

3.11.4 定义和使用浮点常数


和整数运算操作不同, AVX 浮点操作不能以立即数值作为操作数。相反, 编译器必须为所有的常量值分配和初始化存储空间。然后代码在把这些值从内存读入。下面从摄氏度到华氏度转换的函数就说明了这个问题:

```
double cel2fahr(double temp)
{
    return 1.8 * temp + 32.0;
}
```

相应的 x86-64 汇编代码部分如下:

```
double cel2fahr(double temp)
temp in %xmm0
1  cel2fahr:
2  vmulsd .LC2(%rip), %xmm0, %xmm0    Multiply by 1.8
3  vaddsd .LC3(%rip), %xmm0, %xmm0    Add 32.0
4  ret
5  .LC2:
6  .long  3435973837                  Low-order 4 bytes of 1.8
7  .long  1073532108                  High-order 4 bytes of 1.8
8  .LC3:
9  .long  0                          Low-order 4 bytes of 32.0
10 .long  1077936128                  High-order 4 bytes of 32.0
```

可以看到函数从标号为 .LC2 的内存位置读出值 1.8, 从标号为 .LC3 的位置读入值 32.0。观察这些标号对应的值, 可以看出每一个都是通过一对 .long 声明和十进制表示的值指定的。该怎样把这些数解释为浮点值呢? 看看标号为 .LC2 的声明, 有两个值: 3435973837 (0xcccccccd) 和 1073532108 (0x3ffcccccc)。因为机器采用的是小端法字节顺序, 第一个值给出的是低位 4 字节, 第二个给出的是高位 4 字节。从高位字节, 可以抽取指数字段为 0x3ff (1023), 减去偏移 1023 得到指数 0。将两个值的小数位连接起来, 得到小数字段 0xcccccccccccd, 二进制小数表示为 0.8, 加上隐含的 1 得到 1.8。

 **练习题 3.55** 解释标号为 .LC3 处声明的数字是如何对数字 32.0 编码的。

3.11.5 在浮点代码中使用位级操作

有时, 我们会发现 GCC 生成的代码会在 XMM 寄存器上执行位级操作, 得到有用的浮点结果。图 3-50 展示了一些相关的指令, 类似于它们在通用寄存器上对应的操作。这些操作都作用于封装好的数据, 即它们更新整个目的 XMM 寄存器, 对两个源寄存器的所有位都实施指定的位级操作。和前面一样, 我们只对标量数据感兴趣, 只想了解这些指令对目的寄存器的低 4 或 8 字节的影响。从下面的例子中可以看出, 运用这些操作通常可以简单方便地操作浮点数。

单精度	双精度	效果	描述
vxorps	vorpd	$D \leftarrow S_2 \sim S_1$	位级异或 (EXCLUSIVE-OR)
vandps	andpd	$D \leftarrow S_2 \& S_1$	位级与 (AND)

图 3-50 对封装数据的位级操作 (这些指令对一个 XMM 寄存器中的所有 128 位进行布尔操作)