

C++ Library of SIMD Vector Types and Operations

Generated by Doxygen 1.8.13

Contents

1	Introduction	1
1.1	Compiling and running	1
2	Namespace Documentation	1
2.1	vx Namespace Reference	1
2.1.1	Detailed Description	4
2.1.2	Typedef Documentation	4
2.1.3	Function Documentation	14
	Index	17

1 Introduction

This C++ header-only library provides definitions for most common vector types and inline functions to operate on those types. This library relies on [GCC Vector Extensions](#) and architecture specific intrinsics header files, like `immintrin.h` from Intel.

1.1 Compiling and running

Attention

Code compiled with options to enable support for vector instructions, for example, `-mavx` or `-msse4.1`, may not run on machine with CPU that does not support the vector instructions used to generate the program.

2 Namespace Documentation

2.1 vx Namespace Reference

Typedefs

- using `uint128_t` = `__uint128_t`
- using `int128_t` = `__int128_t`
- using `U8x64` = `uint8_t __attribute__((vector_size(64 * sizeof(uint8_t))))`
- using `V64ub` = `U8x64`
- using `U8x32` = `uint8_t __attribute__((vector_size(32 * sizeof(uint8_t))))`
- using `V32ub` = `U8x32`
- using `U8x16` = `uint8_t __attribute__((vector_size(16 * sizeof(uint8_t))))`
- using `V16ub` = `U8x16`
- using `U8x8` = `uint8_t __attribute__((vector_size(8 * sizeof(uint8_t))))`
- using `V8ub` = `U8x8`
- using `U8x4` = `uint8_t __attribute__((vector_size(4 * sizeof(uint8_t))))`

- using `V4ub = U8x4`
- using `U8x2 = uint8_t __attribute__((vector_size(2 * sizeof(uint8_t))))`
- using `V2ub = U8x2`
- using `I8x64 = int8_t __attribute__((vector_size(64 * sizeof(int8_t))))`
- using `V64sb = I8x64`
- using `I8x32 = int8_t __attribute__((vector_size(32 * sizeof(int8_t))))`
- using `V32sb = I8x32`
- using `I8x16 = int8_t __attribute__((vector_size(16 * sizeof(int8_t))))`
- using `V16sb = I8x16`
- using `I8x8 = int8_t __attribute__((vector_size(8 * sizeof(int8_t))))`
- using `V8sb = I8x8`
- using `I8x4 = int8_t __attribute__((vector_size(4 * sizeof(int8_t))))`
- using `V4sb = I8x4`
- using `I8x2 = int8_t __attribute__((vector_size(2 * sizeof(int8_t))))`
- using `V2sb = I8x2`
- using `U16x32 = uint16_t __attribute__((vector_size(32 * sizeof(uint16_t))))`
- using `V32uh = U16x32`
- using `U16x16 = uint16_t __attribute__((vector_size(16 * sizeof(uint16_t))))`
- using `V16uh = U16x16`
- using `U16x8 = uint16_t __attribute__((vector_size(8 * sizeof(uint16_t))))`
- using `V8uh = U16x8`
- using `U16x4 = uint16_t __attribute__((vector_size(4 * sizeof(uint16_t))))`
- using `V4uh = U16x4`
- using `U16x2 = uint16_t __attribute__((vector_size(2 * sizeof(uint16_t))))`
- using `V2uh = U16x2`
- using `I16x32 = int16_t __attribute__((vector_size(32 * sizeof(int16_t))))`
- using `V32sh = I16x32`
- using `I16x16 = int16_t __attribute__((vector_size(16 * sizeof(int16_t))))`
- using `V16sh = I16x16`
- using `I16x8 = int16_t __attribute__((vector_size(8 * sizeof(int16_t))))`
- using `V8sh = I16x8`
- using `I16x4 = int16_t __attribute__((vector_size(4 * sizeof(int16_t))))`
- using `V4sh = I16x4`
- using `I16x2 = int16_t __attribute__((vector_size(2 * sizeof(int16_t))))`
- using `V2sh = I16x2`
- using `U32x16 = uint32_t __attribute__((vector_size(16 * sizeof(uint32_t))))`
- using `V16ui = U32x16`
- using `U32x8 = uint32_t __attribute__((vector_size(8 * sizeof(uint32_t))))`
- using `V8ui = U32x8`
- using `U32x4 = uint32_t __attribute__((vector_size(4 * sizeof(uint32_t))))`
- using `V4ui = U32x4`
- using `U32x2 = uint32_t __attribute__((vector_size(2 * sizeof(uint32_t))))`
- using `V2ui = U32x2`
- using `I32x16 = int32_t __attribute__((vector_size(16 * sizeof(int32_t))))`
- using `V16si = I32x16`
- using `I32x8 = int32_t __attribute__((vector_size(8 * sizeof(int32_t))))`
- using `V8si = I32x8`
- using `I32x4 = int32_t __attribute__((vector_size(4 * sizeof(int32_t))))`
- using `V4si = I32x4`
- using `I32x2 = int32_t __attribute__((vector_size(2 * sizeof(int32_t))))`
- using `V2si = I32x2`
- using `U64x8 = uint64_t __attribute__((vector_size(8 * sizeof(uint64_t))))`
- using `V8ul = U64x8`
- using `U64x4 = uint64_t __attribute__((vector_size(4 * sizeof(uint64_t))))`
- using `V4ul = U64x4`

- using **U64x2** = uint64_t __attribute__((vector_size(2 * sizeof(uint64_t))))
- using **V2ul** = **U64x2**
- using **I64x8** = int64_t __attribute__((vector_size(8 * sizeof(int64_t))))
- using **V8sl** = **I64x8**
- using **I64x4** = int64_t __attribute__((vector_size(4 * sizeof(int64_t))))
- using **V4sl** = **I64x4**
- using **I64x2** = int64_t __attribute__((vector_size(2 * sizeof(int64_t))))
- using **V2sl** = **I64x2**
- using **U128x4** = uint128_t __attribute__((vector_size(4 * sizeof(uint128_t))))
- using **V4uq** = **U128x4**
- using **U128x2** = uint128_t __attribute__((vector_size(2 * sizeof(uint128_t))))
- using **V2uq** = **U128x2**
- using **I128x4** = int128_t __attribute__((vector_size(4 * sizeof(int128_t))))
- using **V4sq** = **I128x4**
- using **I128x2** = int128_t __attribute__((vector_size(2 * sizeof(int128_t))))
- using **V2sq** = **I128x2**
- using **Fx16** = float __attribute__((vector_size(16 * sizeof(float))))
- using **V16f** = **Fx16**
- using **Fx8** = float __attribute__((vector_size(8 * sizeof(float))))
- using **V8f** = **Fx8**
- using **Fx4** = float __attribute__((vector_size(4 * sizeof(float))))
- using **V4f** = **Fx4**
- using **Fx2** = float __attribute__((vector_size(2 * sizeof(float))))
- using **V2f** = **Fx2**
- using **Dx8** = double __attribute__((vector_size(8 * sizeof(double))))
- using **V8d** = **Dx8**
- using **Dx4** = double __attribute__((vector_size(4 * sizeof(double))))
- using **V4d** = **Dx4**
- using **Dx2** = double __attribute__((vector_size(2 * sizeof(double))))
- using **V2d** = **Dx2**

Functions

- template<typename T >
constexpr T **false_vec** ()
Returns 'false' vector {0,0,0,...}.
- template<typename T >
constexpr T **true_vec** ()
Returns 'true' vector {-1,-1,-1,...}.
- template<typename T >
bool **equal** (T a, T b)
- template<typename T >
constexpr unsigned **nrelem** ()
- template<typename Acc , typename T >
Acc **sum** (T v)
- template<typename T >
T **select** (T cond, T a, T b)
- template<typename T , typename M >
T **shuffle** (T a, M mask)
- template<typename T , typename M >
T **shuffle** (T a, T b, M mask)

2.1.1 Detailed Description

Namespace of all vector types and functions.

2.1.2 Typedef Documentation

2.1.2.1 Dx2

```
using vx::Dx2 = typedef double __attribute__ ((vector_size ( 2 *sizeof( double ))))
```

Vector double [2]

2.1.2.2 Dx4

```
using vx::Dx4 = typedef double __attribute__ ((vector_size ( 4 *sizeof( double ))))
```

Vector double [4]

2.1.2.3 Dx8

```
using vx::Dx8 = typedef double __attribute__ ((vector_size ( 8 *sizeof( double ))))
```

Vector double [8]

2.1.2.4 Fx16

```
using vx::Fx16 = typedef float __attribute__ ((vector_size ( 16 *sizeof( float ))))
```

Vector float [16]

2.1.2.5 Fx2

```
using vx::Fx2 = typedef float __attribute__ ((vector_size ( 2 *sizeof( float ))))
```

Vector float [2]

2.1.2.6 Fx4

```
using vx::Fx4 = typedef float __attribute__ ((vector_size ( 4 *sizeof( float ))))
```

Vector float [4]

2.1.2.7 Fx8

```
using vx::Fx8 = typedef float __attribute__ ((vector_size ( 8 *sizeof( float ))))
```

Vector float [8]

2.1.2.8 l128x2

```
using vx::l128x2 = typedef int128_t __attribute__((vector_size ( 2 *sizeof( int128_t ))))
```

Vector int128_t[2]

2.1.2.9 l128x4

```
using vx::l128x4 = typedef int128_t __attribute__((vector_size ( 4 *sizeof( int128_t ))))
```

Vector int128_t[4]

2.1.2.10 l16x16

```
using vx::l16x16 = typedef int16_t __attribute__((vector_size ( 16 *sizeof( int16_t ))))
```

Vector int16_t[16]

2.1.2.11 l16x2

```
using vx::l16x2 = typedef int16_t __attribute__((vector_size ( 2 *sizeof( int16_t ))))
```

Vector int16_t[2]

2.1.2.12 l16x32

```
using vx::l16x32 = typedef int16_t __attribute__((vector_size ( 32 *sizeof( int16_t ))))
```

Vector int16_t[32]

2.1.2.13 l16x4

```
using vx::l16x4 = typedef int16_t __attribute__((vector_size ( 4 *sizeof( int16_t ))))
```

Vector int16_t[4]

2.1.2.14 l16x8

```
using vx::l16x8 = typedef int16_t __attribute__((vector_size ( 8 *sizeof( int16_t ))))
```

Vector int16_t[8]

2.1.2.15 l32x16

```
using vx::l32x16 = typedef int32_t __attribute__((vector_size ( 16 *sizeof( int32_t ))))
```

Vector int32_t[16]

2.1.2.16 l32x2

```
using vx::l32x2 = typedef int32_t __attribute__((vector_size ( 2 *sizeof( int32_t ))))
```

Vector int32_t[2]

2.1.2.17 I32x4

```
using vx::I32x4 = typedef int32_t __attribute__((vector_size ( 4 *sizeof( int32_t ))))
```

Vector int32_t[4]

2.1.2.18 I32x8

```
using vx::I32x8 = typedef int32_t __attribute__((vector_size ( 8 *sizeof( int32_t ))))
```

Vector int32_t[8]

2.1.2.19 I64x2

```
using vx::I64x2 = typedef int64_t __attribute__((vector_size ( 2 *sizeof( int64_t ))))
```

Vector int64_t[2]

2.1.2.20 I64x4

```
using vx::I64x4 = typedef int64_t __attribute__((vector_size ( 4 *sizeof( int64_t ))))
```

Vector int64_t[4]

2.1.2.21 I64x8

```
using vx::I64x8 = typedef int64_t __attribute__((vector_size ( 8 *sizeof( int64_t ))))
```

Vector int64_t[8]

2.1.2.22 I8x16

```
using vx::I8x16 = typedef int8_t __attribute__((vector_size ( 16 *sizeof( int8_t ))))
```

Vector int8_t[16]

2.1.2.23 I8x2

```
using vx::I8x2 = typedef int8_t __attribute__((vector_size ( 2 *sizeof( int8_t ))))
```

Vector int8_t[2]

2.1.2.24 I8x32

```
using vx::I8x32 = typedef int8_t __attribute__((vector_size ( 32 *sizeof( int8_t ))))
```

Vector int8_t[32]

2.1.2.25 I8x4

```
using vx::I8x4 = typedef int8_t __attribute__((vector_size ( 4 *sizeof( int8_t ))))
```

Vector int8_t[4]

2.1.2.26 I8x64

```
using vx::I8x64 = typedef int8_t __attribute__((vector_size ( 64 *sizeof( int8_t ))))
```

Vector int8_t [64]

2.1.2.27 I8x8

```
using vx::I8x8 = typedef int8_t __attribute__((vector_size ( 8 *sizeof( int8_t ))))
```

Vector int8_t [8]

2.1.2.28 U128x2

```
using vx::U128x2 = typedef uint128_t __attribute__((vector_size ( 2 *sizeof( uint128_t ))))
```

Vector uint128_t [2]

2.1.2.29 U128x4

```
using vx::U128x4 = typedef uint128_t __attribute__((vector_size ( 4 *sizeof( uint128_t ))))
```

Vector uint128_t [4]

2.1.2.30 U16x16

```
using vx::U16x16 = typedef uint16_t __attribute__((vector_size ( 16 *sizeof( uint16_t ))))
```

Vector uint16_t [16]

2.1.2.31 U16x2

```
using vx::U16x2 = typedef uint16_t __attribute__((vector_size ( 2 *sizeof( uint16_t ))))
```

Vector uint16_t [2]

2.1.2.32 U16x32

```
using vx::U16x32 = typedef uint16_t __attribute__((vector_size ( 32 *sizeof( uint16_t ))))
```

Vector uint16_t [32]

2.1.2.33 U16x4

```
using vx::U16x4 = typedef uint16_t __attribute__((vector_size ( 4 *sizeof( uint16_t ))))
```

Vector uint16_t [4]

2.1.2.34 U16x8

```
using vx::U16x8 = typedef uint16_t __attribute__((vector_size ( 8 *sizeof( uint16_t ))))
```

Vector uint16_t [8]

2.1.2.35 U32x16

```
using vx::U32x16 = typedef uint32_t __attribute__ ((vector_size ( 16 *sizeof( uint32_t ))))
```

```
Vector uint32_t [ 16 ]
```

2.1.2.36 U32x2

```
using vx::U32x2 = typedef uint32_t __attribute__ ((vector_size ( 2 *sizeof( uint32_t ))))
```

```
Vector uint32_t [ 2 ]
```

2.1.2.37 U32x4

```
using vx::U32x4 = typedef uint32_t __attribute__ ((vector_size ( 4 *sizeof( uint32_t ))))
```

```
Vector uint32_t [ 4 ]
```

2.1.2.38 U32x8

```
using vx::U32x8 = typedef uint32_t __attribute__ ((vector_size ( 8 *sizeof( uint32_t ))))
```

```
Vector uint32_t [ 8 ]
```

2.1.2.39 U64x2

```
using vx::U64x2 = typedef uint64_t __attribute__ ((vector_size ( 2 *sizeof( uint64_t ))))
```

```
Vector uint64_t [ 2 ]
```

2.1.2.40 U64x4

```
using vx::U64x4 = typedef uint64_t __attribute__ ((vector_size ( 4 *sizeof( uint64_t ))))
```

```
Vector uint64_t [ 4 ]
```

2.1.2.41 U64x8

```
using vx::U64x8 = typedef uint64_t __attribute__ ((vector_size ( 8 *sizeof( uint64_t ))))
```

```
Vector uint64_t [ 8 ]
```

2.1.2.42 U8x16

```
using vx::U8x16 = typedef uint8_t __attribute__ ((vector_size ( 16 *sizeof( uint8_t ))))
```

```
Vector uint8_t [ 16 ]
```

2.1.2.43 U8x2

```
using vx::U8x2 = typedef uint8_t __attribute__ ((vector_size ( 2 *sizeof( uint8_t ))))
```

```
Vector uint8_t [ 2 ]
```

2.1.2.44 U8x32

```
using vx::U8x32 = typedef uint8_t __attribute__((vector_size ( 32 *sizeof( uint8_t ))))
```

Vector uint8_t [32]

2.1.2.45 U8x4

```
using vx::U8x4 = typedef uint8_t __attribute__((vector_size ( 4 *sizeof( uint8_t ))))
```

Vector uint8_t [4]

2.1.2.46 U8x64

```
using vx::U8x64 = typedef uint8_t __attribute__((vector_size ( 64 *sizeof( uint8_t ))))
```

Vector uint8_t [64]

2.1.2.47 U8x8

```
using vx::U8x8 = typedef uint8_t __attribute__((vector_size ( 8 *sizeof( uint8_t ))))
```

Vector uint8_t [8]

2.1.2.48 V16f

```
using vx::V16f = typedef Fx16
```

Vector float [16]

2.1.2.49 V16sb

```
using vx::V16sb = typedef I8x16
```

Vector int8_t [16]

2.1.2.50 V16sh

```
using vx::V16sh = typedef I16x16
```

Vector int16_t [16]

2.1.2.51 V16si

```
using vx::V16si = typedef I32x16
```

Vector int32_t [16]

2.1.2.52 V16ub

```
using vx::V16ub = typedef U8x16
```

Vector uint8_t [16]

2.1.2.53 V16uh

```
using vx::V16uh = typedef U16x16
```

Vector uint16_t [16]

2.1.2.54 V16ui

```
using vx::V16ui = typedef U32x16
```

Vector uint32_t [16]

2.1.2.55 V2d

```
using vx::V2d = typedef Dx2
```

Vector double [2]

2.1.2.56 V2f

```
using vx::V2f = typedef Fx2
```

Vector float [2]

2.1.2.57 V2sb

```
using vx::V2sb = typedef I8x2
```

Vector int8_t [2]

2.1.2.58 V2sh

```
using vx::V2sh = typedef I16x2
```

Vector int16_t [2]

2.1.2.59 V2si

```
using vx::V2si = typedef I32x2
```

Vector int32_t [2]

2.1.2.60 V2sl

```
using vx::V2sl = typedef I64x2
```

Vector int64_t [2]

2.1.2.61 V2sq

```
using vx::V2sq = typedef I128x2
```

Vector int128_t [2]

2.1.2.62 V2ub

```
using vx::V2ub = typedef U8x2
```

```
Vector uint8_t [ 2 ]
```

2.1.2.63 V2uh

```
using vx::V2uh = typedef U16x2
```

```
Vector uint16_t [ 2 ]
```

2.1.2.64 V2ui

```
using vx::V2ui = typedef U32x2
```

```
Vector uint32_t [ 2 ]
```

2.1.2.65 V2ul

```
using vx::V2ul = typedef U64x2
```

```
Vector uint64_t [ 2 ]
```

2.1.2.66 V2uq

```
using vx::V2uq = typedef U128x2
```

```
Vector uint128_t [ 2 ]
```

2.1.2.67 V32sb

```
using vx::V32sb = typedef I8x32
```

```
Vector int8_t [ 32 ]
```

2.1.2.68 V32sh

```
using vx::V32sh = typedef I16x32
```

```
Vector int16_t [ 32 ]
```

2.1.2.69 V32ub

```
using vx::V32ub = typedef U8x32
```

```
Vector uint8_t [ 32 ]
```

2.1.2.70 V32uh

```
using vx::V32uh = typedef U16x32
```

```
Vector uint16_t [ 32 ]
```

2.1.2.71 V4d

```
using vx::V4d = typedef Dx4
```

Vector double [4]

2.1.2.72 V4f

```
using vx::V4f = typedef Fx4
```

Vector float [4]

2.1.2.73 V4sb

```
using vx::V4sb = typedef I8x4
```

Vector int8_t [4]

2.1.2.74 V4sh

```
using vx::V4sh = typedef I16x4
```

Vector int16_t [4]

2.1.2.75 V4si

```
using vx::V4si = typedef I32x4
```

Vector int32_t [4]

2.1.2.76 V4sl

```
using vx::V4sl = typedef I64x4
```

Vector int64_t [4]

2.1.2.77 V4sq

```
using vx::V4sq = typedef I128x4
```

Vector int128_t [4]

2.1.2.78 V4ub

```
using vx::V4ub = typedef U8x4
```

Vector uint8_t [4]

2.1.2.79 V4uh

```
using vx::V4uh = typedef U16x4
```

Vector uint16_t [4]

2.1.2.80 V4ui

```
using vx::V4ui = typedef U32x4
```

```
Vector uint32_t [ 4 ]
```

2.1.2.81 V4ul

```
using vx::V4ul = typedef U64x4
```

```
Vector uint64_t [ 4 ]
```

2.1.2.82 V4uq

```
using vx::V4uq = typedef U128x4
```

```
Vector uint128_t [ 4 ]
```

2.1.2.83 V64sb

```
using vx::V64sb = typedef I8x64
```

```
Vector int8_t [ 64 ]
```

2.1.2.84 V64ub

```
using vx::V64ub = typedef U8x64
```

```
Vector uint8_t [ 64 ]
```

2.1.2.85 V8d

```
using vx::V8d = typedef Dx8
```

```
Vector double [ 8 ]
```

2.1.2.86 V8f

```
using vx::V8f = typedef Fx8
```

```
Vector float [ 8 ]
```

2.1.2.87 V8sb

```
using vx::V8sb = typedef I8x8
```

```
Vector int8_t [ 8 ]
```

2.1.2.88 V8sh

```
using vx::V8sh = typedef I16x8
```

```
Vector int16_t [ 8 ]
```

2.1.2.89 V8si

```
using vx::V8si = typedef I32x8
```

```
Vector<int32_t> [ 8 ]
```

2.1.2.90 V8sl

```
using vx::V8sl = typedef I64x8
```

```
Vector<int64_t> [ 8 ]
```

2.1.2.91 V8ub

```
using vx::V8ub = typedef U8x8
```

```
Vector<uint8_t> [ 8 ]
```

2.1.2.92 V8uh

```
using vx::V8uh = typedef U16x8
```

```
Vector<uint16_t> [ 8 ]
```

2.1.2.93 V8ui

```
using vx::V8ui = typedef U32x8
```

```
Vector<uint32_t> [ 8 ]
```

2.1.2.94 V8ul

```
using vx::V8ul = typedef U64x8
```

```
Vector<uint64_t> [ 8 ]
```

2.1.3 Function Documentation**2.1.3.1 equal()**

```
template<typename T >
bool vx::equal (
    T a,
    T b )
```

Compare two vectors for equality.

Returns

true if all elements of two vectors are equal

Example:

```
{c++}
V4si a = {1,2,3,4};
V4si b = {1,2,3,4};
assert(equal(a, b));
assert(equal(a - b, (V4si){0,0,0,0}));
assert(equal(a + b, a * 2));
```

2.1.3.2 nrelem()

```
template<typename T >
constexpr unsigned vx::nrelem ( )
```

Compile-time function that returns number of elements.

```
static_assert(nrelem<U32x8>() == 8 and sizeof(U32x8) == 32);
```

2.1.3.3 select()

```
template<typename T >
T vx::select (
    T cond,
    T a,
    T b )
```

Returns

vector {cond[0]? a[0]:b[0], cond[1] ? a[1]:b[1],...}

2.1.3.4 shuffle()

```
template<typename T , typename M >
T vx::shuffle (
    T a,
    M mask )
```

Shuffle elements according to a rule.

```
{c++}
Fx4 a = {1.1, 2.2, 3.3, 4.4};
U32x4 mask = {3, 2, 1, 0};
assert(equal(shuffle(a, mask), (Fx4){4.4, 3.3, 2.2, 1.1}));
```

2.1.3.5 sum()

```
template<typename Acc , typename T >
Acc vx::sum (
    T v )
```

Returns sum of all elements.

```
V4ui a = {1,2,3,4};
assert(sum<uint32_t>(a) == (1+2+3+4));
```


Index

Dx2
vx, 4

Dx4
vx, 4

Dx8
vx, 4

equal
vx, 14

Fx16
vx, 4

Fx2
vx, 4

Fx4
vx, 4

Fx8
vx, 4

I128x2
vx, 4

I128x4
vx, 5

I16x16
vx, 5

I16x2
vx, 5

I16x32
vx, 5

I16x4
vx, 5

I16x8
vx, 5

I32x16
vx, 5

I32x2
vx, 5

I32x4
vx, 5

I32x8
vx, 6

I64x2
vx, 6

I64x4
vx, 6

I64x8
vx, 6

I8x16
vx, 6

I8x2
vx, 6

I8x32
vx, 6

I8x4
vx, 6

I8x64
vx, 6

I8x8
vx, 7

nrelem
vx, 14

select
vx, 15

shuffle
vx, 15

sum
vx, 15

U128x2
vx, 7

U128x4
vx, 7

U16x16
vx, 7

U16x2
vx, 7

U16x32
vx, 7

U16x4
vx, 7

U16x8
vx, 7

U32x16
vx, 7

U32x2
vx, 8

U32x4
vx, 8

U32x8
vx, 8

U64x2
vx, 8

U64x4
vx, 8

U64x8
vx, 8

U8x16
vx, 8

U8x2
vx, 8

U8x32
vx, 8

U8x4
vx, 9

U8x64
vx, 9

U8x8
vx, 9

V16f

vx, [9](#)
 V16sb
 vx, [9](#)
 V16sh
 vx, [9](#)
 V16si
 vx, [9](#)
 V16ub
 vx, [9](#)
 V16uh
 vx, [9](#)
 V16ui
 vx, [10](#)
 V2d
 vx, [10](#)
 V2f
 vx, [10](#)
 V2sb
 vx, [10](#)
 V2sh
 vx, [10](#)
 V2si
 vx, [10](#)
 V2sl
 vx, [10](#)
 V2sq
 vx, [10](#)
 V2ub
 vx, [10](#)
 V2uh
 vx, [11](#)
 V2ui
 vx, [11](#)
 V2ul
 vx, [11](#)
 V2uq
 vx, [11](#)
 V32sb
 vx, [11](#)
 V32sh
 vx, [11](#)
 V32ub
 vx, [11](#)
 V32uh
 vx, [11](#)
 V4d
 vx, [11](#)
 V4f
 vx, [12](#)
 V4sb
 vx, [12](#)
 V4sh
 vx, [12](#)
 V4si
 vx, [12](#)
 V4sl
 vx, [12](#)
 V4sq
 vx, [12](#)
 V4ub
 vx, [12](#)
 V4uh
 vx, [12](#)
 V4ui
 vx, [12](#)
 V4ul
 vx, [13](#)
 V4uq
 vx, [13](#)
 V64sb
 vx, [13](#)
 V64ub
 vx, [13](#)
 V8d
 vx, [13](#)
 V8f
 vx, [13](#)
 V8sb
 vx, [13](#)
 V8sh
 vx, [13](#)
 V8si
 vx, [13](#)
 V8sl
 vx, [14](#)
 V8ub
 vx, [14](#)
 V8uh
 vx, [14](#)
 V8ui
 vx, [14](#)
 V8ul
 vx, [14](#)
 vx, [1](#)
 Dx2, [4](#)
 Dx4, [4](#)
 Dx8, [4](#)
 equal, [14](#)
 Fx16, [4](#)
 Fx2, [4](#)
 Fx4, [4](#)
 Fx8, [4](#)
 l128x2, [4](#)
 l128x4, [5](#)
 l16x16, [5](#)
 l16x2, [5](#)
 l16x32, [5](#)
 l16x4, [5](#)
 l16x8, [5](#)
 l32x16, [5](#)
 l32x2, [5](#)
 l32x4, [5](#)
 l32x8, [6](#)
 l64x2, [6](#)
 l64x4, [6](#)
 l64x8, [6](#)

I8x16, [6](#)
I8x2, [6](#)
I8x32, [6](#)
I8x4, [6](#)
I8x64, [6](#)
I8x8, [7](#)
nrelem, [14](#)
select, [15](#)
shuffle, [15](#)
sum, [15](#)
U128x2, [7](#)
U128x4, [7](#)
U16x16, [7](#)
U16x2, [7](#)
U16x32, [7](#)
U16x4, [7](#)
U16x8, [7](#)
U32x16, [7](#)
U32x2, [8](#)
U32x4, [8](#)
U32x8, [8](#)
U64x2, [8](#)
U64x4, [8](#)
U64x8, [8](#)
U8x16, [8](#)
U8x2, [8](#)
U8x32, [8](#)
U8x4, [9](#)
U8x64, [9](#)
U8x8, [9](#)
V16f, [9](#)
V16sb, [9](#)
V16sh, [9](#)
V16si, [9](#)
V16ub, [9](#)
V16uh, [9](#)
V16ui, [10](#)
V2d, [10](#)
V2f, [10](#)
V2sb, [10](#)
V2sh, [10](#)
V2si, [10](#)
V2sl, [10](#)
V2sq, [10](#)
V2ub, [10](#)
V2uh, [11](#)
V2ui, [11](#)
V2ul, [11](#)
V2uq, [11](#)
V32sb, [11](#)
V32sh, [11](#)
V32ub, [11](#)
V32uh, [11](#)
V4d, [11](#)
V4f, [12](#)
V4sb, [12](#)
V4sh, [12](#)
V4si, [12](#)
V4sl, [12](#)
V4sq, [12](#)
V4ub, [12](#)
V4uh, [12](#)
V4ui, [12](#)
V4ul, [13](#)
V4uq, [13](#)
V64sb, [13](#)
V64ub, [13](#)
V8d, [13](#)
V8f, [13](#)
V8sb, [13](#)
V8sh, [13](#)
V8si, [13](#)
V8sl, [14](#)
V8ub, [14](#)
V8uh, [14](#)
V8ui, [14](#)
V8ul, [14](#)