


旁注 指令 rep 和 repz 有什么用

本节开始的汇编代码的第 8 行包含指令组合 `rep; ret`。它们在反汇编代码中(第 6 行)对应于 `repz retq`。可以推测出 `repz` 是 `rep` 的同义名,而 `retq` 是 `ret` 的同义名。查阅 Intel 和 AMD 有关 `rep` 的文档,我们发现它通常用来实现重复的字符串操作[3, 51]。在这里用它似乎很不合适。这个问题的答案可以在 AMD 给编译器编写者的指导意见书[1]中找到。他们建议用 `rep` 后面跟 `ret` 的组合来避免使 `ret` 指令成为条件跳转指令的目标。如果没有 `rep` 指令,当分支不跳转时, `jg` 指令(汇编代码的第 7 行)会继续到 `ret` 指令。根据 AMD 的说法,当 `ret` 指令通过跳转指令到达时,处理器不能正确预测 `ret` 指令的目的。这里的 `rep` 指令就是作为一种空操作,因此作为跳转目的插入它,除了能使代码在 AMD 上运行得更快之外,不会改变代码的其他行为。在本书后面其他代码中再遇到 `rep` 或 `repz` 时,我们可以很放心地无视它们。

 **练习题 3.15** 在下面这些反汇编二进制代码节选中,有些信息被 X 代替了。回答下列关于这些指令的问题。

A. 下面 `je` 指令的目标是什么?(在此,你不需要知道任何有关 `callq` 指令的信息。)

```
4003fa: 74 02          je      XXXXXX
4003fc: ff d0         callq   %rax
```

B. 下面 `je` 指令的目标是什么?

```
40042f: 74 f4          je      XXXXXX
400431: 5d            pop     %rbp
```

C. `ja` 和 `pop` 指令的地址是多少?

```
XXXXXX: 77 02          ja      400547
XXXXXX: 5d            pop     %rbp
```

D. 在下面的代码中,跳转目标的编码是 PC 相对的,且是一个 4 字节补码数。字节按照从最低位到最高位的顺序列出,反映出 x86-64 的小端法字节顺序。跳转目标的地址是什么?

```
4005e8: e9 73 ff ff    jmpq   XXXXXXX
4005ed: 90            nop
```

跳转指令提供了一种实现条件执行(`if`)和几种不同循环结构的方式。

3.6.5 用条件控制来实现条件分支

将条件表达式和语句从 C 语言翻译成机器代码,最常用的方式是结合有条件和无条件跳转。(另一种方式在 3.6.6 节中会看到,有些条件可以用数据的条件转移实现,而不是用控制的条件转移来实现。)例如,图 3-16a 给出了一个计算两数之差绝对值的函数的 C 代码^①。这个函数有一个副作用,会增加两个计数器,编码为全局变量 `lt_cnt` 和 `ge_cnt` 之一。GCC 产生的汇编代码如图 3-16c 所示。把这个机器代码再转换成 C 语言,我们称之为函数 `gotodiff_se`(图 3-16b)。它使用了 C 语言中的 `goto` 语句,这个语句类似于汇编代码中的无条件跳转。使用 `goto` 语句通常认为是一种不好的编程风格,因为它会使代码非

① 实际上,如果一个减法溢出,这个函数就会返回一个负数值。这里我们主要是为了展示机器代码,而不是实现代码的健壮性。