val << 1) | (x & 0x1);
        x >>= 1;
    }
    return val;
}
```

B. 这段代码是用 guarded-do 变换生成的，但是编译器发现因为 i 初始化成了 64，所以一定会满足测试 $i \neq 0$，因此初始的测试是没必要的。

C. 这段代码把 x 中的位反过来，创造一个镜像。实现的方法是：将 x 的位从左往右移，然后再填入这些位，就像是把 val 从右往左移。

3.29 我们把 for 循环翻译成 while 循环的规则有些过于简单——这是唯一需要特殊考虑的方面。

A. 使用我们的翻译规则会得到下面的代码：

```
/* Naive translation of for loop into while loop */
/* WARNING: This is buggy code */
long sum = 0;
long i = 0;
while (i < 10) {
    if (i & 1)
        /* This will cause an infinite loop */
        continue;
    sum += i;
    i++;
}
```