附录 C SSE 指令集相关

C.1 整型算术指令

- _mm_add_epi16 (__m128i a, __m128i b)

  返回一个__m128i 的寄存器，将 a 和 b 中对应位置的 16bit 有符号整数分别相加，即 $r_i = a_i + b_i$

- _mm_adds_epi16 (__m128i a, __m128i b)

  返回一个__m128i 的寄存器，将 a 和 b 中对应位置的 16bit 有符号整数分别相加，当计算结果溢出时将其置为边界值。

- _mm_sub_epi16 (__m128i a, __m128i b)

  返回一个__m128i 的寄存器，将 a 和 b 中对应位置的 16bit 有符号整数分别相减，即 $r_i = a_i - b_i$

- _mm_subs_epi16 (__m128i a, __m128i b)

  返回一个__m128i 的寄存器，将 a 和 b 中对应位置的 16bit 有符号整数分别相减，当计算结果溢出时将其置为边界值。

- _mm_avg_epu16 (__m128i a, __m128i b)

  返回一个__m128i 的寄存器，将 a 和 b 中对应位置的 16bit 无符号整数取平均，即 $r_i = (a_i + b_i)/2$

- _mm_madd_epi16 (__m128i a, __m128i b)

  返回一个__m128i 的寄存器，它含 4 个有符号 32bit 的整数，分别满足：

$$r_0 = a_0 \times b_0 + a_1 \times b_1$$

$$r_1 = a_2 \times b_2 + a_3 \times b_3$$

$$r_2 = a_4 \times b_4 + a_5 \times b_5$$

$$r_3 = a_6 \times b_6 + a_7 \times b_7$$

- _mm_max_epi16 (__m128i a, __m128i b)

  返回一个__m128i 的寄存器，取 a 和 b 中对应位置的 16bit 有符号整数的最大值，即 $r_i = \max(a_i, b_i)$

- _mm_min_epi16 (__m128i a, __m128i b)

  返回一个__m128i 的寄存器，取 a 和 b 中对应位置的 16bit 有符号整数的最小值，即 $r_i = \min(a_i, b_i)$

- _mm_mulhi_epi16 (__m128i a, __m128i b)

  返回一个__m128i 的寄存器，它含 8 个有符号 16bit 的整数，分别为 a 和 b 对应位置的 16bit 有符号整数相乘结果的高 16bit 数据，即 $r_i = (a_i \times b_i)[31:16]$

- _mm_mullo_epi16 (__m128i a, __m128i b)

  返回一个__m128i 的寄存器，它含 8 个有符号 16bit 的整数，分别为 a 和 b 对应位置的 16bit 有符号整数相乘结果的低 16bit 数据，即 $r_i = (a_i \times b_i)[15:0]$

C.2 整型移位指令

- _mm_slli_epi16 (__m128i a, int count)

  返回一个__m128i 的寄存器，将寄存器 a 中的 8 个 16bit 整数按照 count 进行相同的逻辑左移。

- _mm_sll_epi16 (__m128i a, __m128i count)

  返回一个__m128i 的寄存器，将寄存器 a 中的 8 个 16bit 整数按照 count 寄存器中对应位置的整数进行逻辑左移。

- _mm_srai_epi16 (__m128i a, int count)

返回一个__m128i 的寄存器，将寄存器 a 中的 8 个 16bit 整数按照 count 进行相同的算术右移。

- _mm_sra_epi16 (__m128i a, __m128i count)

  返回一个__m128i 的寄存器，将寄存器 a 中的 8 个 16bit 整数按照 count 寄存器中对应位置的整数进行算术右移。

- _mm_srli_epi16 (__m128i a, int count)

  返回一个__m128i 的寄存器，将寄存器 a 中的 8 个 16bit 整数按照 count 进行相同的逻辑右移，移位填充值为 0。

- _mm_srl_epi16 (__m128i a, __m128i count)

  返回一个__m128i 的寄存器，将寄存器 a 中的 8 个 16bit 整数按照 count 寄存器中对应位置的整数进行逻辑右移，移位填充值为 0。

C.3 整型逻辑指令

- _mm_and_si128(__m128i a , __m128i b)

  返回为一个__m128i 的寄存器，将寄存器 a 和寄存器 b 的对应位进行按位与运算。

- mm_andnot_si128(__m128i a , __m128i b)

  返回为一个__m128i 的寄存器，将寄存器 a 每一位取非，然后和寄存器 b 的每一位进行按位与运算。

- mm_or_si128(__m128i a , __m128i b)

  返回为一个__m128i 的寄存器，将将寄存器 a 和寄存器 b 的对应位进行按位或运算。

- mm_xor_si128(__m128i a , __m128i b)

返回为一个__m128i 的寄存器，将寄存器 a 和寄存器 b 的对应位进行按位异或运算。

C.4 整型比较指令

- _mm_cmpeq_epi16 (__m128i a, __m128i b)

  返回一个__m128i 的寄存器，分别比较寄存器 a 和寄存器 b 对应位置 16bit 整数是否相等，若相等，该位置返回 0xffff，否则返回 0x0。即 $r_i = (a_i == b_i)$? 0xffff: 0x0

- _mm_cmpgt_epi16 (__m128i a, __m128i b)

  返回一个__m128i 的寄存器，分别比较寄存器 a 的每个 16bit 整数是否大于寄存器 b 对应位置 16bit 整数，若大于，该位置返回 0xffff，否则返回 0x0。即$r_i = (a_i > b_i)$? 0xffff: 0x0

- _mm_cmplt_epi16 (__m128i a, __m128i b)

  返回一个__m128i 的寄存器，分别比较寄存器 a 的每个 16bit 整数是否小于寄存器 b 对应位置 16bit 整数，若小于，该位置返回 0xffff，否则返回 0x0。即$r_i = (a_i < b_i)$? 0xffff: 0x0

C.5 整型混杂指令

- _mm_shuffle_epi32 (__m128i a, int imm)

  返回一个__m128i 的寄存器，它是将 a 中 128bit 数据以 32bit 为单位重新排列得到的，imm 为一个四元组，表示重新排列的顺序。当 a 中原本存储的整数为 16bit 时，这条指令将其两两一组进行排列。

  例如，a = $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$，imm = (2,3,0,1)，其中$a_i$为 16bit 整数，$a_0$为低位，返回结果为$(a_2, a_3, a_0, a_1, a_6, a_7, a_4, a_5)$

- _mm_shufflehi_epi16 (__m128i a, int imm)

返回一个__m128i 的寄存器，它是将 a 中高 64bit 数据以 16bit 为单位重新排列得到的，imm 为一个四元组，表示重新排列的顺序。a 中低 64bit 数据顺序不变。

例如，a = $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$，imm = (2,3,0,1)，其中$a_i$为 16bit 整数，$a_0$为低位，返回结果为$(a_0, a_1, a_2, a_3, a_5, a_4, a_7, a_6)$

- _mm_shufflelo_epi16 (__m128i a, int imm)

  返回一个__m128i 的寄存器，它是将 a 中低 64bit 数据以 16bit 为单位重新排列得到的，imm 为一个四元组，表示重新排列的顺序。a 中高 64bit 数据顺序不变。

  例如，a = $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$，imm = (2,3,0,1)，其中$a_i$为 16bit 整数，$a_0$为低位，返回结果为$(a_1, a_0, a_3, a_2, a_4, a_5, a_6, a_7)$

- _mm_unpackhi_epi16 (__m128i a, __m128i b)

  返回一个__m128i 的寄存器，它将寄存器 a 和寄存器 b 的高 64bit 数以 16bit 为单位交织在一块。

  例如，a = $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$，b = $(b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7)$，

  其中a_i、b_i为 16bit 整数，a_0、b_0为低位，

  返回结果为$(a_4, b_4, a_5, b_5, a_6, b_6, a_7, b_7)$

- _mm_unpackhi_epi32 (__m128i a, __m128i b)

  返回一个__m128i 的寄存器，它将寄存器 a 和寄存器 b 的高 64bit 数以 32bit 为单位交织在一块。

  例如，a = $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$，b = $(b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7)$，

  其中a_i、b_i为 16bit 整数，a_0、b_0为低位，

  返回结果为$(a_4, a_5, b_4, b_5, a_6, a_7, b_6, b_7)$

- _mm_unpackhi_epi64 (__m128i a, __m128i b)

  返回一个__m128i 的寄存器，它将寄存器 a 和寄存器 b 的高 64bit 数以 64bit 为整体交织在一块。

例如，a = $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$，b = $(b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7)$，

其中a_i、b_i为 16bit 整数，a_0、b_0为低位，

返回结果为$(a_4, a_5, a_6, a_7, b_4, b_5, b_6, b_7)$

- _mm_unpacklo_epi16 (__m128i a, __m128i b)

  返回一个__m128i 的寄存器，它将寄存器 a 和寄存器 b 的低 64bit 数以 16bit 为单位交织在一块。

  例如，a = $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$，b = $(b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7)$，

  其中a_i、b_i为 16bit 整数，a_0、b_0为低位，

  返回结果为$(a_0, b_0, a_1, b_1, a_2, b_2, a_3, b_3)$

- _mm_unpacklo_epi32 (__m128i a, __m128i b)

  返回一个__m128i 的寄存器，它将寄存器 a 和寄存器 b 的低 64bit 数以 32bit 为单位交织在一块。

  例如，a = $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$，b = $(b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7)$，

  其中a_i、b_i为 16bit 整数，a_0、b_0为低位，

  返回结果为$(a_0, a_1, b_0, b_1, a_2, a_3, b_2, b_3)$

- _mm_unpackhi_epi64 (__m128i a, __m128i b)

  返回一个__m128i 的寄存器，它将寄存器 a 和寄存器 b 的高 64bit 数以 64bit 为整体交织在一块。

  例如，a = $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$，b = $(b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7)$，

  其中a_i、b_i为 16bit 整数，a_0、b_0为低位，

  返回结果为$(a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3)$

- _mm_extract_epi16 (__m128i a, int imm)

  返回一个 16bit 整数，根据 imm 从 a 中 8 个 16bit 数中选取对应编号的数。

- _mm_insert_epi16 (__m128i a, int b, int imm)

返回一个__m128i 的寄存器，根据 imm 将 a 中 8 个 16bit 数中对应编号的数替换为 b。

C.6 整型读取存储指令

- _mm_load_si128 (__m128i *p)

  返回为一个__m128i 的寄存器，它将 p 指向的数据读到指定寄存器中，实际使用时，p 一般是通过类型转换得到的。

- _mm_store_si128 (__m128i *p, __m128i a)

  返回为空，它将寄存器 a 中的数据存储到 p 指向的地址中，实际使用时，p 一般是通过类型转换得到的。

- _mm_set_epi16 (short w7, short w6, short w5, short w4, short w3, short w2, short w1, short w0)

  返回一个__m128i 的寄存器，使用 8 个具体的 short 型数据来设置寄存器存放数据。

C.7 浮点型算术指令

- _mm_add_ps (__m128 a, __m128 b)

  返回为一个__m128 的寄存器，将寄存器 a 和寄存器 b 的对应位置的 32bit 单精度浮点数相加。

- _mm_add_ss (__m128 a, __m128 b)

  返回为一个__m128 的寄存器，仅将寄存器 a 和寄存器 b 最低对应位置的 32bit 单精度浮点数相加，其余位置取寄存器 a 中的数据。

  例如 $\mathrm{a} = (a_0, a_1, a_2, a_3)$，$\mathrm{b} = (b_0, b_1, b_2, b_3)$，则返回寄存器为 $(a_0 + \mathrm{b}_0, a_1, a_2, a_3)$

对于以下的指令均有类似的对应关系，_ps 结尾的指令表示对 4 个单精度浮点数同时进行运算，_ss 结尾的指令表示仅对 4 个单精度浮点数最低位的浮点数进行运算。

- _mm_sub_ps (__m128 a, __m128 b)

  返回为一个__m128 的寄存器，将寄存器 a 和寄存器 b 的对应位置的 32bit 单精度浮点数相减。

- _mm_mul_ps (__m128 a, __m128 b)

  返回为一个__m128 的寄存器，将寄存器 a 和寄存器 b 的对应位置的 32bit 单精度浮点数相乘。

- _mm_div_ps (__m128 a, __m128 b)

  返回为一个__m128 的寄存器，将寄存器 a 和寄存器 b 的对应位置的 32bit 单精度浮点数相除。

- _mm_sqrt_ps (__m128 a)

  返回为一个__m128 的寄存器，将寄存器 a 的 4 个 32bit 单精度浮点数分别开平方。

- _mm_rcp_ps (__m128 a)

  返回为一个__m128 的寄存器，将寄存器 a 的 4 个 32bit 单精度浮点数分别取倒数。

- _mm_rsqrt_ps (__m128 a)

  返回为一个__m128 的寄存器，将寄存器 a 的 4 个 32bit 单精度浮点数分别取平方根的倒数。

- _mm_min_ps (__m128 a, __m128 b)

  返回为一个__m128 的寄存器，对寄存器 a 和寄存器 b 的对应位置的 32bit 单精度浮点数分别取最小值。

- _mm_max_ps (__m128 a, __m128 b)

返回为一个__m128 的寄存器，对寄存器 a 和寄存器 b 的对应位置的 32bit 单精度浮点数分别取最大值。

- _mm_addsub_ps(__m128 a, __m128 b)

  返回为一个__m128 的寄存器，对寄存器 a 和寄存器 b 的对应位置的 32bit 单精度浮点数分别进行减加运算。如$A = (a_0, a_1, a_2, a_3)$，$B = (b_0, b_1, b_2, b_3)$，返回值为$(a_0 - b_0, a_1 + b_1, a_2 - b_2, a_3 + b_3)$

C.8 浮点型逻辑指令

- _mm_and_ps(__m128 a , __m128 b)

  返回为一个__m128 的寄存器，将寄存器 a 和寄存器 b 的对应位置的 32bit 单精度浮点数分别进行按位与运算。

- mm_andnot_ps(__m128 a , __m128 b)

  返回为一个__m128 的寄存器，将寄存器 a 的每 32bit 单精度浮点数的非和寄存器 b 的对应位置的 32bit 单精度浮点数分别进行按位与运算。

- mm_or_ps(__m128 a , __m128 b)

  返回为一个__m128 的寄存器，将寄存器 a 和寄存器 b 的对应位置的 32bit 单精度浮点数分别进行按位或运算。

- mm_xor_ps(__m128 a , __m128 b)

  返回为一个__m128 的寄存器，将寄存器 a 和寄存器 b 的对应位置的 32bit 单精度浮点数分别进行按位异或运算。

C.9 浮点型比较指令

- _mm_cmpeq_ps (__m128 a , __m128 b)

返回一个__m128 的寄存器，分别比较寄存器 a 和寄存器 b 对应位置 32bit 单精度浮点数是否相等，若相等，该位置返回 0xffff，否则返回 0x0。即 $r_i = (a_i == b_i)$? 0xffff: 0x0

- _mm_cmpneq_ps (__m128 a , __m128 b)

  返回一个__m128 的寄存器，分别比较寄存器 a 和寄存器 b 对应位置 32bit 单精度浮点数是否相等，若不相等，该位置返回 0xffff，否则返回 0x0。即$r_i = (a_i != b_i)$? 0xffff: 0x0

- _mm_cmpgt_ps (__m128 a, __m128 b)

  返回一个__m128 的寄存器，分别比较寄存器 a 的每个 32bit 浮点数是否大于寄存器 b 对应位置 32bit 浮点数，若大于，该位置返回 0xffff，否则返回 0x0。即$r_i = (a_i > b_i)$? 0xffff: 0x0

- _mm_cmpge_ps (__m128 a, __m128 b)

  返回一个__m128 的寄存器，分别比较寄存器 a 的每个 32bit 浮点数是否大于等于寄存器 b 对应位置 32bit 浮点数，若大于等于，该位置返回 0xffff，否则返回 0x0。即$r_i = (a_i \geq b_i)$? 0xffff: 0x0

- _mm_cmplt_ps (__m128 a, __m128 b)

  返回一个__m128 的寄存器，分别比较寄存器 a 的每个 32bit 浮点数是否小于寄存器 b 对应位置 32bit 浮点数，若小于，该位置返回 0xffff，否则返回 0x0。即$r_i = (a_i < b_i)$? 0xffff: 0x0

- _mm_cmple_ps (__m128 a, __m128 b)

  返回一个__m128 的寄存器，分别比较寄存器 a 的每个 32bit 浮点数是否小于等于寄存器 b 对应位置 32bit 浮点数，若小于等于，该位置返回 0xffff，否则返回 0x0。即$r_i = (a_i \leq b_i)$? 0xffff: 0x0

- _mm_ucomieq_ss(__m128 a,__m128 b)

  返回整数，比较寄存器 a 和寄存器 b 最低的 32bit 单精度浮点数是否相等，若相等，该位置返回 0x1，否则返回 0x0。

- _mm_ucomilt_ss(__m128 a,__m128 b)

  返回整数，比较寄存器 a 的最低 32bit 浮点数是否小于寄存器 b 最低 32bit 浮点数，若小于，该位置返回 0x1，否则返回 0x0。

- _mm_ucomigt_ss(__m128 a,__m128 b)

  返回整数，比较寄存器 a 的最低 32bit 浮点数是否大于寄存器 b 最低 32bit 浮点数，若大于，该位置返回 0x1，否则返回 0x0。

C.10 浮点型型读取存储指令

- _mm_load_ps(float * p)

  返回为一个__m128 的寄存器，它将 p 指向的数据读到指定寄存器中，实际使用时，p 一般是通过类型转换得到的。

- _mm_loadr_ps(float * p)

  返回为一个__m128 的寄存器，它将 p 指向的数据按照反序读到指定寄存器中，实际使用时，p 一般是通过类型转换得到的。

- _mm_store_ps(float * p)

  返回为空，它将寄存器 a 中的数据存储到 p 指向的地址中，实际使用时，p 一般是通过类型转换得到的。

- _mm_storer_ps(float * p)

  返回为空，它将寄存器 a 中的数据按照反序存储到 p 指向的地址中，实际使用时，p 一般是通过类型转换得到的。

- _mm_move_ss(__m128 a , __m128 b)

  返回为一个__m128 的寄存器，它将寄存器 a 最低 32bit 浮点数替换为寄存器 b 最低 32bit 浮点数，再将新的寄存器 a 作为返回。

- _mm_set_ps(float z , float y , float x , float w)

  返回一个__m128 的寄存器，使用 4 个具体的 float 型数据来设置寄存器存放数据。

- _mm_setzero_ps(void)

  返回一个__m128 的寄存器，将寄存器中 4 个 32bit 浮点数均置为 0。

- _mm_movehdup_ps(__m128 source)

  返回一个__m128 寄存器，如A = (a_0, a_1, a_2, a_3)，返回值为(a_1, a_1, a_3, a_3)

- _mm_moveldup_ps(__m128 source)

  返回一个__m128 寄存器，如A = (a_0, a_1, a_2, a_3)，返回值为(a_0, a_0, a_2, a_2)