



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

C/C++ Program Design

Lab 8, SIMD and OpenMP

于仕琪



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Intel Intrinsics



SIMD@Intel

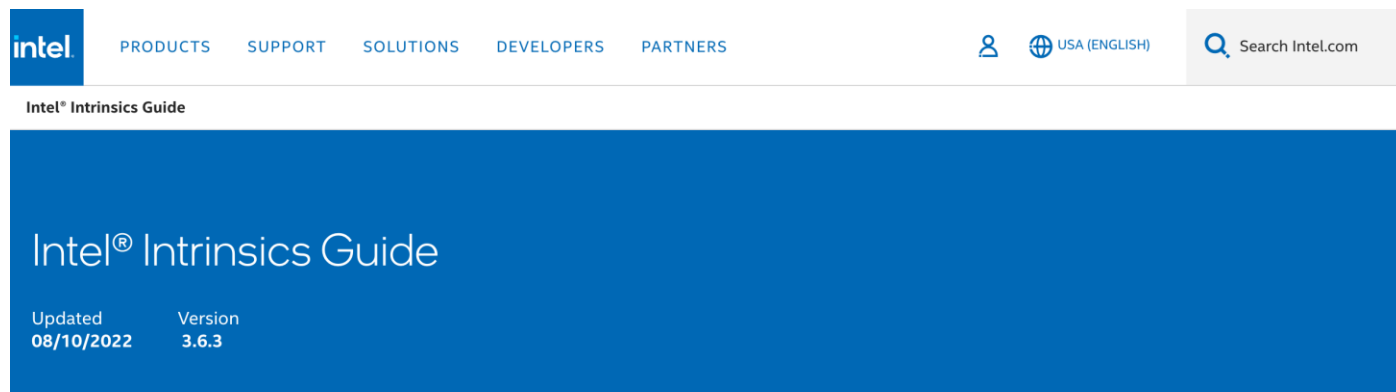
- MMX: 1997, 8 registers, 64 bits,
- SSE (Streaming SIMD Extensions): 1999, 128 bits
- SSE2: 2000
- SSE3: 2004
- SSSE3: 2006
- SSE4.1: 2006
- SSE4.2
- AVX (Advanced Vector Extensions): 2011, 256 bits
- AVX2: 2013
- AVX-512: 2016





Intel® Intrinsics Guide

- <https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html>



Instruction Set

- ☐ MMX
- ☐ SSE
- ☐ SSE2
- ☐ SSE3
- ☐ SSSE3
- ☐ SSE4.1
- ☐ SSE4.2
- ☐ AVX
- ☐ AVX2
- ☐ FMA
- ☐ AVX_VNNI
- ☐ AVX-512
- ☐ KNC
- ☐ AMX
- ☐ SVML
- ☐ Other

Categories

- ☐ Application-Targeted
- ☐ Arithmetic

Search Intel Intrinsics

```
void __mm_2intersect_epi32 (__m128i a, __m128i b, __mmask8* k1, __mmask8* k2)    vp2intersectd
void __mm256_2intersect_epi32 (__m256i a, __m256i b, __mmask8* k1, __mmask8* k2)  vp2intersectd
void __mm512_2intersect_epi32 (__m512i a, __m512i b, __mmask16* k1, __mmask16* k2)  vp2intersectd
void __mm_2intersect_epi64 (__m128i a, __m128i b, __mmask8* k1, __mmask8* k2)    vp2intersectq
void __mm256_2intersect_epi64 (__m256i a, __m256i b, __mmask8* k1, __mmask8* k2)  vp2intersectq
void __mm512_2intersect_epi64 (__m512i a, __m512i b, __mmask8* k1, __mmask8* k2)  vp2intersectq
__m512i __mm512_4dpwssd_epi32 (__m512i src, __m512i a0, __m512i a1, __m512i a2, __m512i a3, __m128i * b)  vp4dpwssd
__m512i __mm512_mask_4dpwssd_epi32 (__m512i src, __mmask16 k, __m512i a0, __m512i a1, __m512i a2, __m512i a3, __m128i * b)  vp4dpwssd
__m512i __mm512_maskz_4dpwssd_epi32 (__mmask16 k, __m512i src, __m512i a0, __m512i a1, __m512i a2, __m512i a3, __m128i * b)  vp4dpwssd
__m512i __mm512_4dpwssds_epi32 (__m512i src, __m512i a0, __m512i a1, __m512i a2, __m512i a3, __m128i * b)  vp4dpwssds
__m512i __mm512_mask_4dpwssds_epi32 (__m512i src, __mmask16 k, __m512i a0, __m512i a1, __m512i a2, __m512i a3, __m128i * b)  vp4dpwssds
```





Load data from memory to registers

`__m256i _mm256_load_epi32 (void const* mem_addr)`

vmovdqa32

Synopsis

`__m256i _mm256_load_epi32 (void const* mem_addr)`
#include <immintrin.h>
Instruction: vmovdqa32 ymm, m256
CPUID Flags: AVX512F + AVX512VL

Description

Load 256-bits (composed of 8 packed 32-bit integers) from memory into `dst.mem_addr` must be aligned on a 32-byte boundary or a general-protection exception may be generated.

Operation

`dst[255:0] := MEM[mem_addr+255:mem_addr]`
`dst[MAX:256] := 0`

Latency and Throughput

Architecture	Latency	Throughput (CPI)
Icelake Intel Core	8	0.5
Icelake Xeon	7	0.56
Skylake	8	0.5

```
float * p = ...;
__m256 a;
a = _mm256_load_ps(p);
```

<code>__m256i _mm256_load_epi64 (void const* mem_addr)</code>	vmovdqa64
<code>__m256d _mm256_load_pd (double const * mem_addr)</code>	vmovapd
<code>__m256h _mm256_load_ph (void const* mem_addr)</code>	vmovaps
<code>__m256 _mm256_load_ps (float const * mem_addr)</code>	vmovaps
<code>__m256i _mm256_load_si256 (__m256i const * mem_addr)</code>	vmovdqa



Add operation

```
__m128 _mm_add_ps (__m128 a, __m128 b)
```

```
__m256 _mm256_add_ps (__m256 a, __m256 b)
```

Synopsis

```
__m256 _mm256_add_ps (__m256 a, __m256 b)
```

```
#include <immintrin.h>
```

```
Instruction: vaddps ymm, ymm, ymm
```

```
CPUID Flags: AVX
```

Description

Add packed single-precision (32-bit) floating-point elements in `a` and `b`, and store the results in `dst`.

Operation

```
FOR j := 0 to 7
    i := j*32
    dst[i+31:i] := a[i+31:i] + b[i+31:i]
ENDFOR
dst[MAX:256] := 0
```

Latency and Throughput

Architecture	Latency	Throughput (CPI)
Alderlake	2	0.5
Icelake Intel Core	4	0.5
Icelake Xeon	4	0.5
Skylake	4	0.5

```
__m256 a, b, c;
```

```
a = _mm256_load_ps(p1 + i);
```

```
b = _mm256_load_ps(p2 + i);
```

```
c = _mm256_add_ps(a, b);
```



Store data from registers to memory

```
void _mm_store_pd (double* mem_addr, __m128d a) movaps
```

```
void _mm256_store_pd (double * mem_addr, __m256d a) vmovaps
```

```
void _mm_store_pd1 (double* mem_addr, __m128d a) .
```

```
void _mm_store_ps (float* mem_addr, __m128 a) movaps
```

Synopsis

```
void _mm_store_ps (float* mem_addr, __m128 a)
#include <immintrin.h>
Instruction: movaps m128, xmm
CPUID Flags: SSE
```

Description

Store 128-bits (composed of 4 packed single-precision (32-bit) floating-point elements) from `a` into memory. `mem_addr` must be aligned on a 16-byte boundary or a general-protection exception may be generated.

Operation

```
MEM[mem_addr+127:mem_addr] := a[127:0]
```

Latency and Throughput

Architecture	Latency	Throughput (CPI)
Alderlake	1	0.5
Skylake	5	1

```
void _mm256_store_ps (float * mem_addr, __m256 a) vmovaps
```

```
void _mm_store_ps1 (float* mem_addr, __m128 a) .
```

```
void _mm_store_sd (double* mem_addr, __m128d a) mov
```

```
void _mm_store_si128 (__m128i* mem_addr, __m128i a) movd
```

```
__m256 c;
```

```
float * p = ...;
```

```
_mm256_store_ps(p, c);
```



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

ARM Neon Intrinsic



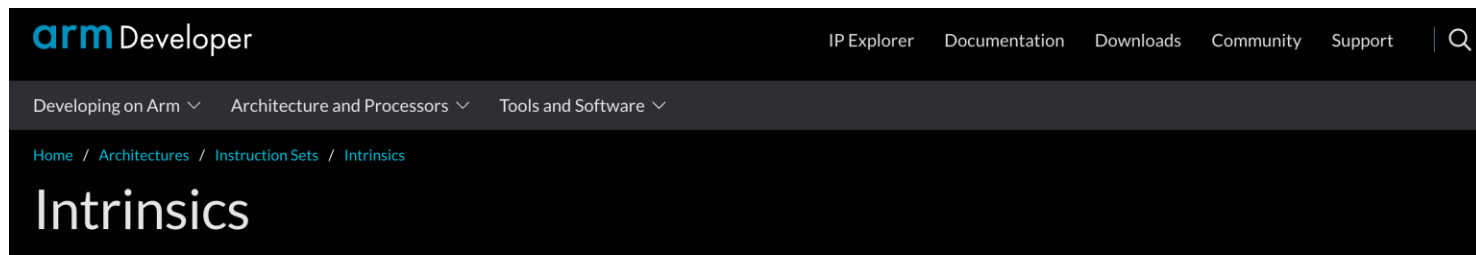
SIMD@ARM

- Neon: 64 bits and 128 bits
- Helium (or MVE): More instructions
- SVE (Scalable Vector Extension): 128 bits to 2048 bits
- SVE2



ARM Intrinsics

- <https://developer.arm.com/architectures/instruction-sets/intrinsics/>



SIMD ISA

☐ Helium

(2471)

☐ Neon

(4344)

☐ sve2

(1900)

☐ sve

(4140)

Return Base Type

☐ float

(2184)

☐ int

(4628)

☐ uint

(4767)

☐ mve_pred

(177)

☐ void

(463)

☐ poly

(371)

Search All Instructions

Search Results

Results 1 - 20 of 12855

	SIMD ISA	Return Type	Name	Arguments	Instruction Group
<div><div>★</div><div>🔗</div></div>	Helium	float16x8_t	[__arm_]vcreateq_f16	(uint64_t a, uint64_t b)	Vector manipulation / Create vector
<div><div>★</div><div>🔗</div></div>	Helium	float32x4_t	[__arm_]vcreateq_f32	(uint64_t a, uint64_t b)	Vector manipulation / Create vector
<div><div>★</div><div>🔗</div></div>	Helium	int8x16_t	[__arm_]vcreateq_s8	(uint64_t a, uint64_t b)	Vector manipulation / Create vector
<div><div>★</div><div>🔗</div></div>	Helium	int16x8_t	[__arm_]vcreateq_s16	(uint64_t a, uint64_t b)	Vector manipulation / Create vector
<div><div>★</div><div>🔗</div></div>	Helium	int32x4_t	[__arm_]vcreateq_s32	(uint64_t a, uint64_t b)	Vector manipulation /





Load data from memory to registers

	SIMD ISA	Return Type	Name	Arguments	
★ ↗	Neon	int8x16_t	vld1q_s8	(int8_t const * ptr)	l
★ ↗	Neon	int16x8_t	vld1q_s16	(int16_t const * ptr)	l
★ ↗	Neon	int32x4_t	vld1q_s32	(int32_t const * ptr)	l
★ ↗	Neon	int64x2_t	vld1q_s64	(int64_t const * ptr)	l
★ ↗	Neon	uint8x16_t	vld1q_u8	(uint8_t const * ptr)	l
★ ↗	Neon	uint16x8_t	vld1q_u16	(uint16_t const * ptr)	l
★ ↗	Neon	uint32x4_t	vld1q_u32	(uint32_t const * ptr)	l
★ ↗	Neon	uint64x2_t	vld1q_u64	(uint64_t const * ptr)	l
★ ↗	Neon	poly64x2_t	vld1q_p64	(poly64_t const * ptr)	l
★ ↗	Neon	float16x8_t	vld1q_f16	(float16_t const * ptr)	l
★ ↗	Neon	float32x4_t	vld1q_f32	(float32_t const * ptr)	l
★ ↗	Load multiple single-element structures to one, two, three, or four registers. This instruction loads multiple single-element structures from memory and writes the result to one, two, three, or four SIMD&FP registers.				
★ ↗					

















Add operation

★	🔗	Neon	uint8x16_t	vaddq_u8	(uint8x16_t a, uint8x16_t b)	Vector arithmetic / Add / Addition
★	🔗	Neon	uint16x8_t	vaddq_u16	(uint16x8_t a, uint16x8_t b)	Vector arithmetic / Add / Addition
★	🔗	Neon	uint32x4_t	vaddq_u32	(uint32x4_t a, uint32x4_t b)	Vector arithmetic / Add / Addition
★	🔗	Neon	uint64x2_t	vaddq_u64	(uint64x2_t a, uint64x2_t b)	Vector arithmetic / Add / Addition
★	🔗	Neon	float32x4_t	vaddq_f32	(float32x4_t a, float32x4_t b)	Vector arithmetic / Add / Addition
Description		Floating-point Add (vector). This instruction adds corresponding vector elements in the two source SIMD&FP registers, writes the result into a vector, and writes the vector to the destination SIMD&FP register. All the values in this instruction are floating-point values.				
Results		Vd.4S → result				
This intrinsic compiles to the following instructions:		FADD Vd.4S, Vn.4S, Vm.4S				
Argument Preparation		a → register: Vn.4S b → register: Vm.4S				
Architectures		v7, A32, A64				



Store data from registers to memory

 	Neon	void	vst1q_u8	(uint8_t * ptr, uint8x16_t val)	Store / Stride
 	Neon	void	vst1q_u16	(uint16_t * ptr, uint16x8_t val)	Store / Stride
 	Neon	void	vst1q_u32	(uint32_t * ptr, uint32x4_t val)	Store / Stride
 	Neon	void	vst1q_u64	(uint64_t * ptr, uint64x2_t val)	Store / Stride
 	Neon	void	vst1q_p64	(poly64_t * ptr, poly64x2_t val)	Store / Stride
 	Neon	void	vst1q_f16	(float16_t * ptr, float16x8_t val)	Store / Stride
 	Neon	void	vst1q_f32	(float32_t * ptr, float32x4_t val)	Store / Stride
Description		Store multiple single-element structures from one, two, three, or four registers. This instruction stores elements to memory from one, two, three, or four SIMD&FP registers, without interleaving. Every element of each register is stored.			
Results		void → result			
This intrinsic compiles to the following instructions:		ST1 {Vt.4S}, [Xn]			
Argument Preparation		ptr → register: Xn val → register: Vt.4S			
Architectures		v7, A32, A64			



Exercise:

Write a program to add 2 `float` vectors whose size should be more than 1M. You can initialize the two vectors with values like `0.f`, `1.f`, `2.f`, ...

- Use pure C source code and SIMD (AVX2 or NEON) separately, and compare their speeds
- Use OpenMP to speed up the addition. Can you get the correct result?