# 3.60

## Problem

We are given the following assembly code:

```
loop:
  movl %esi, %ecx
  movl $1, %edx
  movl $0, %eax
  jmp .L2

.L3:
  movq %rdi, %r8
  andq %rdx, %r8
  orq  %r8, %rax
  salq %cl, %rdx

.L2:
  testq %rdx, %rdx
  jne .L3
  rep; ret
```

and the partially filled corresponding C code:

```
long loop(long x, long n) {
  long result = _____;

  long mask;
  for (mask = ____; mask _____; mask = _____) {
    result |= _____;
  }

  return result;
}
```

## Solutions

Commented assembly code

```
loop:
  movl %esi, %ecx     # move n into rcx
  movl $1, %edx       # move 1 into rdx
  movl $0, %eax       # move 0 into rax ==> (2): rax must be result since rdx is mask
  jmp .L2

.L3:
  movq %rdi, %r8      # move x into r8
  andq %rdx, %r8      # rdx (mask) & r8 (x) -> r8
  orq  %r8, %rax      # r8 | rax (result) -> rax (result) ==> (3): re-assign result
  salq %cl, %rdx      # rdx (mask) << cl (lsB of n) -> rdx (mask) ==> (4): update mask

.L2:
  testq %rdx, %rdx    # rdx & rdx
  jne .L3             # jump back (loop) if rdx not 0 ==> (1): rdx must be mask
  rep; ret
```

A. Which registers hold program values `x`, `n`, `result`, and `mask`?

- `x` -> `%rdi`
- `n` -> `%rcx`
- `result` -> `%rax`
- `mask` -> `%rdx`

B. What are the initial values of `result` and `mask`?

- `result = 0`
- `mask = 1`

C. What is the test condition for `mask`?

- `mask != 0`

D. How does `mask` get updated?

- `mask = mask << (n & 0xFF)`

E. How does `result` get updated?

- `result |= (mask & x);`

F. Fill in all the missing parts of the C code.

```
long loop(long x, long n) {
  long result = 0;

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

  return result;
}
```

# 3.63

## Problem

We are given the following unfilled C code:

```
long switch_prob(long x, long n) {
    long result = x;
    switch(n) {
        /* Fill in code here */
    }
    return result;
}
```

and the corresponding disassembled machine code:

```
0000000000400590 <switch_prob>:

400590: 48 83 ee 3c              sub $0x3c,%rsi

400594: 48 83 fe 05              cmp $0x5,%rsi

400598: 77 29 ja                 4005c3 <switch_prob+0x33>

40059a: ff 24 f5 f8 06 40 00     jmpq *0x4006f8(,%rsi,8)

4005a1: 48 8d 04 fd 00 00 00     lea 0x0(,%rdi,8),%rax

4005a8: 00

4005a9: c3                       retq

4005aa: 48 89 f8                 mov %rdi,%rax

4005ad: 48 c1 f8 03              sar $0x3,%rax

4005b1: c3                       retq

4005b2: 4889f8                   mov %rdi,%rax

4005b5: 48 c1 e0 04              shl $0x4,%rax

4005b9: 48 29 f8                 sub %rdi,%rax

4005bc: 48 89 c7                 mov %rax,%rdi

4005bf: 48 0f af ff              imul %rdi,%rdi

4005c3: 48 8d 47 4b              lea 0x4b(%rdi),%rax

4005c7: c3                       retq
```

as well as the jump table consisting of six 8-byte words:

```
(gdb) x/6gx 0x4006f8

0x4006f8: 0x00000000004005a1  0x00000000004005c3

0x400708: 0x00000000004005a1  0x00000000004005aa

0x400718: 0x00000000004005b2  0x00000000004005bf
```

## Solutions

```
0000000000400590 <switch_prob>:

400590: 48 83 ee 3c              sub $0x3c,%rsi              # %rsi (n) - 60 -> %rsi (n)

400594: 48 83 fe 05              cmp $0x5,%rsi               # %rsi (n) - 5  -> %rsi (n)

400598: 77 29                    ja 4005c3 <switch_prob+0x33> # jump if %rsi (n) is not between 0-5

40059a: ff 24 f5 f8 06 40 00     jmpq *0x4006f8(,%rsi,8)     # ==> jump based on %rsi, switch arg must be n

4005a1: 48 8d 04 fd 00 00 00     lea 0x0(,%rdi,8),%rax       # %rdi (x) * 8 -> %rax (result)  ==> `case 60 & case 62`

4005a8: 00

4005a9: c3                       retq

4005aa: 48 89 f8                 mov %rdi,%rax               # %rdi (x) -> %rax (result) ==> `case 63`

4005ad: 48 c1 f8 03              sar $0x3,%rax               # %rax (result) >> 3 -> %rax (result)

4005b1: c3                       retq

4005b2: 4889f8                   mov %rdi,%rax               # %rdi (x) -> %rax (result) ==> `case 64` (falls through)

4005b5: 48 c1 e0 04              shl $0x4,%rax               # %rax (result) << 4 -> %rax (result)

4005b9: 48 29 f8                 sub %rdi,%rax               # %rax (result) - %rdi (x) -> %rax (result)

4005bc: 48 89 c7                 mov %rax,%rdi               # %rax (result) -> %rdi

4005bf: 48 0f af ff              imul %rdi,%rdi              # %rdi * %rdi -> %rdi ==> `case 65` (falls through)

4005c3: 48 8d 47 4b              lea 0x4b(%rdi),%rax         # 75 + %rdi -> %rax (result) ==> `case 61`

4005c7: c3                       retq
```

We can tell which lines of code correspond to the cases based on the jump table.

```
long switch_prob(long x, unsigned long n) {
    long result = x;
    switch(n) {
        case 60:
        case 62:
            result *= 8;
            break;
        case 63:
            result /= 8;
            break;
        case 64:
            result *= 16;
            x = result - x;
        case 65:
            x *= x;
        default:
            result = 75 + x;
    }
    return result;
}
```