产生的实现各个动作的汇编代码部分如图 3-52 所示。注释指明了参数位置，寄存器值，以及各个跳转目的的情况标号。

```
     p1 in %rdi, p2 in %rsi, action in %edx
1    .L8:                            MODE_E
2      movl    $27, %eax
3      ret
4    .L3:                            MODE_A
5      movq    (%rsi), %rax
6      movq    (%rdi), %rdx
7      movq    %rdx, (%rsi)
8      ret
9    .L5:                            MODE_B
10     movq    (%rdi), %rax
11     addq    (%rsi), %rax
12     movq    %rax, (%rdi)
13     ret
14   .L6:                            MODE_C
15     movq    $59, (%rdi)
16     movq    (%rsi), %rax
17     ret
18   .L7:                            MODE_D
19     movq    (%rsi), %rax
20     movq    %rax, (%rdi)
21     movl    $27, %eax
22     ret
23   .L9:                            default
24     movl    $12, %eax
25     ret
```

图 3-52　家庭作业 3.62 的汇编代码。这段代码实现了 switch 语句的各个分支

填写 C 代码中缺失的部分。代码包括落入其他情况的情况，试着重建这个情况。

••3.63 这个程序给你一个机会，从反汇编机器代码逆向工程一个 switch 语句。在下面这个过程中，去掉了 switch 语句的主体：

```
1  long switch_prob(long x, long n) {
2      long result = x;
3      switch(n) {
4          /* Fill in code here */
5
6      }
7      return result;
8  }
```

图 3-53 给出了这个过程的反汇编机器代码。

跳转表驻留在内存的不同区域中。可以从第 5 行的间接跳转看出来，跳转表的起始地址为 0x4006f8。用调试器 GDB，我们可以用命令 x/6gx 0x4006f8 来检查组成跳转表的 6 个 8 字节字的内存。GDB 打印出下面的内容：

```
(gdb) x/6gx 0x4006f8
0x4006f8:    0x00000000004005a1    0x00000000004005c3
0x400708:    0x00000000004005a1    0x00000000004005aa
0x400718:    0x00000000004005b2    0x00000000004005bf
```

用 C 代码填写开关语句的主体，使它的行为与机器代码一致。