北京大学 PEKING UNIVERSITY | 物理学院

程序的机器级表示（浮点代码部分）

计算机系统导论　课程回课

包涛尼

北京大学物理学院

2022 年 10 月 5 日

Contents

浮点体系结构的历史和基础知识

一点历史

Single Instruction Multiple Data，简称 SIMD．SIMD 描述的是微处理器中单条指令能完成对数据的并行处理．SIMD 所使用的是特殊的寄存器，一个寄存器上存储有多个数据，在进行 SIMD 运算时，这些数据会被分别进行处理，以此实现了数据的并行处理．

1. MultiMedia eXtensions（MMX）,1997.
2. Streaming SIMD eXtensions（SSE），1999，在 Pentium III 时对 SIMD 做了扩展.
3. SSE2，2000，在 Pentium IV 中引入，包括对**标量浮点**数据进行操作的指令，使用 XMM/YMM 寄存器的低 32 位/64 位中的单个值.
4. SSE3（2004），SSSE3，SSE4（2006）.
5. Advanced Vector eXtentions（AVX），2008，对 XMM 寄存器做了扩展，从原来的 128–bit 扩展到了 256–bit，256–bit 的寄存器命名为 YMM．YMM 的低 128–bit 是与 XMM 混用的．AVX2，2013.

媒体寄存器

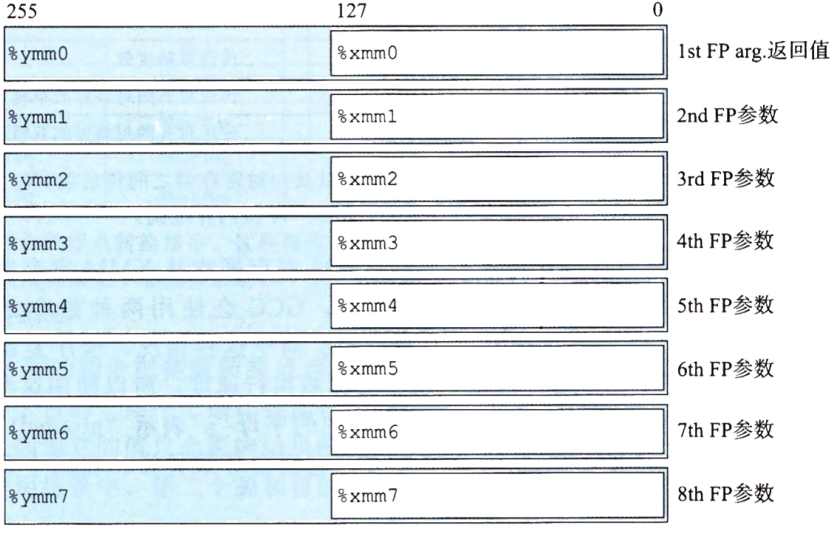

图 1 媒体寄存器（%ymm0～%ymm7）

媒体寄存器

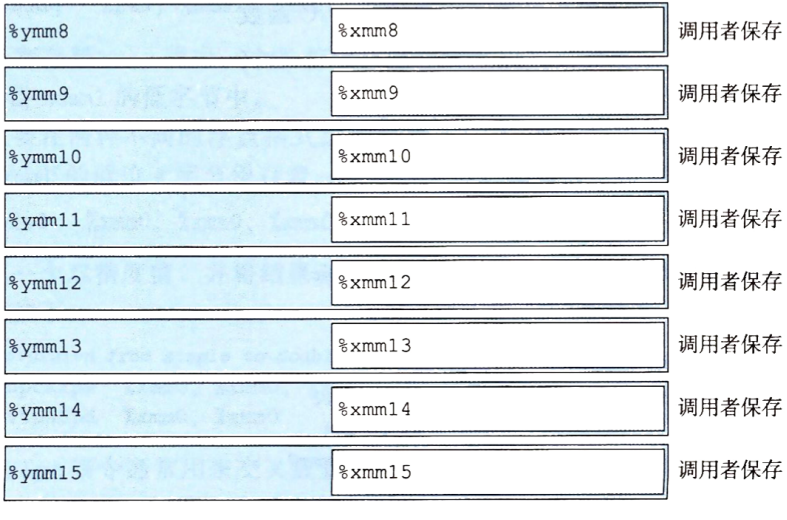

图 2 媒体寄存器（%ymm7～%ymm15）

媒体寄存器

- 如图 1 和图 2 所示，AVX 浮点体系结构允许数据存储在 16 个 YMM 寄存器中，它们的名字为 `%ymm0`～`%ymm15`.
- 只保存浮点数，而且只使用低 32 位（对于 `float`）或 64 位（对于 `double`）.
- 汇编代码用寄存器的 SSE XMM 寄存器名字 `%xmm0`～`%xmm15` 来引用它们，每个 XMM 寄存器都是对应的 YMM 寄存器的低 128 位（16 字节）.

浮点传送与转换

浮点传送操作

表 1 浮点传送指令，在内存和寄存器之间/一对寄存器之间传送值

指令	源	目的	描述
vmovss	M_{32}	X	传送单精度数
vmovss	X	M_{32}	传送单精度数
vmovsd	M_{64}	X	传送双精度数
vmovsd	X	M_{64}	传送双精度数
vmovaps	X	X	传送对齐的封装好的单精度数
vmovapd	X	X	传送对齐的封装好的双精度数

浮点传送操作

```
1  float float_mov(float v1, float *src, float *dst)
2  {
3      float v2 = *src;
4      *dst = v1;
5      return v2;
6  }
```

在 x86–64 Linux 机器上使用 gcc -Og -S 命令，得到如下汇编代码：

```
1  /* v1 in %xmm0, src in %rdi, dst in %rsi */
2  float_mov:
3      movaps  %xmm0,  %xmm1    /* Copy v1 */
4      movss   (%rdi), %xmm0    /* Read v2 from src */
5      movss   %xmm1,  (%rsi)   /* Write v1 to dst */
6      ret                      /* Return v2 to %xmm0 */
```

浮点转换操作

表 2 双操作数浮点转换指令（浮点 $\rightarrow$ 整数）

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

注意：目的只能是**通用寄存器**！

浮点转换操作

表 3 三操作数浮点转换指令（整数 → 浮点）

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

注意：目的只能是 **XMM 寄存器**！

在最常见的使用场景中，源 2 和目的操作数通常是一样的（因为源 2 的值只影响结果的高位字节，我们可以忽略）.

浮点转换操作

- 那么怎么进行 `float` 和 `double` 类型之间的转换?
- `vcvtss2sd %xmm0, %xmm0, %xmm0`? `vcvtsd2ss %xmm0, %xmm0, %xmm0`?
- GCC 并不是这么实现的.
- 对于单精度到双精度的转换:

```
1  vunpcklps   %xmm0, %xmm0, %xmm0     /* Replicate first vector element */
2  vcvtps2pd   %xmm0, %xmm0            /* Convert two vector elements to double */
```

- `vunpcklps` 用于交叉放置来自两个 XMM 寄存器的值，而 `vcvtps2pd` 用于把源 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.
- 过程: $[x_3, x_2, x_1, x_0] \rightarrow [x_1, x_1, x_0, x_0] \rightarrow [dx_0, dx_0]$.

浮点转换操作

- 那么怎么进行 `float` 和 `double` 类型之间的转换?
- `vcvtss2sd %xmm0, %xmm0, %xmm0`? `vcvtsd2ss %xmm0, %xmm0, %xmm0`?
- GCC 并不是这么实现的.
- 类似地，对于双精度到单精度的转换：

```
1  vmovddup    %xmm0, %xmm0     /* Replicate first vector element */
2  vcvtpd2psx  %xmm0, %xmm0     /* Convert two vector elements to single */
```

- `vmovddup` 将 $[dx_1, dx_0]$ 变成 $[dx_0, dx_0]$；而 `vcvtpd2psx` 将这两个值转换成单精度，再存放到该寄存器低位一半中，并将高位置 0.
- 过程：$[dx_1, dx_0] \to [dx_0, dx_0] \to [0.0, 0.0, x_0, x_0]$.

过程中的浮点代码

过程中的浮点代码

- 程序使用寄存器 %xmm0 来返回浮点值.
- 当函数包含指针、整数和浮点数混合的参数时，指针和整数通过通用寄存器传递，而浮点值通过 XMM 寄存器传递.
- 参数到寄存器的映射取决于它们的类型和排列顺序.

```
1  double f1(int x, double y, long z);
2      /* x in %edi, y in %xmm0, z in %rsi */
3
4  double f2(double y, int x, long z);
5      /* x in %edi, y in %xmm0, z in %rsi */
6
7  double f3(float x, double *y, long *z);
8      /* x in %xmm0, y in %rdi, z in %rsi */
```

浮点运算、位级和比较操作

浮点运算操作

表 4 标量浮点算术运算

单精度	双精度	效果	描述
vaddss	vaddsd	$D \leftarrow S_2 + S_1$	浮点数加
vsubss	vsubsd	$D \leftarrow S_2 - S_1$	浮点数减
vmulss	vmulsd	$D \leftarrow S_2 \times S_1$	浮点数乘
vdivss	vdivsd	$D \leftarrow S_2 / S_1$	浮点数除
vmaxss	vmaxsd	$D \leftarrow \max(S_2, S_1)$	浮点数最大值
vminss	vminsd	$D \leftarrow \min(S_2, S_1)$	浮点数最小值
sqrtss	sqrtsd	$D \leftarrow \sqrt{S_1}$	浮点数平方根

浮点位级操作

表 5 对封装数据的位级操作（对 XMM 寄存器全部 128 位进行 Bool 操作）

单精度	双精度	效果	描述
vxorps	xorpd	$D \leftarrow S_2 \hat{} S_1$	位级异或（EXCLUSIVE–OR）
vandps	andpd	$D \leftarrow S_2 \,\&\, S_1$	位级与（AND）

浮点比较操作

表 6 浮点比较操作

指令	基于	描述
`ucomiss` S_1, S_2	$S_2 - S_1$	比较单精度值
`ucomisd` S_1, S_2	$S_2 - S_1$	比较双精度值

- 可能会设置三个条件码：
  1. 零标志位 `ZF`
  2. 进位标志位 `CF`
  3. **奇偶标志位** `PF`（运算产生的值最低位字节是偶校验时/其中一个操作数是 `NaN` 时设置）

浮点比较操作

表 7 浮点比较操作对条件码的设置

顺序	CF	ZF	PF
无序的	1	1	1
$S_2 < S_1$	1	0	0
$S_2 = S_1$	0	1	0
$S_2 > S_1$	0	0	1

- 从而可以利用诸如 `ja`、`jb`、`jp`（jump on parity）等实现条件跳转.

参考文献

[1] Bryant R E, O'Hallaron D R. 深入理解计算机系统 [M]. 北京: 机械工业出版社, 2016.

谢 谢！