入，这是 C 和大多数其他编程语言的要求。

指令	源	目的	描述
vcvttss2si	X/M_{32}	R_{32}	用截断的方法把单精度数转换成整数
vcvttsd2si	X/M_{64}	R_{32}	用截断的方法把双精度数转换成整数
vcvttss2siq	X/M_{32}	R_{64}	用截断的方法把单精度数转换成四字整数
vcvttsd2siq	X/M_{64}	R_{64}	用截断的方法把双精度数转换成四字整数

图 3-47　双操作数浮点转换指令。这些操作将浮点数转换成整数（X：XMM 寄存器（例如%xmm3）；R_{32}：32 位通用寄存器（例如%eax）；R_{64}：64 位通用寄存器（例如%rax）；M_{32}：32 位内存范围；M_{64}：64 位内存范围）

指令	源 1	源 2	目的	描述
vcvtsi2ss	M_{32}/R_{32}	X	X	把整数转换成单精度数
vcvtsi2sd	M_{32}/R_{32}	X	X	把整数转换成双精度数
vcvtsi2ssq	M_{64}/R_{64}	X	X	把四字整数转换成单精度数
vcvtsi2sdq	M_{64}/R_{64}	X	X	把四字整数转换成双精度数

图 3-48　三操作数浮点转换指令。这些操作将第一个源的数据类型转换成目的的数据类型。第二个源值对结果的低位字节没有影响（X：XMM 寄存器（例如%xmm3）；M_{32}：32 位内存范围；M_{64}：64 位内存范围）

图 3-48 中的指令把整数转换成浮点数。它们使用的是不太常见的三操作数格式，有两个源和一个目的。第一个操作数读自于内存或一个通用目的寄存器。这里可以忽略第二个操作数，因为它的值只会影响结果的高位字节。而我们的目标必须是 XMM 寄存器。在最常见的使用场景中，第二个源和目的操作数都是一样的，就像下面这条指令：

```
vcvtsi2sdq   %rax, %xmm1, %xmm1
```

这条指令从寄存器%rax 读出一个长整数，把它转换成数据类型 double，并把结果存放进 XMM 寄存器%xmm1 的低字节中。

最后，要在两种不同的浮点格式之间转换，GCC 的当前版本生成的代码需要单独说明。假设%xmm0 的低位 4 字节保存着一个单精度值，很容易就想到用下面这条指令

```
vcvtss2sd   %xmm0, %xmm0, %xmm0
```

把它转换成一个双精度值，并将结果存储在寄存器%xmm0 的低 8 字节。不过我们发现 GCC 生成的代码如下

```
      Conversion from single to double precision
1     vunpcklps  %xmm0, %xmm0, %xmm0   Replicate first vector element
2     vcvtps2pd  %xmm0, %xmm0          Convert two vector elements to double
```

vunpcklps 指令通常用来交叉放置来自两个 XMM 寄存器的值，把它们存储到第三个寄存器中。也就是说，如果一个源寄存器的内容为字$[s_3, s_2, s_1, s_0]$，另一个源寄存器为字$[d_3, d_2, d_1, d_0]$，那么目的寄存器的值会是$[s_1, d_1, s_0, d_0]$。在上面的代码中，我们看到三个操作数使用同一个寄存器，所以如果原始寄存器的值为$[x_3, x_2, x_1, x_0]$，那么该指令会将寄存器的值更新为值$[x_1, x_1, x_0, x_0]$。vcvtps2pd 指令把源 XMM 寄存器中的两个低位单精度值扩展成目的 XMM 寄存器中的两个双精度值。对前面 vunpcklps 指令的结果应用这条指令会得到值$[dx_0, dx_0]$，这里 dx_0 是将 x 转换成双精度后的结果。