1. 已知 *C* 代码如下：

```
long dw_loop(long x) {
    long y = x*x;
    long *p = &x;
    long n = 2*x;
    do {
        x += y;
        (*p)++;
        n--;
    } while (n > 0);
    return x;
}
```

GCC 产生的汇编代码如下：

```
    long dw_loop(long x)
    x initially in %rdi
1   dw_loop:
2     movq    %rdi, %rax
3     movq    %rdi, %rcx
4     imulq   %rdi, %rcx
5     leaq    (%rdi,%rdi), %rdx
6   .L2:
7     leaq    1(%rcx,%rax), %rax
8     subq    $1, %rdx
9     testq   %rdx, %rdx
10    jg      .L2
11    rep; ret
```

A. 哪些寄存器用来存放程序值*x*、*y* 和 *n*?

> x: %rdi
> y: %rcx
> z: %rdx

B. 编译器如何消除指针变量 *p* 和表达式 *(*p)++* 隐含的指针间接引用的需求?

> 编译器认为指针变量 *p* 在这段代码中总是指向变量 *x* 的，所以直接将 *(*p)++* 改写成对 *x* 的操作，并通过第 7 行的 *leaq* 指令同时实现 *x++* 和 *x += y*

C. 对汇编代码添加一些注释，描述程序的操作，类似于教材图 *3-19c* 中所示的那样。

3.26

2. 函数 *fun_a* 有如下整体结构：

```
long fun_a(unsigned long x) {
    long val = 0;
    while ( ... ) {
        .
        .
        .
    }
    return ...;
}
```

GCC 编译器产生如下汇编代码：

```
    long fun_a(unsigned long x)
    x in %rdi
1   fun_a:
2     movl    $0, %eax
3     jmp     .L5
4   .L6:
5     xorq    %rdi, %rax
6     shrq    %rdi            Shift right by 1
7   .L5:
8     testq   %rdi, %rdi
9     jne     .L6
10    andl    $1, %eax
11    ret
```

逆向工程这段代码的操作，然后完成下面作业：

A. 确定这段代码使用的循环翻译方法

> 跳转至中间(*jump to middle*)翻译方法

B. 根据汇编代码版本填写 *C* 代码中缺失的部分。

C. 用自然语言描述这个函数是计算什么的。

> 判断 *x* 的二进制表示中有奇数或偶数个 1。如果有奇数个 1，则返回 1；如果有偶数个 1，则返回 0

3. 函数 *fun_b* 有如下整体结构：

```
long fun_b(unsigned long x) {
    long val = 0;
    long i;
    for ( ... ; ... ; ... ) {
        .
        .
        .
    }
    return val;
}
```

GCC 产生的汇编代码如下：

```
    long fun_b(unsigned long x)
    x in %rdi
1   fun_b:
2     movl    $64, %edx
3     movl    $0, %eax
4   .L10:
5     movq    %rdi, %rcx

6     andl    $1, %ecx
7     addq    %rax, %rax
8     orq     %rcx, %rax
9     shrq    %rdi            Shift right by 1
10    subq    $1, %rdx
11    jne     .L10
12    rep; ret
```

逆向工程这段代码的操作，然后完成下面作业：

A. 根据汇编代码版本填写 *C* 代码中缺失的部分。

B. 解释循环前为什么没有初始测试也没有初始转跳到循环内部的测试部分。

> 因为 *i* 被初始化为 64，编译器发现其一定满足测试 *i != 0*，因此优化掉了初始测试

C. 用自然语言描述这个函数是计算什么的。

> 将 *x* 的二进制位反转

1. 对于一个通用结构的 *C* 函数 *switcher*：

```
void switcher(long a, long b, long c, long *dest)
{
    long val;
    switch(a) {
    case __________:        /* Case A */
        c = __________;
        /* Fall through */
    case __________:        /* Case B */
        val = __________;
        break;
    case __________:        /* Case C */
    case __________:        /* Case D */
        val = __________;
        break;
    case __________:        /* Case E */
        val = __________;
        break;
    default:
        val = __________;
    }
    *dest = val;
}
```

GCC 产生下面所示的汇编代码和转跳表：

```
    void switcher(long a, long b, long c, long *dest)
    a in %rdi, b in %rsi, c in %rdx, dest in %rcx
1   switcher:
2     cmpq    $7, %rdi
3     ja      .L2
4     jmp     *.L4(,%rdi,8)
5     .section        .rodata
6   .L7:
7     xorq    $15, %rsi
8     movq    %rsi, %rdx
9   .L3:
10    leaq    112(%rdx), %rdi
11    jmp     .L6
12  .L5:
13    leaq    (%rdx,%rsi), %rdi
14    salq    $2, %rdi
15    jmp     .L6
16  .L2:
17    movq    %rsi, %rdi
18  .L6:
19    movq    %rdi, (%rcx)
20    ret
```

a）代码

```
1   .L4:
2     .quad   .L3
3     .quad   .L2
4     .quad   .L5
5     .quad   .L2
6     .quad   .L6
7     .quad   .L7
8     .quad   .L2
9     .quad   .L5
```

b）跳转表

填写 *C* 代码中缺失的部分。除了情况标号 *C* 和 *D* 的顺序之外，将不同情况填入这个模板的方法是唯一的。