


```
void remdiv(long x, long y, long *qp, long *rp)
```

```
x in %rdi, y in %rsi, qp in %rdx, rp in %rcx
```

```
1 remdiv:
2     movq    %rdx, %r8           Copy qp
3     movq    %rdi, %rax          Move x to lower 8 bytes of dividend
4     cqto    %rax                Sign-extend to upper 8 bytes of dividend
5     idivq   %rsi                Divide by y
6     movq    %rax, (%r8)         Store quotient at qp
7     movq    %rdx, (%rcx)        Store remainder at rp
8     ret
```

在上述代码中，必须首先把参数 `qp` 保存到另一个寄存器中（第 2 行），因为除法操作要使用参数寄存器 `%rdx`。接下来，第 3~4 行准备被除数，复制并符号扩展 `x`。除法之后，寄存器 `%rax` 中的商被保存在 `qp`（第 6 行），而寄存器 `%rdx` 中的余数被保存在 `rp`（第 7 行）。

无符号除法使用 `divq` 指令。通常，寄存器 `%rdx` 会事先设置为 0。

 **练习题 3.12** 考虑如下函数，它计算两个无符号 64 位数的商和余数：

```
void uremdiv(unsigned long x, unsigned long y,
             unsigned long *qp, unsigned long *rp) {
    unsigned long q = x/y;
    unsigned long r = x%y;
    *qp = q;
    *rp = r;
}
```

修改有符号除法的汇编代码来实现这个函数。

3.6 控制

到目前为止，我们只考虑了直线代码的行为，也就是指令一条接一条顺序地执行。C 语言中的某些结构，比如条件语句、循环语句和分支语句，要求有条件的执行，根据数据测试的结果来决定操作执行的顺序。机器代码提供两种基本的低级机制来实现有条件的行为：测试数据值，然后根据测试的结果来改变控制流或者数据流。

与数据相关的控制流是实现有条件行为的更一般和更常见的方法，所以我们先来介绍它。通常，C 语言中的语句和机器代码中的指令都是按照它们在程序中出现的次序，顺序执行的。用 `jump` 指令可以改变一组机器代码指令的执行顺序，`jump` 指令指定控制应该被传递到程序的某个其他部分，可能是依赖于某个测试的结果。编译器必须产生构建在这种低级机制基础之上的指令序列，来实现 C 语言的控制结构。

本文会先涉及实现条件操作的两种方式，然后描述表达循环和 `switch` 语句的方法。

3.6.1 条件码

除了整数寄存器，CPU 还维护着一组单个位的条件码（condition code）寄存器，它们描述了最近的算术或逻辑操作的属性。可以检测这些寄存器来执行条件分支指令。最常用的条件码有：

CF：进位标志。最近的操作使最高位产生了进位。可用来检查无符号操作的溢出。

ZF：零标志。最近的操作得出的结果为 0。

SF：符号标志。最近的操作得到的结果为负数。

OF：溢出标志。最近的操作导致一个补码溢出——正溢出或负溢出。