3.58：

```
long decode(long x, long y, long z) {
  long tmp = y - z;
  return (tmp * x) ^ (tmp << 63 >> 63);
}
```

3.60：

A://对应关系如下表

val	reg
x	%rdi
n	%esi
result	%rax
mask	%rdx

```
B: result = 0 mask = 1
C: mask != 0
D: mask = mask << n
```

E: //通过不同位的掩码，将 x 的对应位传给 result 对应位

```
for (mask = 1; mask != 0; mask <<= n) {
   result |= (x & mask);
}
```

F:

```
long loop(long x, int n) {
  long result = 0;
  long mask;
  for (mask = 1; mask != 0; mask <<= n) {
    result |= (x & mask);
  }
  return result;
}
```

3.63：

```
long switch_prob(long x, long n) {
  long result = x;
  switch(n) {
    /* Fill in code here */
    case 60:
    case 62:
      result = x * 8;
      break;
    case 63:
      result = x >> 3;
      break;
```

```
    case 64:
      x = x << 4 - x;
    case 65:
      x = x * x;
    default:
      result = x + 0x4B;
  }
  return result;
}
```

3.69：

A:

因为 7*40 + 8 = 288 = 0x120

所以 CNT = 7

B:

```
typedef struct {
  long idx,
  long x[4]
} a_struct
```

3.70：

A:

Val	offset
e1.p	0
e1.y	8
e2.x	0
e2.next	8

B:

16

C:

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

分析如下

```
# void proc(union ele *up)
# up 存在 %rdi 中
proc:
  # %rax = *(up+8), 可能是 next or y
  movq 8(%rdi), %rax

  # %rdx = *( *(up+8) ), %rax 表示一个指针
  # 因为 *( *(up+8) ) 所以意思是 *(up->e2.next)
  movq (%rax), %rdx
```

```
# %rdx = *( *(up->e2.next) )
# %rdx 表示一个指针
# 因此 %rdx 存储了 *( *(up->e2.next).e1.p )
movq (%rdx), %rdx

# %rax 存储了 *(up+8)
# %rax 同样是一个指针
# 因此 %rax = *( up->e2.next ), 表示另一个联合 union ele's 的地址

# 由于 subq, %rdx 是长整型
# *( *(up->e2.next)+8 ) 也一定是长整型
# 因此 8(%rax) 表示 *(up->e2.next).e1.y
subq 8(%rax), %rdx

# %rdi 放的东西 up 在之前的指令中从未改变过
# 在下面的指令中再次出现
# 所以 (%rdi) 表示 up->e2.x
movq %rdx, (%rdi)
ret
```