MSA Intrinsics Guide

1. MSA Instruction Set Architecture

1.1. Instruction syntax

Instructions have the following general format:

dest = **__builtin_msa_op_suffix_fmt**(src1,src2,src3...)

where
op Operations(for example,add,sub,fadd,fmul)
suffix Modifier

- a(absolute)
- s(signed or saturating)
- u(unsigned)
- r(rounding)

fmt Element data type

- b: 8-bit signed or unsigned integer
- h: 16-bit signed or unsigned integer or half-precision floating-point or 16-bit Q15 fixed point
- w: 32-bit signed or unsigned integer or single floating-point or 32-bit Q31 fixed point
- d: 64-bit integer or double floating-point or 64-bit
- v: 128-bit vector

1.2. Supported data type

The MSA instructions support the following data type:

- Integer format
- Fixed-Point format
- Floating-Point format
- Half-Precision Floating-Point format
- Vector format
- Others

1.2.1. Integer format

The MSA provides 8-bit,16-bit,32-bit,and 64-bit signed and unsigned integers formats. Following table lists all integer types that are supported by MSA.

Table 1.1 MSA supported integral data types

Bits	Name	Range(assuming two's complement for signed)
8	Byte	Signed char: -128 ~ 127（$-2^7 \sim 2^7-1$）
		Unsigned char: 0 ~ 255 （$0 \sim 2^8-1$）
16	Halfword	Signed short: -32768 ~ 32767($-2^{15} \sim 2^{15}-1$)
		Unsigned short: 0 ~ 65536($0 \sim 2^{16}-1$)
32	Word	Signed int: -2,147,483,648 ~ 2,147,483,647 ($-2^{31} \sim 2^{31}-1$)
		Unsigned int: 0 ~ 4,294,967,295($0 \sim 2^{32}-1$)
64	Doubleword	Signed long long: −9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807($-2^{63} \sim 2^{63}-1$)
		Unsigned long long: 0 ~ 18,446,744,073,709,551,615($0 \sim 2^{64}-1$)

1.2.2. Fixed-Point format

The MSA provides two fixed-point data types.

- A 16-bit Q15 fixed point
- A 32-bit Q31 fixed point

The fixed-point values are held in 2's complement format, which is used for signed integers in the CPU. Unsigned fixed-point data types are not provided by the architecture; application software can synthesize computations or unsigned integer from the existing instructions and data types.
Fixed-point data values contain fractional and optional integral parts.

1.2.3. Floating-Point format

The MSA provides the following two floating-point formats.

- float:A 32-bit single-precision floating-point
- double:A 64-bit double-precision floating-point

The floating-point data types represent numeric values as well as the following special entities.

- Two infinities,+∞ and-∞
- Signaling non-numbers(SNaN)
- Quiet non-numbers(QNaN)
- Numbers of the form $(-1)^S 2^E b_0 b_1 b_2 ... b_p\text{-}1$

where:

- s = 0 or 1
- E = any integer between E_min and E_max, inclusive
- b_i = 0 or 1(b_0 is the left most of binary point)
- p the signed-magnitude precision

1.2.4. Half-Precision Floating-Point format

The MSA provides half-precision floating-point format. The half-precision is a binary floating-point computer number format that occupies 16 bits in computer memory.
The IEEE 754 standard specifies a half-precision as having the following format.

- Sign bit : 1bit
- Exponent width : 5 bits
- Significant precision : 11 bits(10 explicitly stored)

The advantage over 8-bit or 16-bit binary integers is that the increased dynamic range allows for more detail to be preserved in highlights and shows for images. The advantage over 32-bit single-precision binary formats is that it requires half of the storage and bandwidth.

1.2.5. Vector format

- v16i8: a vector of sixteen signed 8-bit integers;
- v16u8: a vector of sixteen unsigned 8-bit integers;
- v8i16: a vector of eight signed 16-bit integers;
- v8u16: a vector of eight unsigned 16-bit integers;
- v4i32: a vector of four signed 32-bit integers;
- v4u32: a vector of four unsigned 32-bit integers;
- v2i64: a vector of two signed 64-bit integers;
- v2u64: a vector of two unsigned 64-bit integers;
- v4f32: a vector of four 32-bit floats;
- v2f64: a vector of two 64-bit doubles;

1.2.6. Other

- imm0_1: an integer literal in range 0 to 1;
- imm0_3: an integer literal in range 0 to 3;
- imm0_7: an integer literal in range 0 to 7;
- imm0_15: an integer literal in range 0 to 15;
- imm0_31: an integer literal in range 0 to 31;
- imm0_63: an integer literal in range 0 to 63;
- imm0_255: an integer literal in range 0 to 255;
- imm_n16_15: an integer literal in range -16 to 15;
- imm_n512_511: an integer literal in range -512 to 511;
- imm_n1024_1022: an integer literal in range -512 to 511 left shifted by 1 bit,i.e,-1024,-1022,...,1020,1022;
- imm_n2048_2044: an integer literal in range -512 to 511 left shifted by 2 bit,i.e,-2048,-2044,...,2040,2044;
- imm_n4096_4088: an integer literal in range -512 to 511 left shifted by 3 bit,i.e,4096,-4088,...,4080,4088;
- i32:equal to int;
- i64:equal to long long;
- u32:equal to unsigned int;
- u64:equal to unsigned long long;
- f16: half-precision;
- f32:equal to float;

- f64:equal to double;
- NAN：Floating-point variable invalid values;
- one8:0xff
- one16:0xffff
- one32:0xffff ffff
- one64:0xffff ffff ffff ffff
- one128:0xffff ffff ffff ffff ffff ffff ffff ffff
- zero8:0x00
- zero16:0x0000
- zero32:0x0000 0000
- zero64:0x0000 0000 0000 0000
- zero128:0x0000 0000 0000 0000 0000 0000 0000 0000

2. instruction

2.1. Integer Arithmetic

2.1.1. ADDV.df

2.1.1.1. v16i8 __builtin_msa_addv_b(v16i8,v16i8)

Specific

Vector Add.

v16i8 wd=__builtin_msa_addv_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := ws[0] + wt[0];

wd[1] := ws[1] + wt[1];

.....

wd[15] := ws[15] + wt[15];

Requirements

Header:#include<msa.h>

2.1.1.2. v8i16 __builtin_msa_addv_h(v8i16,v8i16)

Specific

Vector Add.

v8i16 wd = __builtin_msa_addv_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] :=ws[0] + wt[0];

wd[1] :=ws[1] + wt[1];

.....

wd[7] :=ws[7] + wt[7];

Requirements

Header:#include <msa.h>

2.1.1.3. v4i32 __builtin_msa_addv_w(v4i32,v4i32)

Specific

Vector Add.

v4i32 wd = __builtin_msa_addv_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] :=ws[0] + wt[0];

wd[1] :=ws[1] + wt[1];

wd[2] :=ws[2] + wt[2];

wd[3] :=ws[3] + wt[3];

Requirements

Header:#include <msa.h>

2.1.1.4. v2i64 __builtin_msa_addv_d(v2i64,v2i64)

Specific

Vector Add.

v2i64 wd = __builtin_msa_addv_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] :=ws[0] + wt[0];

wd[1] :=ws[1] + wt[1];

Requirements

Header:#include <msa.h>

2.1.2. ADDVI.df

2.1.2.1. v16i8 __builtin_msa_addvi_b(v16i8,imm0_31)

Specific

Vector Add.

v16i8 wd = __builtin_msa_addvi_b(v16i8 ws,imm0_31 u5);

Return Value

wd[0] := ws[0] + u5;

wd[1] := ws[1] + u5;

.....

wd[15] := ws[15] + u5;

Requirements

Header:#include<msa.h>

2.1.2.2. v8i16 __builtin_msa_addvi_h(v8i16,imm0_31)

Specific

Vector Add.

v8i16 wd = __builtin_msa_addvi_h(v8i16 ws,imm0_31 u5);

Return Value

wd[0] := ws[0] + u5;

wd[1] := ws[1] + u5;

.....

wd[7] := ws[7] + u5;

Requirements

Header:#include<msa.h>

2.1.2.3. v4i32 __builtin_msa_addvi_w(v4i32,imm0_31)

Specific

Vector Add.

v4i32 wd = __builtin_msa_addvi_w(v4i32 ws,imm0_31 u5);

Return Value

wd[0] := ws[0] + u5;

wd[1] := ws[1] + u5;

wd[2] := ws[2] + u5;

wd[3] := ws[3] + u5;

Requirements

Header:#include<msa.h>

2.1.2.4. v2i64 __builtin_msa_addvi_d(v2i64,imm0_31)

Specific

Vector Add.

v2i64 wd = __builtin_msa_addvi_d(v2i64 ws,imm0_31 u5);

Return Value

wd[0] := ws[0] + u5;

wd[1] := ws[1] + u5;

Requirements

Header:#include<msa.h>

2.1.3. ADD_A.df

2.1.3.1. v16i8 __builtin_msa_add_a_b(v16i8,v16i8)

Specific

Vector Add Absolute Values

v16i8 wd = __builtin_msa_add_a_b(v16i8 ws,v16i8 wt)

Return Value

wd[0] := abs(ws[0],8) + abs(wt[0],8);

wd[1] := abs(ws[1],8) + abs(wt[1],8)

.....

wd[15] := abs(ws[15],8) + abs(wt[15],8)

Requirements

Header:#inlcude<msa.h>

2.1.3.2. v8i16 __builtin_msa_add_a_h(v8i16,v8i16)

Specific

Vector Add Absolute Values

v8i16 wd = __builtin_msa_add_a_h(v8i16 ws,v8i16 wt)

Return Value

wd[0] := abs(ws[0],16) + abs(wt[0],16);

wd[1] := abs(ws[1],16) + abs(wt[1],16);

.....

wd[7] := abs(ws[7],16) + abs(wt[1],16);

Requirements

Header:#inlcude<msa.h>

2.1.3.3. v4i32 __builtin_msa_add_a_w(v4i32,v4i32)

Specific

Vector Add Absolute Values

v4i32 wd = __builtin_msa_add_a_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := abs(ws[0],32) + abs(wt[0],32);

wd[1] := abs(ws[1],32) + abs(wt[1],32);

wd[2] := abs(ws[2],32) + abs(wt[2],32);

wd[3] := abs(ws[3],32) + abs(wt[3],32);

Requirements

Header:#inlcude<msa.h>

2.1.3.4. v2i64 __builtin_msa_add_a_d(v2i64,v2i64)

Specific

Vector Add Aboslute Values

v2i64 wd = __builtin_msa_add_a_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := abs(ws[0],64) + abs(wt[0],64);

wd[1] := abs(ws[1],64) + abs(wt[1],64);

Requirements

Header:#inlcude<msa.h>

2.1.4. ADDS_A.df

2.1.4.1. v16i8 __builtin_msa_adds_a_b(v16i8,v16i8);

Specific

Vector Saturated Add Absolute Values

v16i8 wd = __builtin_msa_adds_a_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := sats(abs(ws[0],8) + abs(wt[0],8) , 8)

wd[1] := sats(abs(ws[1],8) + abs(wt[1],8) , 8)

.....

wd[15] := sats(abs(ws[15],8) + abs(wt[15],8) ,8)

Requirements

Header:#include<msa.h>

eg:

test_adds_a_b:

unsigned char a[16] = {128,129,3,4,

5,63,64,65,

127,128,129,253,

252,193,192,191};

v16i8 va;

va = __builtin_msa_ld_b(a,0);

va = __builtin_msa_adds_a_b(va,va);

__builtin_msa_st_b(va,a,0);

result:

a[16] = {128,127,6,8,

10,126,127,127,

127,128,127,6,

8,126,127,127};

2.1.4.2. v8i16 __builtin_msa_adds_a_h(v8i16,v8i16)

Specific

Vector Saturated Add Absolute Values

v8i16 wd = __builtin_msa_adds_a_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := sats(abs(ws[0],16) + abs(wt[0],16) , 16)

wd[1] := sats(abs(ws[1],16) + abs(wt[1],16) , 16)

.....

wd[7] := sats(abs(ws[7],16) + abs(wt[7],16) , 16)

Requirements

Header:#include<msa.h>

2.1.4.3. v4i32 __builtin_msa_adds_a_w(v4i32,v4i32)

Specific

Vector Saturated Add Absolute Values

v4i32 wd = __builtin_msa_adds_a_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := sats(abs(ws[0],32) + abs(wt[0],32) ,32)

wd[1] := sats(abs(ws[1],32) + abs(wt[1],32) ,32)

wd[2] := sats(abs(ws[2],32) + abs(wt[2],32) ,32)

wd[3] := sats(abs(ws[3],32) + abs(wt[3],32) ,32)

Requirements

Header:#include<msa.h>

2.1.4.4. v2i64 __builtin_msa_adds_a_d(v2i64,v2i64)

Specific

Vector Saturated Add Absolute Values

v2i64 wd = __builtin_msa_adds_a_d(v2i64 ws,v2i64 wt)

Return Value

wd[0] := sats(abs(ws[0],64) + abs(wt[0],64) ,32)

wd[1] := sats(abs(ws[1],64) + abs(wt[1],64) ,32)

Requirements

Header:#include<msa.h>

2.1.5. HADD_S.df

2.1.5.1. v8i16 __builtin_msa_hadd_s_h(v16i8,v16i8)

Specific

Vector Signed Horizontal Add

v8i16 wd = __builtin_msa_hadd_s_h(v16i8 ws,v16i8 wt);

Return Value

wd[0] := ws[1] + wt[0];

wd[1] := ws[3] + wt[2];

.....

wd[7] := ws[15] + wt[14];

Requirements

Header:#include<msa.h>

eg:

```
signed char a[16] = {12,29,3,4,
                     5,63,64,65,
                     -7,12,9,25,
                     52,93,19,91};
short c[8];
v16i8 va;
v8i16 vc;
va = __builtin_msa_ld_b(a,0);
vc = __builtin_msa_hadd_s_h(va,va);
__builtin_msa_st_h(vc,c,0);
```

result:

c[8]={-17,7,68,129,5,34,145,110};

2.1.5.2. v4i32 __builtin_msa_hadd_s_w(v8i16,v8i16)

Specific

Vector Signed Horizontal Add

```
v4i32 wd = __builtin_msa_hadd_s_w(v8i16 ws,v8i16 wd);
```

Return Value

```
wd[0] = ws[1] + wt[0];
wd[1] = ws[3] + wt[2];
```

wd[2] = ws[5] + wt[4];

wd[3] = ws[7] + wt[6];

Requirements

Header:#include<msa.h>

2.1.5.3. v2i64 __builtin_msa_hadd_s_d(v4i32,v4i32)

Specific

Vector Signed Horizontal Add

v2i64 wd = __builtin_msa_hadd_s_d(v4i32 ws,v4i32 wd);

Return Value

wd[0] = ws[1] + wt[0];

wd[1] = ws[3] + wt[2];

Requirements

Header:#include<msa.h>

2.1.6. HADD_U.df

2.1.6.1. v8u16 __builtin_msa_hadd_u_h(v16u8,v16u8)

Specific

Vector Unsigned Horizontal Add

v8u16 wd = __builtin_msa_hadd_u_h(v16u8 ws,v16u8 wt);

Return Value

wd[0] := ws[1] + wt[0];

wd[1] := ws[3] + wt[2];

.....

wd[7] :=ws[15] + wt[14];

Requirements

Header:#include<msa.h>

2.1.6.2. v4u32 __builtin_msa_hadd_u_w(v8u16,v8u16)

Specific

Vector Unsigned Horizontal Add

v4u32 wd = __builtin_msa_hadd_u_w(v8u16 ws,v8u16 wt);

Return Value

wd[0] := ws[1] + wt[0];

wd[1] := ws[3] + wt[2];

wd[2] := ws[5] + wt[4];

wd[3] := ws[7] + wt[6];

Requirements

Header:#include<msa.h>

2.1.6.3. v2u64 __builtin_msa_hadd_u_d(v4u32,v4u32)

Specific

Vector Unsigned Horizontal Add

v2u64 wd = __builtin_msa_hadd_u_d(v4u32 ws,v4u32 wt);

Return Value

wd[0] :=ws[1] + wt[0];

wd[1] :=ws[3] + wt[2];

Requirements

Header:#include<msa.h>

2.1.7. ASUB_S.df

2.1.7.1. v16i8 __builtin_msa_asub_s_b(v16i8,v16i8)

Specific

Vector Absolute Values of Signed Subtract.

v16i8 wd = __builtin_msa_asub_s_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := abs(ws[0] - wt[0] , 8);

wd[1] := abs(ws[1] - wt[1] , 8);

.....

wd[15] := abs(ws[15] - wt[15] , 8);

Requirements

Header:#include<msa.h>

2.1.7.2. v8i16 __builtin_msa_asub_s_h(v8i16,v8i16)

Specific

Vector Absolute Values of Signed Subtract.

v8i16 wd = __builtin_msa_asub_s_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := abs(ws[0] - wt[0] , 16);

wd[1] := abs(ws[1] - wt[1] , 16);

.....

wd[7] := abs(ws[7] - wt[7] , 16);

Requirements

Header:#include<msa.h>

2.1.7.3. v4i32 __builtin_msa_asub_s_w(v4i32,v4i32)

Specific

Vector Absolute Values of Signed Subtract.

v4i32 wd = __builtin_msa_asub_s_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := abs(ws[0] - wt[0] , 32);

wd[1] := abs(ws[1] - wt[1] , 32);

wd[2] := abs(ws[2] - wt[2] , 32);

wd[3] := abs(ws[3] - wt[3] , 32);

Requirements

Header:#include<msa.h>

2.1.7.4. v2i64 __builtin_msa_asub_s_d(v2i64,v2i64)

Specific

Vector Absolute Values of Signed Subtract.

v2i64 wd = __builtin_msa_asub_s_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := abs(ws[0] - wt[0] , 64);

wd[1] := abs(ws[1] - wt[1] , 64);

Requirements

Header:#include<msa.h>

2.1.8. ASUB_U.df

2.1.8.1. v16u8 __builtin_msa_asub_u_b(v16u8,v16u8)

Specific

Vector Absolute Values of Unsigned Subtract.

v16u8 wd = __builtin_msa_asub_u_b(v16u8 ws,v16u8 wt);

Return Value

wd[0] := abs(ws[0] - wt[0] , 8);

wd[1] := abs(ws[1] - wt[1] , 8);

.....

wd[15] := abs(ws[15] - wt[15] , 8);

Requirements

Header:#include<msa.h>

2.1.8.2. v8u16 __builtin_msa_asub_u_h(v8u16,v8u16)

Specific

Vector Absolute Values of Unsigned Subtract.

v8u16 wd = __builtin_msa_asub_u_h(v8u16 ws,v8u16 wt);

Return Value

wd[0] := abs(ws[0] - wt[0] , 16);

wd[1] := abs(ws[1] - wt[1] , 16);

.....

wd[7] := abs(ws[7] - wt[7] , 16);

Requirements

Header:#include<msa.h>

2.1.8.3. v4u32 __builtin_msa_asub_u_w(v4u32,v4u32)

Specific

Vector Absolute Values of Unsigned Subtract.

v4u32 wd = __builtin_msa_asub_u_w(v4u32 ws,v4u32 wt);

Return Value

wd[0] := abs(ws[0] - wt[0] , 32);

wd[1] := abs(ws[1] - wt[1] , 32);

wd[2] := abs(ws[2] - wt[2] , 32);

wd[3] := abs(ws[3] - wt[3] , 32);

Requirements

Header:#include<msa.h>

2.1.8.4. v2u64 __builtin_msa_asub_u_d(v2u64,v2u64)

Specific

Vector Absolute Values of Unsigned Subtract.

v2u64 wd = __builtin_msa_asub_u_d(v2u64 ws,v2u64 wt);

Return Value

wd[0] := abs(ws[0] - wt[0] , 64);

wd[1] := abs(ws[1] - wt[1] , 64);

Requirements

Header:#include<msa.h>

2.1.9. AVER_S.df

2.1.9.1. v16i8 __builtin_msa_aver_s_b(v16i8,v16i8)

Specific

Vector Signed Average With Rounding.

v16i8 wd = __builtin_msa_aver_s_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := (ws[0] + wt[0]+1) / 2;

wd[1] := (ws[1] + wt[1]+1) / 2;

.....

wd[15] := (ws[15] + wt[15]+1) / 2;

Requirements

Header:#include<msa.h>

2.1.9.2. v8i16 __builtin_msa_aver_s_h(v8i16,v8i16)

Specific

Vector Signed Average With Rounding.

v8i16 wd = __builtin_msa_aver_s_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := (ws[0] + wt[0]+1) / 2;

wd[1] := (ws[1] + wt[1]+1) / 2;

.....

wd[7] := (ws[7] + wt[7]+1) / 2;

Requirements

Header:#include<msa.h>

2.1.9.3. v4i32 __builtin_msa_aver_s_w(v4i32,v4i32)

Specific

Vector Signed Average With Rounding.

v4i32 wd = __builtin_msa_aver_s_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := (ws[0] + wt[0]+1) / 2;

wd[1] := (ws[1] + wt[1]+1) / 2;

wd[2] := (ws[2] + wt[2]+1) / 2;

wd[3] := (ws[3] + wt[3]+1) / 2;

Requirements

Header:#include<msa.h>

2.1.9.4. v2i64 __builtin_msa_aver_s_d(v2i64,v2i64)

Specific

Vector Signed Average With Rounding.

v2i64 wd = __builtin_msa_aver_s_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := (ws[0] + wt[0]+1) / 2;

wd[1] := (ws[1] + wt[1]+1) / 2;

Requirements

Header:#include<msa.h>

2.1.10. AVER_U.df

2.1.10.1. v16u8 __builtin_msa_aver_u_b(v16u8,v16u8)

Specific

Vector Unsigned Average With Rounding.

v16u8 wd = __builtin_msa_aver_u_b(v16u8 ws,v16u8 wt);

Return Value

wd[0] := (ws[0] + wt[0]+1) / 2;

wd[1] := (ws[1] + wt[1]+1) / 2;

.....

wd[15] := (ws[15] + wt[15]+1) / 2;

Requirements

Header:#include<msa.h>

2.1.10.2. v8u16 __builtin_msa_aver_u_h(v8u16,v8u16)

Specific

Vector Unsigned Average With Rounding.

v8u16 wd = __builtin_msa_aver_u_h(v8u16 ws,v8u16 wt);

Return Value

wd[0] := (ws[0] + wt[0]+1) / 2;

wd[1] := (ws[1] + wt[1]+1) / 2;

.....

wd[7]:= (ws[7] + wt[7]+1) / 2;

Requirements

Header:#include<msa.h>

2.1.10.3. v4u32 __builtin_msa_aver_u_w(v4u32,v4u32)

Specific

Vector Unsigned Average With Rounding.

v4u32 wd = __builtin_msa_aver_u_w(v4u32 ws,v4u32 wt)

Return Value

wd[0] := (ws[0] + wt[0]+1) / 2;

wd[1] := (ws[1] + wt[1]+1) / 2;

wd[2] := (ws[2] + wt[2]+1) / 2;

wd[3] := (ws[3] + wt[3]+1) / 2;

Requirements

Header:#include<msa.h>

2.1.10.4. v2u64 __builtin_msa_aver_u_d(v2u64,v2u64)

Specific

Vector Unsigned Average With Rounding.

v2u64 wd = __builtin_msa_aver_u_d(v2u64 ws,v2u64 wt);

Return Value

wd[0] := (ws[0] + wt[0]+1) / 2;

wd[1] := (ws[1] + wt[1]+1) / 2;

Requirements

Header:#include<msa.h>

2.1.11. DOTP_S.df

2.1.11.1. v8i16 __builtin_msa_dotp_s_h(v16i8,v16i8)

Specific

Vector Signed Dot Product.

v8i16 wd = __builtin_msa_dotp_s_h(v16i8 ws,v16i8 wt);

Return Value

wd[0] :=ws[0]*wt[0] + ws[1]*wt[1];

wd[1] :=ws[2]*wt[2] + ws[3]*wt[3];

.....

wd[7] :=ws[14]*wt[14] + ws[15]*wt[15];

Requirements

Header:#include<msa.h>

eg:

test_dotp_s:

```
signed char a[16] = {0,1,2,3,
                     4,5,6,7,
                     8,9,10,11,
                     12,13,14,15};
signed char b[16] = {1,2,3,4,
                     5,6,7,8,
                     9,10,11,12,
                     13,14,15,16};
short c[8];
v16i8 va,vb;
v8i16 vc;
```

va = __builtin_msa_ld_b(a,0);

vb = __builtin_msa_ld_b(b,0);

vc = __builtin_msa_dotp_s_h(va,vb);

__builtin_msa_st_h(vc,c,0);

result:

c[8] = {2,18,50,98,162,242,338,450}

2.1.11.2. v4i32 __builtin_msa_dotp_s_w(v8i16,v8i16)

Specific

Vector Signed Dot Product.

v4i32 wd = __builtin_msa_dotp_s_w(v8i16 ws,v8i16 wt);

Return Value

wd[0] := ws[0]*wt[0] + ws[1]*wt[1];

wd[1] := ws[2]*wt[2] + ws[3]*wt[3];

wd[2] := ws[4]*wt[4] + ws[5]*wt[5];

wd[3] := ws[6]*ws[6] + ws[7]*wt[7];

Requirements

Header:#include<msa.h>

2.1.11.3. v2i64 __builtin_msa_dotp_s_d(v4i32,v4i32)

Specific

Vector Signed Dot Product.

v2i64 wd = __builtin_msa_dotp_s_d(v4i32 ws,v4i32 wt)

Return Value

wd[0] := ws[0]*wt[0] + ws[1]*wt[1];

wd[1] := ws[2]*wt[2] + ws[3]*wt[3];

Requirements

Header:#include<msa.h>

2.1.12. DOTP_U.df

2.1.12.1. v8u16 __builtin_msa_dotp_u_h(v16u8,v16u8)

Specific

Vector Unsigned Dot Product.

v8u16 wd = __builtin_msa_dotp_u_h(v16u8 ws,v16u8 wt);

Return Value

wd[0] := ws[0]*wt[0] + ws[1]*wt[1];

wd[1] := ws[2]*wt[2] + ws[3]*wt[3];

.....

wd[7] := ws[14]*wt[14] + ws[15]*wt[15];

Requirements

Header:#include<msa.h>

2.1.12.2. v4u32 __builtin_msa_dotp_u_w(v8u16,v8u16)

Specific

Vector Unsigned Dot Product.

v4u32 wd = __builtin_msa_dotp_u_w(v8u16 ws,v8u16 wt);

Return Value

wd[0] := ws[0]*wt[0] + ws[1]*wt[1];

wd[1] := ws[2]*wt[2] + ws[3]*wt[3];

wd[2] := ws[4]*wt[4] + ws[5]*wt[5];

wd[3] := ws[6]*wt[6] + ws[7]*wt[7];

Requirements

Header:#include<msa.h>

2.1.12.3. v2u64 __builtin_msa_dotp_u_d(v4u32,v4u32)

Specific

Vector Unsigned Dot Product.

v2u64 wd = __builtin_msa_dotp_u_d(v4u32 ws,v4u32 wt);

Return Value

wd[0] := ws[0]*wt[0] + ws[1]*wt[1];

wd[1] := ws[2]*wt[2] + ws[3]*wt[3];

Requirements

Header:#include<msa.h>

2.1.13. DPADD_S.df

2.1.13.1. v8i16 __builtin_msa_dpadd_s_h(v8i16,v16i8,v16i8)

Specific

Vector Signed Dot Product and Add.

v8i16 wd = __builtin_msa_dpadd_s_h(v8i16 wd,v16i8 ws,v16i8 wt);

Return Value

wd[0] := wd[0] + ws[0]*wt[0] + ws[1]*wt[1];

wd[1] := wd[1] + ws[2]*wt[2] + ws[3]*wt[3];

.....

wd[7] := wd[7] + ws[14]*wt[14] + ws[15]*wt[15];

Requirements

Header:#include<msa.h>

2.1.13.2. v4i32 __builtin_msa_dpadd_s_w(v4i32,v8i16,v8i16)

Specific

Vector Signed Dot Product and Add

v4i32 wd = __builtin_msa_dpadd_s_w(v4i32 wd,v8i16 ws,v8i16 wt);

Return Value

wd[0] := wd[0] + ws[0]*wt[0] + ws[1]*wt[1];

wd[1] := wd[1] + ws[2]*wt[2] + ws[3]*wt[3];

wd[2] := wd[2] + ws[4]*wt[4] + ws[5]*wt[5];

wd[3] := wd[3] + ws[6]*wt[6] + ws[7]*wt[7];

Requirements

Header:#include<msa.h>

2.1.13.3. v2i64 __builtin_msa_dpadd_s_d(v2i64,v4i32,v4i32)

Specific

Vector Signed Dot Product and Add

v2i64 wd = __builtin_msa_dpadd_s_d(v2i64,v4i32,v4i32);

Return Value

wd[0] := wd[0] + ws[0]*wt[0] + ws[1]*wt[1];

wd[1] := wd[1] + ws[2]*wt[2] + ws[3]*wt[3];

Requirements

Header:#include<msa.h>

2.1.14. DPADD_U.df

2.1.14.1. v8u16 __builtin_msa_dpadd_u_h(v8u16,v16u8,v16u8)

Specific

Vector Unsigned Dot Product and Add.

v8u16 wd = __builtin_msa_dpadd_u_h(v8u16 wd,v16u8 ws,v16u8 wt);

Return Value

wd[0] := wd[0] + ws[0]*wt[0] + ws[1]*wt[1];

wd[1] := wd[1] + ws[2]*wt[2] + ws[3] *wt[3];

.....

wd[7] := wd[7] + ws[14]*wt[14] + ws[15]*wt[15];

Requirements

Header:#include<msa.h>

2.1.14.2. v4u32

__builtin_msa_dpadd_u_w(v4u32,v8u16,v8u16)

Specific

Vector Unsigned Dot Product and Add.

v4u32 wd = __builtin_msa_dpadd_u_w(v4u32 wd,v8u16 ws,v8u16 wt);

Return Value

wd[0] := wd[0] + ws[0]*wt[0] + ws[1]*wt[1];

wd[1] := wd[1] + ws[2]*wt[2] + ws[3]*wt[3];

wd[2] := wd[2] + ws[4]*wt[4] + ws[5]*wt[5];

wd[3] := wd[3] + ws[6]*wt[6] + ws[7]*wt[7];

Requirements

Header:#include<msa.h>

2.1.14.3. v2u64

__builtin_msa_dpadd_u_d(v2u64,v4u32,v4u32)

Specific

Vector Unsigned Dot Product and Add.

v2u64 wd = __builtin_msa_dpadd_u_d(v2u64 wd,v4u32 ws,v4u32 wt);

Return Value

wd[0] := wd[0] + ws[0]*wt[0] + ws[1]*wt[1];

wd[1] := wd[1] + ws[2]*wt[2] + ws[3]*wt[3];

Requirements

Header:#include<msa.h>

2.1.15. DPSUB_S.df

2.1.15.1. v8i16 __builtin_msa_dpsub_s_h(v8i16,v16i8,v16i8)

Specific

Vector Signed Dot Product and Subtract.

v8i16 wd = __builtin_msa_dpsub_s_h(v8i16 wd,v16i8 ws,v16i8 wt);

Return Value

wd[0] := wd[0] - (ws[0]*wt[0] + ws[1]*wt[1]);

wd[1] := wd[1] - (ws[2]*wt[2] + ws[3]*wt[3]);

.....

wd[7] := wd[7] - (ws[14]*wt[14] + ws[15]*wt[15]);

Requirements

Header:#include<msa.h>

2.1.15.2. v4i32 __builtin_msa_dpsub_s_w(v4i32,v8i16,v8i16)

Specific

Vector Signed Dot Product and Subtract.

v4i32 wd = __builtin_msa_dpsub_s_w(v4i32 wd,v8i16 ws,v8i16 wt);

Return Value

wd[0] := wd[0] - (ws[0]*wt[0] + ws[1]*wt[1]);

wd[1] := wd[1] - (ws[2]*wt[2] + ws[3]*wt[3]);

wd[2] := wd[2] - (ws[4]*wt[4] + ws[5]*wt[5]);

wd[3] := wd[3] - (ws[6]*wt[6] + ws[7]*wt[7]);

Requirements

Header:#include<msa.h>

2.1.15.3. v2i64 __builtin_msa_dpsub_s_d(v2i64,v4i32,v4i32)

Specific

Vector Signed Dot Product and Subtract.

v2i64 wd = __builtin_msa_dpsub_s_d(v2i64 wd,v4i32 ws,v4i32 wt);

Return Value

wd[0] := wd[0] - (ws[0]*wt[0] + ws[1]*wt[1]);

wd[1] := wd[1] - (ws[2]*wt[2] + ws[3]*wt[3]);

Requirements

Header:#include<msa.h>

2.1.16. DPSUB_U.df

2.1.16.1. v8i16

__builtin_msa_dpsub_u_h(v8i16,v16u8,v16u8)

Specific

Vector Unsigned Dot Product and Subtract.

v8i16 wd __builtin_msa_dpsub_u_h(v8i16 wd,v16u8 ws,v16u8 wt);

Return Value

wd[0] := wd[0] - (ws[0]*wt[0] + ws[1]*wt[1]);

wd[1] := wd[1] - (ws[2]*wt[2] + ws[3]*wt[3]);

.....

wd[7] := wd[7] - (ws[14]*wt[14] + ws[15]*wt[15]);

Requirements

Header:#include<msa.h>

2.1.16.2. v4i32

__builtin_msa_dpsub_u_w(v4i32,v8u16,v8u16)

Specific

Vector Unsigned Dot Product and Subtract.

v4i32 wd __builtin_msa_dpsub_u_w(v4i32 wd,v8u16 ws,v8u16 wt);

Return Value

wd[0] := wd[0] - (ws[0]*wt[0] + ws[1]*wt[1]);

wd[1] := wd[1] - (ws[2]*wt[2] + ws[3]*wt[3]);

wd[2] := wd[2] - (ws[4]*wt[4] + ws[5]*wt[5]);

wd[3] := wd[3] - (ws[6]*wt[6] + ws[7]*wt[7]);

Requirements

Header:#include<msa.h>

2.1.16.3. v2i64 __builtin_msa_dpsub_u_d(v2i64,v4u32,v4u32)

Specific

Vector Unsigned Dot Product and Subtract.

v2i64 wd = __builtin_msa_dpsub_u_d(v2i64 wd,v4u32 ws,v4u32 wt);

Return Value

wd[0] := wd[0] - (ws[0]*wt[0] + ws[1]*wt[1]);

wd[1] := wd[1] - (ws[2]*wt[2] + ws[3]*wt[3]);

Requirements

Header:#include<msa.h>

2.1.17. DIV_S.df

2.1.17.1. v16i8 __builtin_msa_div_s_b(v16i8,v16i8)

Specific

Vector Signed Divide.

v16i8 wd = __builtin_msa_div_s_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := ws[0] / wt[0];

wd[1] := ws[1] / wt[1];

.....

wd[15] := ws[15] / wt[15];

Requirements

Header:#include<msa.h>

2.1.17.2. v8i16 __builtin_msa_div_s_h(v8i16,v8i16)

Specific

Vector Signed Divide.

v8i16 wd = __builtin_msa_div_s_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := ws[0] / wt[0];

wd[1] := ws[1] / wt[1];

.....

wd[7] := ws[7] / wt[7];

Requirements

Header:#include<msa.h>

2.1.17.3. v4i32 __builtin_msa_div_s_w(v4i32,v4i32)

Specific

Vector Signed Divide.

v4i32 wd = __builtin_msa_div_s_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := ws[0] / wt[0];

wd[1] := ws[1] / wt[1];

wd[2] := ws[2] / wt[2];

wd[3] := ws[3] / wt[3];

Requirements

Header:#include<msa.h>

2.1.17.4. v2i64 __builtin_msa_div_s_d(v2i64,v2i64)

Specific

Vector Signed Divide.

v2i64 wd = __builtin_msa_div_s_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := ws[0] / wt[0];

wd[1] := ws[1] / wt[1];

Requirements

Header:#include<msa.h>

2.1.18. DIV_U.df

2.1.18.1. v16u8 __builtin_msa_div_u_b(v16u8,v16u8)

Specific

Vector Unsigned Divide.

v16u8 wd = __builtin_msa_div_u_b(v16u8 ws,v16u8 wt);

Return Value

wd[0] := ws[0] / wt[0];

wd[1] := ws[1] / wt[1];

.....

wd[15] := ws[15] / wt[15];

Requirements

Header:#include<msa.h>

2.1.18.2. v8u16 __builtin_msa_div_u_h(v8u16,v8u16)

Specific

Vector Unsigned Divide.

v8u16 wd = __builtin_msa_div_u_h(v8u16 ws,v8u16 wt);

Return Value

wd[0] := ws[0] / wt[0];

wd[1] := ws[1] / wt[1];

.....

wd[7] := ws[7] / wt[7];

Requirements

Header:#include<msa.h>

2.1.18.3. v4u32 __builtin_msa_div_u_w(v4u32,v4u32)

Specific

Vector Unsigned Divide.

v4u32 wd = __builtin_msa_div_u_w(v4u32 ws,v4u32 wt);

Return Value

wd[0] := ws[0] / wt[0];

wd[1] := ws[1] / wt[1];

wd[2] := ws[2] / wt[2];

wd[3] := ws[3] / wt[3];

Requirements

Header:#include<msa.h>

2.1.18.4. v2u64 __builtin_msa_div_u_d(v2u64,v2u64)

Specific

Vector Unsigned Divide.

v2u64 wd = __builtin_msa_div_u_d(v2u64 ws,v2u64 wt);

Return Value

wd[0] := ws[0] / wt[0];

wd[1] := ws[1] / wt[1];

Requirements

Header:#include<msa.h>

2.1.19. MADDV.df

2.1.19.1. v16i8 __builtin_msa_maddv _b(v16i8,v16i8,v16i8)

Specific

Vector Multiply-Add.

v16i8 wd = __builtin_msa_maddv _b(v16i8 wd,v16i8 ws,v16i8 wt);

Return Value

wd[0] := wd[0] + ws[0]*wt[0];

wd[1] := wd[1] + ws[1]*wt[1];

.....

wd[15] := wd[15] + ws[15]*wt[15];

Requirements

Header:#include<msa.h>

2.1.19.2. v8i16 __builtin_msa_maddv_h(v8i16,v8i16,v8i16)

Specific

Vector Multiply-Add.

v8i16 wd = __builtin_msa_maddv_h(v8i16 wd,v8i16 ws,v8i16 wt);

Return Value

wd[0] := wd[0] + ws[0]*wt[0];

wd[1] := wd[1] + ws[1]*wt[1];

.....

wd[7] := wd[7] + ws[7]*wt[7];

Requirements

Header:#include<msa.h>

2.1.19.3. v4i32 __builtin_msa_maddv_w(v4i32,v4i32,v4i32)

Specific

Vector Multiply-Add.

v4i32 wd = __builtin_msa_maddv_w(v4i32 wd,v4i32 ws,v4i32 wt);

Return Value

wd[0] := wd[0] + ws[0]*wt[0];

wd[1] := wd[1] + ws[1]*wt[1];

wd[2] := wd[2] + ws[2]*wt[2];

wd[3] := wd[3] + ws[3]*wt[3];

Requirements

Header:#include<msa.h>

2.1.19.4. v2i64 __builtin_msa_maddv_d(v2i64,v2i64,v2i64)

Specific

Vector Multiply-Add.

v2i64 wd = __builtin_msa_maddv_d(v2i64 wd,v2i64 ws,v2i64 wt);

Return Value

wd[0] := wd[0] + ws[0]*wt[0];

wd[1] := wd[1] + ws[1]*wt[1];

Requirements

Header:#include<msa.h>

2.1.20. MAX_A.df

2.1.20.1. v16i8 __builtin_msa_max_a_b(v16i8,v16i8)

Specific

Vector Maximum of Absolute Values.

v16i8 wd = __builtin_msa_max_a_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := (abs(ws[0] , 8) > abs(wt[0] , 8)) ? ws[0] : wt[0];

wd[1] := (abs(ws[1] , 8) > abs(wt[1] , 8)) ? ws[1] : wt[1];

...

w15 := (abs(ws[15] , 8) > abs(wt[15] , 8)) ? ws[15] : wt[15];

Requirements

Header:#include<msa.h>

2.1.20.2. v8i16 __builtin_msa_max_a_h(v8i16,v8i16)

Specific

Vector Maximum of Absolute Values.

v8i16 wd = __builtin_msa_max_a_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := (abs(ws[0] , 16) > abs(wt[0] , 16)) ? ws[0] : wt[0];

wd[1] := (abs(ws[1] , 16) > abs(wt[1] , 16)) ? ws[1] : wt[1];

...

wd[7] := (abs(ws[7] , 16) > abs(wt[7] , 16)) ? ws[7] : wt[7];

Requirements

Header:#include<msa.h>

2.1.20.3. v4i32 __builtin_msa_max_a_w(v4i32,v4i32)

Specific

Vector Maximum of Absolute Values.

v4i32 wd = __builtin_msa_max_a_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := (abs(ws[0] , 32) > abs(wt[0] , 32)) ? ws[0] : wt[0];

wd[1] := (abs(ws[1] , 32) > abs(wt[1] , 32)) ? ws[1] : wt[1];

wd[2] := (abs(ws[2] , 32) > abs(wt[2] , 32)) ? ws[2] : wt[2];

wd[3] := (abs(ws[3] , 32) > abs(wt[3] , 32)) ? ws[3] : wt[3];

Requirements

Header:#include<msa.h>

2.1.20.4. v2i64 __builtin_msa_max_a_d(v2i64,v2i64)

Specific

Vector Maximum of Absolute Values.

v2i64 wd = __builtin_msa_max_a_d(v2i64 ws,v2i64 wt)

Return Value

wd[0] := (abs(ws[0] , 64) > abs(wt[0] , 64)) ? ws[0] : wt[0];

wd[1] := (abs(ws[1] , 64) > abs(wt[1] , 64)) ? ws[1] : wt[1];

Requirements

Header:#include<msa.h>

2.1.21. MIN_A.df

2.1.21.1. v16i8 __builtin_msa_min_a_b(v16i8,v16i8)

Specific

Vector Minimum of Absolute Values.

v16i8 wd = __builtin_msa_min_a_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := (abs(ws[0] , 8) < abs(wt[0] , 8)) ? ws[0] : wt[0];

wd[1] := (abs(ws[1] , 8) < abs(wt[1] , 8)) ? ws[1] : wt[1];

...

wd[15] := (abs(ws[15] , 8) < abs(wt[15] , 8)) ? ws[15] : wt[15];

Requirements

Header:#include<msa.h>

2.1.21.2. v8i16 __builtin_msa_min_a_h(v8i16,v8i16)

Specific

Vector Minimum of Absolute Values.

v8i16 wd = __builtin_msa_min_a_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := (abs(ws[0] , 16) < abs(wt[0] , 16)) ? ws[0] : wt[0];

wd[1] := (abs(ws[1] , 16) < abs(wt[1] , 16)) ? ws[1] : wt[1];

...

wd[7] := (abs(ws[7] , 16) < abs(wt[7] , 16)) ? ws[7] : wt[7];

Requirements

Header:#include<msa.h>

2.1.21.3. v4i32 __builtin_msa_min_a_w(v4i32,v4i32)

Specific

Vector Minimum of Absolute Values.

v4i32 wd = __builtin_msa_min_a_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := (abs(ws[0] , 32) < abs(wt[0] , 32)) ? ws[0] : wt[0];

wd[1] := (abs(ws[1] , 32) < abs(wt[1] , 32)) ? ws[1] : wt[1];

wd[2] := (abs(ws[2] , 32) < abs(wt[2] , 32)) ? ws[2] : wt[2];

wd[3] := (abs(ws[3] , 32) < abs(wt[3] , 32)) ? ws[3] : wt[3];

Requirements

Header:#include<msa.h>

2.1.21.4. v2i64 __builtin_msa_min_a_d(v2i64,v2i64)

Specific

Vector Minimum of Absolute Values.

v2i64 wd = __builtin_msa_min_a_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := (abs(ws[0] , 64) < abs(wt[0] , 64)) ? ws[0] : wt[0];

wd[1] := (abs(ws[1] , 64) < abs(wt[1] , 64)) ? ws[1] : wt[1];

Requirements

Header:#include<msa.h>

2.1.22. MAX_S.df

2.1.22.1. v16i8 __builtin_msa_max_s_b(v16i8,v16i8)

Specific

Vector Signed Maximum.

v16i8 wd = __builtin_msa_max_s_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := (ws[0] > wt[0]) ? ws[0] : wt[0];

wd[1] := (ws[1] > wt[1]) ? ws[1] : wt[1];

...

wd[15] := (ws[15] > wt[15]) ? ws[15] : wt[15];

Requirements

Header:#include<msa.h>

2.1.22.2. v8i16 __builtin_msa_max_s_h(v8i16,v8i16)

Specific

Vector Signed Maximum.

v8i16 wd = __builtin_msa_max_s_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := (ws[0] > wt[0]) ? ws[0] : wt[0];

wd[1] := (ws[1] > wt[1]) ? ws[1] : wt[1];

...

wd[7] := (ws[7] > wt[7]) ? ws[7] : wt[7];

Requirements

Header:#include<msa.h>

2.1.22.3. v4i32 __builtin_msa_max_s_w(v4i32,v4i32)

Specific

Vector Signed Maximum.

v4i32 wd = __builtin_msa_max_s_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := (ws[0] > wt[0]) ? ws[0] : wt[0];

wd[1] := (ws[1] > wt[1]) ? ws[1] : wt[1];

wd[2] := (ws[2] > wt[2]) ? ws[2] : wt[2];

wd[3] := (ws[3] > wt[3]) ? ws[3] : wt[3];

Requirements

Header:#include<msa.h>

2.1.22.4. v2i64 __builtin_msa_max_s_d(v2i64,v2i64)

Specific

Vector Signed Maximum.

v2i64 wd = __builtin_msa_max_s_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := (ws[0] > wt[0]) ? ws[0] : wt[0];

wd[1] := (ws[1] > wt[1]) ? ws[1] : wt[1];

Requirements

Header:#include<msa.h>

2.1.23. MAXI_S.df

2.1.23.1. v16i8 __builtin_msa_maxi_s_b(v16i8,imm_n16_15)

Specific

Immediate Signed Maximum.

v16i8 wd = __builtin_msa_maxi_s_b(v16i8 ws,imm_n16_15 s5)

Return Value

wd[0] := (ws[0] > s5) ? ws[0] : s5;

wd[1] := (ws[1] > s5) ? ws[1] : s5;

...

wd[15] := (ws[15] > s5) ? ws[15] : s5;

Requirements

Header:#include<msa.h>

2.1.23.2. v8i16

__builtin_msa_maxi_s_h(v8i16,imm_n16_15)

Specific

Immediate Signed Maximum.

v8i16 wd = __builtin_msa_maxi_s_h(v8i16 ws,imm_n16_15 s5);

Return Value

wd[0] := (ws[0] > s5) ? ws[0] : s5;

wd[1] := (ws[1] > s5) ? ws[1] : s5;

...

wd[7] := (ws[7] > s5) ? ws[7] : s5;

Requirements

Header:#include<msa.h>

2.1.23.3. v4i32

__builtin_msa_maxi_s_w(v4i32,imm_n16_15)

Specific

Immediate Signed Maximum.

v4i32 wd = __builtin_msa_maxi_s_w(v4i32 ws,imm_n16_15 s5);

Return Value

wd[0] := (ws[0] > s5) ? ws[0] : s5;

wd[1] := (ws[1] > s5) ? ws[1] : s5;

wd[2] := (ws[2] > s5) ? ws[2] : s5;

wd[3] := (ws[3] > s5) ? ws[3] : s5;

Requirements

Header:#include<msa.h>

2.1.23.4. v2i64 __builtin_msa_maxi_s_d(v2i64,imm_n16_15)

Specific

Immediate Signed Maximum.

v2i64 wd = __builtin_msa_maxi_s_d(v2i64 ws,imm_n16_15 s5)

Return Value

wd[0] := (ws[0] > s5) ? ws[0] : s5;

wd[1] := (ws[1] > s5) ? ws[1] : s5;

Requirements

Header:#include<msa.h>

2.1.24. MAX_U.df

2.1.24.1. v16u8 __builtin_msa_max_u_b(v16u8,v16u8)

Specific

Vector Unsigned Maximum.

v16u8 wd = __builtin_msa_max_u_b(v16u8 ws,v16u8 wt);

Return Value

wd[0] := (ws[0] > wt[0]) ? ws[0] : wt[0];

wd[1] := (ws[1] > wt[1]) ? ws[1] : wt[1];

...

wd[15] := (ws[15] > wt[15]) ? ws[15] : wt[15];

Requirements

Header:#include<msa.h>

2.1.24.2. v8u16 __builtin_msa_max_u_h(v8u16,v8u16)

Specific

Vector Unsigned Maximum.

v8u16 wd = __builtin_msa_max_u_h(v8u16 ws,v8u16 wt);

Return Value

wd[0] := (ws[0] > wt[0]) ? ws[0] : wt[0];

wd[1] := (ws[1] > wt[1]) ? ws[1] : wt[1];

...

wd[7] := (ws[7] > wt[7]) ? ws[7] : wt[7];

Requirements

Header:#include<msa.h>

2.1.24.3. v4u32 __builtin_msa_max_u_w(v4u32,v4u32)

Specific

Vector Unsigned Maximum.

v4u32 wd = __builtin_msa_max_u_w(v4u32 ws,v4u32 wt);

Return Value

wd[0] := (ws[0] > wt[0]) ? ws[0] : wt[0];

wd[1] := (ws[1] > wt[1]) ? ws[1] : wt[1];

wd[2] := (ws[2] > wt[2]) ? ws[2] : wt[2];

wd[3] := (ws[3] > wt[3]) ? ws[3] : wt[3];

Requirements

Header:#include<msa.h>

2.1.24.4. v2u64 __builtin_msa_max_u_d(v2u64,v2u64)

Specific

Vector Unsigned Maximum.

v2u64 wd = __builtin_msa_max_u_d(v2u64 ws,v2u64 wt);

Return Value

wd[0] := (ws[0] > wt[0]) ? ws[0] : wt[0];

wd[1] := (ws[1] > wt[1]) ? ws[1] : wt[1];

Requirements

Header:#include<msa.h>

2.1.25. MAXI_U.df

2.1.25.1. v16u8 __builtin_msa_maxi_u_b(v16i8,imm0_31)

Specific

Immediate Unsigned Maximum.

v16u8 wd = __builtin_msa_maxi_u_b(v16i8 ws,imm0_31 u5);

Return Value

wd[0] := (ws[0] > u5) ? ws[0] : u5;

wd[1] := (ws[1] > u5) ? ws[1] : u5;

...

wd[15] := (ws[15] > u5) ? ws[15] : u5;

Requirements

Header:#include<msa.h>

2.1.25.2. v8u16 __builtin_msa_maxi_u_h(v8i16,imm0_31)

Specific

Immediate Unsigned Maximum.

v8u16 wd = __builtin_msa_maxi_u_h(v8i16 ws,imm0_31 u5);

Return Value

wd[0] := (ws[0] > u5) ? ws[0] : u5;

wd[1] := (ws[1] > u5) ? ws[1] : u5;

...

wd[7] := (ws[7] > u5) ? ws[7] : u5;

Requirements

Header:#include<msa.h>

2.1.25.3. v4u32 __builtin_msa_maxi_u_w(v4i32,imm0_31)

Specific

Immediate Unsigned Maximum.

v4u32 wd = __builtin_msa_maxi_u_w(v4i32 ws,imm0_31 u5);

Return Value

wd[0] := (ws[0] > u5) ? ws[0] : u5;

wd[1] := (ws[1] > u5) ? ws[1] : u5;

wd[2] := (ws[2] > u5) ? ws[2] : u5;

wd[3] := (ws[3] > u5) ? ws[3] : u5;

Requirements

Header:#include<msa.h>

2.1.25.4. v2u64 __builtin_msa_maxi_u_d(v2i64,imm0_31)

Specific

Immediate Unsigned Maximum.

v2u64 wd = __builtin_msa_maxi_u_d(v2i64 ws,imm0_31 u5);

Return Value

wd[0] := (ws[0] > u5) ? ws[0] : u5;

wd[1] := (ws[1] > u5) ? ws[1] : u5;

Requirements

Header:#include<msa.h>

2.1.26. MIN_S.df

2.1.26.1. v16i8 __builtin_msa_min_s_b(v16i8,v16i8)

Specific

Vector Signed Minimum.

v16i8 wd = __builtin_msa_min_s_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := (ws[0] < wt[0]) ? ws[0] : wt[0];

wd[1] := (ws[1] < wt[1]) ? ws[1] : wt[1];

...

wd[15] :=(ws[15] < wt[15]) ? ws[15] : wt[15];

Requirements

Header:#include<msa.h>

2.1.26.2. v8i16 __builtin_msa_min_s_h(v8i16,v8i16)

Specific

Vector Signed Minimum.

v8i16 wd[0] = __builtin_msa_min_s_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := (ws[0] < wt[0]) ? ws[0] : wt[0];

wd[1] := (ws[1] < wt[1]) ? ws[1] : wt[1];

...

wd[7] :=(ws[7] < wt[7]) ? ws[7] : wt[7];

Requirements

Header:#include<msa.h>

2.1.26.3. v4i32 __builtin_msa_min_s_w(v4i32,v4i32)

Specific

Vector Signed Minimum.

v4i32 wd = __builtin_msa_min_s_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := (ws[0] < wt[0]) ? ws[0] : wt[0];

wd[1] := (ws[1] < wt[1]) ? ws[1] : wt[1];

wd[2] := (ws[2] < wt[2]) ? ws[2] : wt[2];

wd[3] := (ws[3] < wt[3]) ? ws[3] : wt[3];

Requirements

Header:#include<msa.h>

2.1.26.4. v2i64 __builtin_msa_min_s_d(v2i64,v2i64)

Specific

Vector Signed Minimum.

v2i64 wd = __builtin_msa_min_s_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := (ws[0] < wt[0]) ? ws[0] : wt[0];

wd[1] := (ws[1] < wt[1]) ? ws[1] : wt[1];

Requirements

Header:#include<msa.h>

2.1.27. MINI_S.df

2.1.27.1. v16i8 __builtin_msa_mini_s_b(v16i8,imm_n16_15)

Specific

Immediate Signed Minimum.

v16i8 wd = __builtin_msa_mini_s_b(v16i8 ws,imm_n16_15 s5);

Return Value

wd[0] := (ws[0] < s5) ? ws[0] : s5;

wd[1] := (ws[1] < s5) ? ws[1] : s5;

...

wd[15] :=(ws[15] < s5) ? ws[15] : s5;

Requirements

Header:#include<msa.h>

2.1.27.2. v8i16 __builtin_msa_mini_s_h(v8i16,imm_n16_15)

Specific

Immediate Signed Minimum.

v8i16 wd = __builtin_msa_mini_s_h(v8i16 ws,imm_n16_15 s5);

Return Value

wd[0] := (ws[0] < s5) ? ws[0] : s5;

wd[1] := (ws[1] < s5) ? ws[1] : s5;

...

wd[7] :=(ws[7] < s5) ? ws[7] : s5;

Requirements

Header:#include<msa.h>

2.1.27.3. v4i32 __builtin_msa_mini_s_w(v4i32,imm_n16_15)

Specific

Immediate Signed Minimum.

v4i32 wd = __builtin_msa_mini_s_w(v4i32 ws,imm_n16_15 s5);

Return Value

wd[0] := (ws[0] < s5) ? ws[0] : s5;

wd[1] := (ws[1] < s5) ? ws[1] : s5;

wd[2] := (ws[2] < s5) ? ws[2] : s5;

wd[3] := (ws[3] < s5) ? ws[3] : s5;

Requirements

Header:#include<msa.h>

2.1.27.4. v2i64 __builtin_msa_mini_s_d(v2i64,imm_n16_15)

Specific

Immediate Signed Minimum.

v2i64 wd = __builtin_msa_mini_s_d(v2i64 ws,imm_n16_15 s5);

Return Value

wd[0] := (ws[0] < s5) ? ws[0] : s5;

wd[1] := (ws[1] < s5) ? ws[1] : s5;

Requirements

Header:#include<msa.h>

2.1.28. MIN_U.df

2.1.28.1. v16u8 __builtin_msa_min_u_b(v16u8,v16u8)

Specific

Vector Unsigned Minimum.

v16u8 wd = __builtin_msa_min_u_b(v16u8 ws,v16u8 wt);

Return Value

wd[0] := (ws[0] < wt[0]) ? ws[0] : wt[0];

wd[1] := (ws[1] < wt[1]) ? ws[1] : wt[1];

...

wd[15] := (ws[15] < wt[15]) ? ws[15] : wt[15];

Requirements

Header:#include<msa.h>

2.1.28.2. v8u16 __builtin_msa_min_u_h(v8u16,v8u16)

Specific

Vector Unsigned Minimum.

v8u16 wd = __builtin_msa_min_u_h(v8u16 ws,v8u16 wt);

Return Value

wd[0] := (ws[0] < wt[0]) ? ws[0] : wt[0];

wd[1] := (ws[1] < wt[1]) ? ws[1] : wt[1];

...

wd[7] := (ws[7] < wt[7]) ? ws[7] : wt[7];

Requirements

Header:#include<msa.h>

2.1.28.3. v4u32 __builtin_msa_min_u_w(v4u32,v4u32)

Specific

Vector Unsigned Minimum.

v4u32 wd = __builtin_msa_min_u_w(v4u32 ws,v4u32 wt);

Return Value

wd[0] := (ws[0] < wt[0]) ? ws[0] : wt[0];

wd[1] := (ws[1] < wt[1]) ? ws[1] : wt[1];

wd[2] := (ws[2] < wt[2]) ? ws[2] : wt[2];

wd[3] := (ws[3] < wt[3]) ? ws[3] : wt[3];

Requirements

Header:#include<msa.h>

2.1.28.4. v2u64 __builtin_msa_min_u_d(v2u64,v2u64)

Specific

Vector Unsigned Minimum.

v2u64 wd = __builtin_msa_min_u_d(v2u64 ws,v2u64 wt);

Return Value

wd[0] := (ws[0] < wt[0]) ? ws[0] : wt[0];

wd[1] := (ws[1] < wt[1]) ? ws[1] : wt[1];

Requirements

Header:#include<msa.h>

2.1.29. MINI_U.df

2.1.29.1. v16u8 __builtin_msa_mini_u_b(v16i8,imm0_31)

Specific

Immediate Unsigned Minimum.

v16u8 wd = __builtin_msa_mini_u_b(v16i8 ws,imm0_31 u5);

Return Value

wd[0] := (ws[0] < u5) ? ws[0] : u5;

wd[1] := (ws[1] < u5) ? ws[1] : u5;

...

wd[15] :=(ws[15] < u5) ? ws[15] : u5;

Requirements

Header:#include<msa.h>

2.1.29.2. v8u16 __builtin_msa_mini_u_h(v8i16,imm0_31)

Specific

Immediate Unsigned Minimum.

v8u16 wd = __builtin_msa_mini_u_h(v8i16 ws,imm0_31 u5);

Return Value

wd[0] := (ws[0] < u5) ? ws[0] : u5;

wd[1] := (ws[1] < u5) ? ws[1] : u5;

...

wd[7] :=(ws[7] < u5) ? ws[7] : u5;

Requirements

Header:#include<msa.h>

2.1.29.3. v4u32 __builtin_msa_mini_u_w(v4i32,imm0_31)

Specific

Immediate Unsigned Minimum.

v4u32 wd = __builtin_msa_mini_u_w(v4i32 ws,imm0_31 u5);

Return Value

wd[0] := (ws[0] < u5) ? ws[0] : u5;

wd[1] := (ws[1] < u5) ? ws[1] : u5;

wd[2] := (ws[2] < u5) ? ws[2] : u5;

wd[3] := (ws[3] < u5) ? ws[3] : u5;

Requirements

Header:#include<msa.h>

2.1.29.4. v2u64 __builtin_msa_mini_u_d(v2i64,imm0_31)

Specific

Immediate Unsigned Minimum.

v2u64 wd = __builtin_msa_mini_u_d(v2i64 ws,imm0_31 u5);

Return Value

wd[0] := (ws[0] < u5) ? ws[0] : u5;

wd[1] := (ws[1] < u5) ? ws[1] : u5;

Requirements

Header:#include<msa.h>

2.1.30. MSUBV.df

2.1.30.1. v16i8 __builtin_msa_msubv_b(v16i8,v16i8,v16i8)

Specific

Vector Multiply-Subtract.

v16i8 wd = __builtin_msa_msubv_b(v16i8 wd ,v16i8 ws,v16i8 wt);

Return Value

wd[0] := wd[0] - ws[0] * wt[0];

wd[1] := wd[1] - ws[1] * wt[1];

...

wd[15] := wd[15] - ws[15] * wt[15];

Requirements

Header:#include<msa.h>

2.1.30.2. v8i16 __builtin_msa_msubv_h(v8i16,v8i16,v8i16)

Specific

Vector Multiply-Subtract.

v8i16 wd = __builtin_msa_msubv_h(v8i16 wd ,v8i16 ws,v8i16 wt);

Return Value

wd[0] := wd[0] - ws[0] * wt[0];

wd[1] := wd[1] - ws[1] * wt[1];

...

wd[7] := wd[7] - ws[7] * wt[7];

Requirements

Header:#include<msa.h>

2.1.30.3. v4i32 __builtin_msa_msubv_w(v4i32,v4i32,v4i32)

Specific

Vector Multiply-Subtract.

v4i32 wd = __builtin_msa_msubv_w(v4i32 wd,v4i32 ws,v4i32 wt);

Return Value

wd[0] := wd[0] - ws[0] * wt[0];

wd[1] := wd[1] - ws[1] * wt[1];

wd[2] := wd[2] - ws[2] * wt[2];

wd[3] := wd[3] - ws[3] * wt[3];

Requirements

Header:#include<msa.h>

2.1.30.4. v2i64 __builtin_msa_msubv_d(v2i64,v2i64,v2i64)

Specific

Vector Multiply-Subtract.

v2i64 wd = __builtin_msa_msubv_d(v2i64 wd,v2i64 ws,v2i64 wt);

Return Value

wd[0] := wd[0] - ws[0] * wt[0];

wd[1] := wd[1] - ws[1] * wt[1];

Requirements

Header:#include<msa.h>

2.1.31. MULV.df

2.1.31.1. v16i8 __builtin_msa_msubv_b(v16i8,v16i8)

Specific

Vector Multiply.

v16i8 wd = __builtin_msa_msubv_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := ws[0] * wt[0];

wd[1] := ws[1] * wt[1];

.....

wd[15] := ws[15] * wt[15];

Requirements

Header:#include<msa.h>

2.1.31.2. v8i16 __builtin_msa_msubv_h(v8i16,v8i16)

Specific

Vector Multiply.

v8i16 wd = __builtin_msa_msubv_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := ws[0] * wt[0];

wd[1] := ws[1] * wt[1];

.....

wd[7] := ws[7] * wt[7];

Requirements

Header:#include<msa.h>

2.1.31.3. v4i32 __builtin_msa_msubv_w(v4i32,v4i32)

Specific

Vector Multiply.

v4i32 wd = __builtin_msa_msubv_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := ws[0] * wt[0];

wd[1] := ws[1] * wt[1];

wd[2] := ws[2] * wt[2];

wd[3] := ws[3] * wt[3];

Requirements

Header:#include<msa.h>

2.1.31.4. v2i64 __builtin_msa_msubv_d(v2i64,v2i64)

Specific

Vector Multiply.

v2i64 wd = __builtin_msa_msubv_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := ws[0] * wt[0];

wd[1] := ws[1] * wt[1];

Requirements

Header:#include<msa.h>

2.1.32. MOD_S.df

2.1.32.1. v16i8 __builtin_msa_mod_s_b(v16i8,v16i8)

Specific

Vector Signed Remainder.

v16i8 wd = __builtin_msa_mod_s_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := ws[0] % wt[0];

wd[1] := ws[1] % wt[1];

.....

wd[15] := ws[15] % wt[15];

Requirements

Header:#include<msa.h>

2.1.32.2. v8i16 __builtin_msa_mod_s_h(v8i16,v8i16)

Specific

Vector Signed Remainder.

v8i16 wd = __builtin_msa_mod_s_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := ws[0] % wt[0];

wd[1] := ws[1] % wt[1];

.....

wd[7] := ws[7] % wt[7];

Requirements

Header:#include<msa.h>

2.1.32.3. v4i32 __builtin_msa_mod_s_w(v4i32,v4i32)

Specific

Vector Signed Remainder.

v4i32 wd = __builtin_msa_mod_s_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := ws[0] % wt[0];

wd[1] := ws[1] % wt[1];

wd[2] := ws[2] % wt[2];

wd[3] := ws[3] % wt[3];

Requirements

Header:#include<msa.h>

2.1.32.4. v2i64 __builtin_msa_mod_s_d(v2i64,v2i64)

Specific

Vector Signed Remainder.

v2i64 wd = __builtin_msa_mod_s_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := ws[0] % wt[0];

wd[1] := ws[1] % wt[1];

Requirements

Header:#include<msa.h>

2.1.33. MOD_U.df

2.1.33.1. v16u8 __builtin_msa_mod_u_b(v16u8,v16u8)

Specific

Vector Unsigned Remainder.

v16u8 wd = __builtin_msa_mod_u_b(v16u8 ws,v16u8 wt);

Return Value

wd[0] := ws[0] % wt[0];

wd[1] := ws[1] % wt[1];

.....

wd[15] := ws[15] % wt[15];

Requirements

Header:#include<msa.h>

2.1.33.2. v8u16 __builtin_msa_mod_u_h(v8u16,v8u16)

Specific

Vector Unsigned Remainder.

v8u16 wd = __builtin_msa_mod_u_h(v8u16 ws,v8u16 wt);

Return Value

wd[0] := ws[0] % wt[0];

wd[1] := ws[1] % wt[1];

.....

wd[7] := ws[7] % wt[7];

Requirements

Header:#include<msa.h>

2.1.33.3. v4u32 __builtin_msa_mod_u_w(v4u32,v4u32)

Specific

Vector Unsigned Remainder.

v4u32 wd = __builtin_msa_mod_u_w(v4u32 ws,v4u32 wt);

Return Value

wd[0] := ws[0] % wt[0];

wd[1] := ws[1] % wt[1];

wd[2] := ws[2] % wt[2];

wd[3] := ws[3] % wt[3];

Requirements

Header:#include<msa.h>

2.1.33.4. v2u64 __builtin_msa_mod_u_d(v2u64,v2u64)

Specific

Vector Unsigned Remainder.

v2u64 wd = __builtin_msa_mod_u_d(v2u64 ws,v2u64 wt);

Return Value

wd[0] := ws[0] % wt[0];

wd[1] := ws[1] % wt[1];

Requirements

Header:#include<msa.h>

2.1.34. SAT_S.df

2.1.34.1. v16i8 __builtin_msa_sat_s_b(v16i8,imm0_7)

Specific

Vector Signed Saturate.

v16i8 wd = __builtin_msa_sat_s_b(v16i8 ws,imm0_7 m);

Return Value

wd[0] := sat_s(ws[0],m,8);

wd[1] := sat_s(ws[1],m,8);

.....

wd[15] := sat_s(ws[15],m,8);

Requirements

Header:#include<msa.h>

2.1.34.2. v8i16 __builtin_msa_sat_s_h(v8i16,imm0_7)

Specific

Vector Signed Saturate.

v8i16 wd = __builtin_msa_sat_s_h(v8i16 ws,imm0_7 m);

Return Value

wd[0] := sat_s(ws[0],m,16);

wd[1] := sat_s(ws[1],m,16);

.....

wd[7] := sat_s(ws[7],m,16);

Requirements

Header:#include<msa.h>

2.1.34.3. v4i32 __builtin_msa_sat_s_w(v4i32,imm0_7)

Specific

Vector Signed Saturate.

v4i32 wd = __builtin_msa_sat_s_w(v4i32 ws,imm0_7 m);

Return Value

wd[0] := sat_s(ws[0],m,32);

wd[1] := sat_s(ws[1],m,32);

wd[2] := sat_s(ws[2],m,32);

wd[3] := sat_s(ws[3],m,32);

Requirements

Header:#include<msa.h>

2.1.34.4. v2i64 __builtin_msa_sat_s_d(v2i64,imm0_7)

Specific

Vector Signed Saturate.

v2i64 wd = __builtin_msa_sat_s_d(v2i64 ws,imm0_7 m);

Return Value

wd[0] := sat_s(ws[0],m,64);

wd[1] := sat_s(ws[1],m,64);

Requirements

Header:#include<msa.h>

2.1.35. SAT_U.df

2.1.35.1. v16u8 __builtin_msa_sat_u_b(v16u8,imm0_7)

Specific

Vector Unsigned Saturate.

v16u8 wd = __builtin_msa_sat_u_b(v16u8 ws,imm0_7 m);

Return Value

wd[0] := sat_u(ws[0],m,8);

wd[1] := sat_u(ws[1],m,8);

.....

wd[15] := sat_u(ws[15],m,8);

Requirements

Header:#include<msa.h>

2.1.35.2. v8u16 __builtin_msa_sat_u_h(v8u16,imm0_7)

Specific

Vector Unsigned Saturate.

v8u16 wd = __builtin_msa_sat_u_h(v8u16 ws,imm0_7 m);

Return Value

wd[0] := sat_u(ws[0],m,16);

wd[1] := sat_u(ws[1],m,16);

.....

wd[7] := sat_u(ws[7],m,16);

Requirements

Header:#include<msa.h>

2.1.35.3. v4u32 __builtin_msa_sat_u_w(v4u32,imm0_7)

Specific

Vector Unsigned Saturate.

v4u32 wd = __builtin_msa_sat_u_w(v4u32 ws,imm0_7 m);

Return Value

wd[0] := sat_u(ws[0],m,32);

wd[1] := sat_u(ws[1],m,32);

wd[2] := sat_u(ws[2],m,32);

wd[3] := sat_u(ws[3],m,32);

Requirements

Header:#include<msa.h>

2.1.35.4. v2u64 __builtin_msa_sat_u_d(v2u64,imm0_7)

Specific

Vector Unsigned Saturate.

v2u64 wd = __builtin_msa_sat_u_d(v2u64 ws,imm0_7 m);

Return Value

wd[0] := sat_u(ws[0],m,64);

wd[1] := sat_u(ws[1],m,64);

Requirements

Header:#include<msa.h>

2.1.36. SUBS_S.df

2.1.36.1. v16i8 __builtin_msa_subs_s_b(v16i8,v16i8)

Specific

Vector Signed Saturated Signed Subtract.

v16i8 wd = __builtin_msa_subs_s_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := sats(ws[0] - wt[0] , 8);

wd[1] := sats(ws[1] - wt[1] , 8);

.....

wd[15] := sats(ws[15] - wt[15] , 8);

Requirements

Header:#include<msa.h>

2.1.36.2. v8i16 __builtin_msa_subs_s_h(v8i16,v8i16)

Specific

Vector Signed Saturated Signed Subtract.

v8i16 wd = __builtin_msa_subs_s_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := sats(ws[0] - wt[0] , 16);

wd[1] := sats(ws[1] - wt[1] , 16);

.....

wd[7] := sats(ws[7] - wt[7] , 16);

Requirements

Header:#include<msa.h>

2.1.36.3. v4i32 __builtin_msa_subs_s_w(v4i32,v4i32)

Specific

Vector Signed Saturated Signed Subtract.

v4i32 wd = __builtin_msa_subs_s_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := sats(ws[0] - wt[0] , 32);

wd[1] := sats(ws[1] - wt[1] , 32);

wd[2] := sats(ws[2] - wt[2] , 32);

wd[3] := sats(ws[3] - wt[3] , 32);

Requirements

Header:#include<msa.h>

2.1.36.4. v2i64 __builtin_msa_subs_s_d(v2i64,v2i64)

Specific

Vector Signed Saturated Signed Subtract.

v2i64 wd = __builtin_msa_subs_s_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := sats(ws[0] - wt[0] , 64);

wd[1] := sats(ws[1] - wt[1] , 64);

Requirements

Header:#include<msa.h>

2.1.37. SUBS_U.df

2.1.37.1. v16u8 __builtin_msa_subs _u_b(v16u8,v16u8)

Specific

Vector Unsigned Saturate Unsigned Subtract.

v16u8 wd = __builtin_msa_subs _u_b(v16u8 ws,v16u8 wt);

Return Value

wd[0] := satu(ws[0] - wt[0]);

wd[1] := satu(ws[1] - wt[1]);

.....

wd[15] := satu(ws[15] - wt[15]);

Requirements

Header:#include<msa.h>

2.1.37.2. v8u16 __builtin_msa_subs_u_h(v8u16,v8u16)

Specific

Vector Unsigned Saturate Unsigned Subtract.

v8u16 wd = __builtin_msa_subs_u_h(v8u16 ws,v8u16 wt);

Return Value

wd[0] := satu(ws[0] - wt[0]);

wd[1] := satu(ws[1] - wt[1]);

.....

wd[7] := satu(ws[7] - wt[7]);

Requirements

Header:#include<msa.h>

2.1.37.3. v4u32 __builtin_msa_subs_u_w(v4u32,v4u32)

Specific

Vector Unsigned Saturate Unsigned Subtract.

v4u32 wd = __builtin_msa_subs_u_w(v4u32 ws,v4u32 wt);

Return Value

wd[0] := satu(ws[0] - wt[0]);

wd[1] := satu(ws[1] - wt[1]);

wd[2] := satu(ws[2] - wt[2]);

wd[3] := satu(ws[3] - wt[3]);

Requirements

Header:#include<msa.h>

2.1.37.4. v2u64 __builtin_msa_subs_u_d(v2u64,v2u64)

Specific

Vector Unsigned Saturate Unsigned Subtract.

v2u64 wd = __builtin_msa_subs_u_d(v2u64 ws,v2u64 wt);

Return Value

wd[0] := satu(ws[0] - wt[0]);

wd[1] := satu(ws[1] - wt[1]);

Requirements

Header:#include<msa.h>

2.1.38. HSUB_S.df

2.1.38.1. v8i16 __builtin_msa_hsub_s_h(v16i8,v16i8)

Specific

Vector Signed Horizontal Subtract.

v8i16 wd = __builtin_msa_hsub_s_h(v16i8 ws,v16i8 wt);

Return Value

wd[0] := ws[1] - wt[0];

wd[1] := ws[3] - wt[2];

.....

wd[7] := ws[15] - wt[14];

Requirements

Header:#include<msa.h>

eg:

signed char a[16] = {12,29,3,4,

```
                    5,63,64,65,
                    -7,12,9,25,
                    52,93,19,91};
signed char b[16] ={1,2,3,4,
                    5,6,7,8,
                    9,10,11,12,
                    13,14,15,16};
short c[8];
v16i8 va,vb;
v8i16 vc;
va = __builtin_msa_ld_b(a,0);
vb = __builtin_msa_ld_b(b,0);
vc = __builtin_msa_hsub_s_h(va,vb);
__builtin_msa_st_h(vc,c,0);
result:
c[8]={28,1,58,58,3,14,80,76};
```

2.1.38.2. v4i32 __builtin_msa_hsub_s_w(v8i16,v8i16)

Specific

Vector Signed Horizontal Subtract.

```
v4i32 wd = __builtin_msa_hsub_s_w(v8i16 ws,v8i16 wt);
```

Return Value

wd[0] := ws[1] - wt[0];

wd[1] := ws[3] - wt[2];

wd[2] := ws[5] - wt[4];

wd[3] := ws[7] - wt[6];

Requirements

Header:#include<msa.h>

2.1.38.3. v2i64 __builtin_msa_hsub_s_d(v4i32,v4i32)

Specific

Vector Signed Horizontal Subtract.

v2i64 wd = __builtin_msa_hsub_s_d(v4i32 ws,v4i32 wt);

Return Value

wd[0] := ws[1] - wt[0];

wd[1] := ws[3] - wt[2];

Requirements

Header:#include<msa.h>

2.1.39. HSUB_U.df

2.1.39.1. v8i16 __builtin_msa_hsub_u_h(v16u8,v16u8)

Specific

Vector Unsigned Horizontal Subtract.

v8i16 wd = __builtin_msa_hsub_u_h(v16u8 ws,v16u8 wt);

Return Value

wd[0] := ws[1] - wt[0];

wd[1] := ws[3] - wt[2];

.....

wd[7] := ws[15] - wt[14];

Requirements

Header:#include<msa.h>

2.1.39.2. v4i32 __builtin_msa_hsub_u_w(v8u16,v8u16)

Specific

Vector Unsigned Horizontal Subtract.

v4i32 wd = __builtin_msa_hsub_u_w(v8u16 ws,v8u16 wt);

Return Value

wd[0] := ws[1] - wt[0];

wd[1] := ws[3] - wt[2];

wd[2] := ws[5] - wt[4];

wd[3] := ws[7] - wt[6];

Requirements

Header:#include<msa.h>

2.1.39.3. v2i64 __builtin_msa_hsub_u_d(v4u32,v4u32)

Specific

Vector Unsigned Horizontal Subtract.

v2i64 wd = __builtin_msa_hsub_u_d(v4u32 ws,v4u32 wt);

Return Value

wd[0] := ws[1] - wt[0];

wd[1] := ws[3] - wt[2];

Requirements

Header:#include<msa.h>

2.1.40. SUBSUU_S.df

2.1.40.1. v16i8 __builtin_msa_subsuu _s_b(v16u8,v16u8)

Specific

Vector Signed Saturate Unsigned Subtract.

v16i8 wd = __builtin_msa_subsuu _s_b(v16u8 ws,v16u8 wt);

Return Value

wd[0] := sats(ws[0] - wt[0] , 8);

wd[1] := sats(ws[1] - wt[1] , 8);

.....

wd[15] := sats(ws[15] - wt[15] , 8);

Requirements

Header:#include<msa.h>

2.1.40.2. v8i16 __builtin_msa_subsuu_s_h(v8u16,v8u16)

Specific

Vector Signed Saturate Unsigned Subtract.

v8i16 wd = __builtin_msa_subsuu_s_h(v8u16 ws,v8u16 wt);

Return Value

wd[0] := sats(ws[0] - wt[0] , 16);

wd[1] := sats(ws[1] - wt[1] , 16);

.....

wd[7] := sats(ws[7] - wt[7] , 16);

Requirements

Header:#include<msa.h>

2.1.40.3. v4i32 __builtin_msa_subsuu_s_w(v4u32,v4u32)

Specific

Vector Signed Saturate Unsigned Subtract.

v4i32 wd = __builtin_msa_subsuu_s_w(v4u32 ws,v4u32 wt);

Return Value

wd[0] := sats(ws[0] - wt[0] , 32);

wd[1] := sats(ws[1] - wt[1] , 32);

wd[2] := sats(ws[2] - wt[2] , 32);

wd[3] := sats(ws[3] - wt[3] , 32);

Requirements

Header:#include<msa.h>

2.1.40.4. v2i64 __builtin_msa_subsuu_s_d(v2u64,v2u64)

Specific

Vector Signed Saturate Unsigned Subtract.

v2i64 wd = __builtin_msa_subsuu_s_d(v2u64 ws,v2u64 wt);

Return Value

wd[0] := sats(ws[0] - wt[0] , 64);

wd[1] := sats(ws[1] - wt[1] , 64);

Requirements

Header:#include<msa.h>

2.1.41. SUBSUS_U.df

2.1.41.1. v16u8 __builtin_msa_subsus _u_b(v16u8,v16i8)

Specific

Vector Unsigned Saturated Subtract of Signed from Unsigned.

v16u8 wd = __builtin_msa_subsus _u_b(v16u8 ws,v16i8 wt);

Return Value

wd[0] := subsus_u(ws[0] - wt[0] , 8);

wd[1] := subsus_u(ws[1] - wt[1] , 8);

.....

wd[15] := subsus_u(ws[15] - wt[15] , 8);

Requirements

Header:#include<msa.h>

2.1.41.2. v8u16 __builtin_msa_subsus_u_h(v8u16,v8i16)

Specific

Vector Unsigned Saturated Subtract of Signed from Unsigned.

v8u16 wd = __builtin_msa_subsus_u_h(v8u16 ws,v8i16 wt);

Return Value

wd[0] := subsus_u(ws[0] - wt[0] , 16);

wd[1] := subsus_u(ws[1] - wt[1] , 16);

.....

wd[7] := subsus_u(ws[7] - wt[7] , 16);

Requirements

Header:#include<msa.h>

2.1.41.3. v4u32 __builtin_msa_subsus_u_w(v4u32,v4i32)

Specific

Vector Unsigned Saturated Subtract of Signed from Unsigned.

v4u32 wd = __builtin_msa_subsus_u_w(v4u32 ws,v4i32 wt);

Return Value

wd[0] := subsus_u(ws[0] - wt[0] , 32);

wd[1] := subsus_u(ws[1] - wt[1] , 32);

wd[2] := subsus_u(ws[2] - wt[2] , 32);

wd[3] := subsus_u(ws[3] - wt[3] , 32);

Requirements

Header:#include<msa.h>

2.1.41.4. v2u64 __builtin_msa_subsus_u_d(v2u64,v2i64)

Specific

Vector Unsigned Saturated Subtract of Signed from Unsigned.

v2u64 wd = __builtin_msa_subsus_u_d(v2u64 ws,v2i64 wt);

Return Value

wd[0] := subsus_u(ws[0] - wt[0] , 64);

wd[1] := subsus_u(ws[1] - wt[1] , 64);

Requirements

Header:#include<msa.h>

2.1.42. SUBV.df

2.1.42.1. v16i8 __builtin_msa_subv_b(v16i8,v16i8)

Specific

Vector Subtract.

v16i8 wd = __builtin_msa_subv_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := ws[0] - wt[0];

wd[1] := ws[1] - wt[1];

.....

wd[15] := ws[15] -wt[15];

Requirements

Header:#include<msa.h>

2.1.42.2. v8i16 __builtin_msa_subv_h(v8i16,v8i16)

Specific

Vector Subtract.

v8i16 wd = __builtin_msa_subv_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := ws[0] - wt[0];

wd[1] := ws[1] - wt[1];

.....

wd[7] := ws[7] -wt[7];

Requirements

Header:#include<msa.h>

2.1.42.3. v4i32 __builtin_msa_subv_w(v4i32,v4i32)

Specific

Vector Subtract.

v4i32 wd = __builtin_msa_subv_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := ws[0] - wt[0];

wd[1] := ws[1] - wt[1];

wd[2] := ws[2] - wt[2];

wd[3] := ws[3] - wt[3];

Requirements

Header:#include<msa.h>

2.1.42.4. v2i64 __builtin_msa_subv_d(v2i64,v2i64)

Specific

Vector Subtract.

v2i64 wd = __builtin_msa_subv_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := ws[0] - wt[0];

wd[1] := ws[1] - wt[1];

Requirements

Header:#include<msa.h>

2.1.43. SUBVI.df

2.1.43.1. v16i8 __builtin_msa_subvi_b(v16i8,imm0_31)

Specific

Immediate Subtract.

v16i8 wd = __builtin_msa_subvi_b(v16i8 ws,imm0_31 u5);

Return Value

wd[0] := ws[0] - u5;

wd[1] := ws[1] - u5;

.....

wd[15] := ws[15] - u5;

Requirements

Header:#include<msa.h>

2.1.43.2. v8i16 __builtin_msa_subvi_h(v8i16,imm0_31)

Specific

Immediate Subtract.

v8i16 wd = __builtin_msa_subvi_h(v8i16 ws,imm0_31 u5);

Return Value

wd[0] := ws[0] - u5;

wd[1] := ws[1] - u5;

.....

wd[7] := ws[7] - u5;

Requirements

Header:#include<msa.h>

2.1.43.3. v4i32 __builtin_msa_subvi_w(v4i32,imm0_31)

Specific

Immediate Subtract.

v4i32 wd = __builtin_msa_subvi_w(v4i32 ws,imm0_31 u5);

Return Value

wd[0] := ws[0] - u5;

wd[1] := ws[1] - u5;

wd[2] := ws[2] - u5;

wd[3] := ws[3] - u5;

Requirements

Header:#include<msa.h>

2.1.43.4. v2i64 __builtin_msa_subvi_d(v2i64,imm0_31)

Specific

Immediate Subtract.

v2i64 wd = __builtin_msa_subvi_d(v2i64 ws,imm0_31 u5);

Return Value

wd[0] := ws[0] - u5;

wd[1] := ws[1] - u5;

Requirements

Header:#include<msa.h>

2.1.44. SATS.df

2.1.44.1. v16i8 __builtin_msa_sats_b_h (v8i16,v8i16)

Specific

Fixed Signed Saturate.

v16i8 wd = __builtin_msa_sats_b_h (v8i16 ws,v8i16 wt);

Return Value

wd[0] := sats_fixed(wt[0],8);

wd[1] := sats_fixed(wt[1],8);

wd[2] := sats_fixed(wt[2],8);

wd[3] := sats_fixed(wt[3],8);

wd[4] := sats_fixed(wt[4],8);

wd[5] := sats_fixed(wt[5],8);

wd[6] := sats_fixed(wt[6],8);

wd[7] := sats_fixed(wt[7],8);

wd[8] := sats_fixed(ws[0],8);

wd[9] := sats_fixed(ws[1],8);

wd[10] := sats_fixed(ws[2],8);

wd[11] := sats_fixed(ws[3],8);

wd[12] := sats_fixed(ws[4],8);

wd[13] := sats_fixed(ws[5],8);

wd[14] := sats_fixed(ws[6],8);

wd[15] := sats_fixed(ws[7],8);

Requirements

Header:#include<msa.h>

2.1.44.2. v8i16 __builtin_msa_sats_h_w (v4i32,v4i32)

Specific

Fixed Signed Saturate.

v8i16 wd = __builtin_msa_sats_h_w (v4i32 ws,v4i32 wt);

Return Value

wd[0] := sats_fixed (wt[0],16);

wd[1] := sats_fixed (wt[1],16);

wd[2] := sats_fixed (wt[2],16);

wd[3] := sats_fixed (wt[3],16);

wd[4] := sats_fixed (ws[0],16);

wd[5] := sats_fixed (ws[1],16);

wd[6] := sats_fixed (ws[2],16);

wd[7] := sats_fixed (ws[3],16);

Requirements

Header:#include<msa.h>

2.1.45. SATU.df

2.1.45.1. v16u8 __builtin_msa_satu_b_h (v8u16,v8u16)

Specific

Fixed Unsigned Saturate.

v16u8 wd = __builtin_msa_satu_b_h (v8u16 ws,v8u16 wt);

Return Value

wd[0] := satu_fixed (wt[0] , 8);

wd[1] := satu_fixed (wt[1] , 8);

wd[2] := satu_fixed (wt[2] , 8);

wd[3] := satu_fixed (wt[3] , 8);

wd[4] := satu_fixed (wt[4] , 8);

wd[5] := satu_fixed (wt[5] , 8);

wd[6] := satu_fixed (wt[6] , 8);

wd[7] := satu_fixed (wt[7] , 8);

wd[8] := satu_fixed (ws[0] , 8);

wd[9] := satu_fixed (ws[1] , 8);

wd[10] := satu_fixed (ws[2] , 8);

wd[11] := satu_fixed (ws[3] , 8);

wd[12] := satu_fixed (ws[4] , 8);

wd[13] := satu_fixed (ws[5] , 8);

wd[14] := satu_fixed (ws[6] , 8);

wd[15] := satu_fixed (ws[7] , 8);

Requirements

Header:#include<msa.h>

2.1.45.2. v8u16 __builtin_msa_satu_h_w (v4u32,v4u32)

Specific

Fixed Unsigned Saturate.

v8u16 wd = __builtin_msa_satu_h_w (v4u32 ws,v4u32 wt);

Return Value

wd[0] := satu_fixed (wt[0] , 16);

wd[1] := satu_fixed (wt[1] , 16);

wd[2] := satu_fixed (wt[2] , 16);

wd[3] := satu_fixed (wt[3] , 16);

wd[4] := satu_fixed (ws[0] , 16);

wd[5] := satu_fixed (ws[1] , 16);

wd[6] := satu_fixed (ws[2] , 16);

wd[7] := satu_fixed (ws[3] , 16);

Requirements

Header:#include<msa.h>

2.1.46. ADDSL.df

2.1.46.1. v8i16 __builtin_msa_addsl_h (v16i8,v16i8)

Specific

Vector Add Left of signed Values.

v8i16 wd = __builtin_msa_addsl_h (v16i8 ws,v16i8 wt);

Return Value

wd[0] := (short)ws[8] + (short)wt[8];

wd[1] := (short)ws[9] + (short)wt[9];

.....

wd[7] := (short)ws[15] + (short)wt[15];

Requirements

Header:#include<msa.h>

2.1.46.2. v4i32 __builtin_msa_addsl_w (v8i16,v8i16)

Specific

Vector Add Left of signed Values.

v4i32 wd = __builtin_msa_addsl_w (v8i16 ws,v8i16 wt);

Return Value

wd[0] := (int)ws[4] + (int)wt[4];

wd[1] := (int)ws[5] + (int)wt[5];

wd[2] := (int)ws[6] + (int)wt[6];

wd[3] := (int)ws[7] + (int)wt[7];

Requirements

Header:#include<msa.h>

2.1.47. ADDSR.df

2.1.47.1. v8i16 __builtin_msa_addsr_h (v16i8,v16i8)

Specific

Vector Add Right of signed Values.

v8i16 wd = __builtin_msa_addsr_h (v16i8 ws,v16i8 wt);

Return Value

wd[0] := (short)ws[0] + (short)wt[0];

wd[1] := (short)ws[1] + (short)wt[1];

.....

wd[7] := (short)ws[7] + (short)wt[7];

Requirements

Header:#include<msa.h>

2.1.47.2. v4i32 __builtin_msa_addsr_w (v8i16,v8i16)

Specific

Vector Add Right of signed Values.

v4i32 wd = __builtin_msa_addsr_w (v8i16 ws,v8i16 wt);

Return Value

wd[0] := (int)ws[0] + (int)wt[0];

wd[1] := (int)ws[1] + (int)wt[1];

wd[2] := (int)ws[2] + (int)wt[2];

wd[3] := (int)ws[3] + (int)wt[3];

Requirements

Header:#include<msa.h>

2.1.48. ADDUL.df

2.1.48.1. v8u16 __builtin_msa_addul_h (v16u8,v16u8)

Specific

Vector Add Left of Unsigned Values.

v8u16 wd = __builtin_msa_addul_h (v16u8 ws,v16u8 wt);

Return Value

wd[0] := (unsigned short)ws[8] + (unsigned short)wt[8];

wd[1] := (unsigned short)ws[9] + (unsigned short)wt[9];

.....

wd[7] := (unsigned short)ws[15] + (unsigned short)wt[15];

Requirements

Header:#include<msa.h>

2.1.48.2. v4u32 __builtin_msa_addul_w (v8u16,v8u16)

Specific

Vector Add Left of Unsigned Values.

v4u32 wd = __builtin_msa_addul_w (v8u16 ws,v8u16 wt);

Return Value

wd[0] := (unsigned int)ws[4] + (unsigned int)wt[4];

wd[1] := (unsigned int)ws[5] + (unsigned int)wt[5];

wd[2] := (unsigned int)ws[6] + (unsigned int)wt[6];

wd[3] := (unsigned int)ws[7] + (unsigned int)wt[7];

Requirements

Header:#include<msa.h>

2.1.49. ADDUR.df

2.1.49.1. v8u16 __builtin_msa_addur_h (v16u8,v16u8)

Specific

Vector Add Right of Unsigned Values.

v8u16 wd = __builtin_msa_addur_h (v16u8 ws,v16u8 wt);

Return Value

wd[0] := (unsigned short)ws[0] + (unsigned short)wt[0];

wd[1] := (unsigned short)ws[1] + (unsigned short)wt[1];

.....

wd[7] := (unsigned short)ws[7] + (unsigned short)wt[7];

Requirements

Header:#include<msa.h>

2.1.49.2. v4u32 __builtin_msa_addur_w (v8u16,v8u16)

Specific

Vector Add Right of Unsigned Values.

v4u32 wd = __builtin_msa_addur_w (v8u16 ws,v8u16 wt);

Return Value

wd[0] := (unsigned int)ws[0] + (unsigned int)wt[0];

wd[1] := (unsigned int)ws[1] + (unsigned int)wt[1];

wd[2] := (unsigned int)ws[2] + (unsigned int)wt[2];

wd[3] := (unsigned int)ws[3] + (unsigned int)wt[3];

Requirements

Header:#include<msa.h>

2.1.50. ACCSL.df

2.1.50.1. v8i16 __builtin_msa_accsl_h (v8i16,v16i8)

Specific

Vector Accumulate Left of Signed Values.

v8i16 wd = __builtin_msa_accsl_h (v8i16 wd,v16i8 ws);

Return Value

wd[0] := wd[0] + (short)ws[8];

wd[1] := wd[1] + (short)ws[9];

.....

wd[7] := wd[7] + (short)ws[15];

Requirements

Header:#include<msa.h>

2.1.50.2. v4i32 __builtin_msa_accsl_w (v4i32,v8i16)

Specific

Vector Accumulate Left of Signed Values.

v4i32 wd = __builtin_msa_accsl_w (v4i32 wd,v8i16 ws);

Return Value

wd[0] := wd[0] + (int)ws[4];

wd[1] := wd[1] + (int)ws[5];

wd[2] := wd[2] + (int)ws[6];

wd[3] := wd[3] + (int)ws[7];

Requirements

Header:#include<msa.h>

2.1.51. ACCSR.df

2.1.51.1. v8i16 __builtin_msa_accsr_h (v8i16,v16i8)

Specific

Vector Accumulate Right of Signed Values.

v8i16 wd = __builtin_msa_accsr_h (v8i16 wd,v16i8 ws);

Return Value

wd[0] := wd[0] + (short)ws[0];

wd[1] := wd[1] + (short)ws[1];

.....

wd[7] := wd[7] + (short)ws[7];

Requirements

Header:#include<msa.h>

2.1.51.2. v4i32 __builtin_msa_accsr_w (v4i32,v8i16)

Specific

Vector Accumulate Right of Signed Values.

v4i32 wd = __builtin_msa_accsr_w (v4i32 wd,v8i16 ws);

Return Value

wd[0] := wd[0] + (int)ws[0];

wd[1] := wd[1] + (int)ws[1];

wd[2] := wd[2] + (int)ws[2];

wd[3] := wd[3] + (int)ws[3];

Requirements

Header:#include<msa.h>

2.1.52. ACCUL.df

2.1.52.1. v8u16 __builtin_msa_accul_h (v8u16,v16u8)

Specific

Vector Accumulate Left of Unsigned Values.

v8u16 wd = __builtin_msa_accul_h (v8u16 wd,v16u8 ws);

Return Value

wd[0] := wd[0] + (unsigned short)ws[8];

wd[1] := wd[1] + (unsigned short)ws[9];

.....

wd[7] := wd[7] + (unsigned short)ws[15];

Requirements

Header:#include<msa.h>

2.1.52.2. v4u32 __builtin_msa_accul_w (v4u32,v8u16)

Specific

Vector Accumulate Left of Unsigned Values.

v4u32 wd = __builtin_msa_accul_w (v4u32 wd,v8u16 ws);

Return Value

wd[0] := wd[0] + (unsigned int)ws[4];

wd[1] := wd[1] + (unsigned int)ws[5];

wd[2] := wd[2] + (unsigned int)ws[6];

wd[3] := wd[3] + (unsigned int)ws[7];

Requirements

Header:#include<msa.h>

2.1.53. ACCUR.df

2.1.53.1. v8u16 __builtin_msa_accur_h (v8u16,v16u8)

Specific

Vector Accumulate Right of Unsigned Values.

v8u16 wd = __builtin_msa_accur_h (v8u16 wd,v16u8 ws);

Return Value

wd[0] := wd[0] + (unsigned short)ws[0];

wd[1] := wd[1] + (unsigned short)ws[1];

.....

wd[7] := wd[7] + (unsigned short)ws[7];

Requirements

Header:#include<msa.h>

2.1.53.2. v4u32 __builtin_msa_accur_w (v4u32,v8u16)

Specific

Vector Accumulate Right of Unsigned Values.

v4u32 wd = __builtin_msa_accur_w (v4u32 wd,v8u16 ws);

Return Value

wd[0] := wd[0] + (unsigned int)ws[0];

wd[1] := wd[1] + (unsigned int)ws[1];

wd[2] := wd[2] + (unsigned int)ws[2];

wd[3] := wd[3] + (unsigned int)ws[3];

Requirements

Header:#include<msa.h>

2.1.54. SUBSL.df

2.1.54.1. v8i16 __builtin_msa_subsl_h (v16i8,v16i8)

Specific

Vector Subtract Left of Signed Values.

v8i16 wd = __builtin_msa_subsl_h (v16i8 ws,v16i8 wt);

Return Value

wd[0] := (short)ws[8] - (short)wt[8];

wd[1] := (short)ws[9] - (short)wt[9];

.....

wd[7] := (short)ws[15] - (short)wt[15];

Requirements

Header:#include<msa.h>

2.1.54.2. v4i32 __builtin_msa_subsl_w (v8i16,v8i16)

Specific

Vector Subtract Left of Signed Values.

v4i32 wd = __builtin_msa_subsl_w (v8i16 ws,v8i16 wt);

Return Value

wd[0] := (int)ws[4] - (int)wt[4];

wd[1] := (int)ws[5] - (int)wt[5];

wd[2] := (int)ws[6] - (int)wt[6];

wd[3] := (int)ws[7] - (int)wt[7];

Requirements

Header:#include<msa.h>

2.1.55. SUBSR.df

2.1.55.1. v8i16 __builtin_msa_subsr_h (v16i8,v16i8)

Specific

Vector Subtract Right of Signed Values.

v8i16 wd = __builtin_msa_subsr_h (v16i8 ws,v16i8 wt);

Return Value

wd[0] := (short)ws[0] - (short)wt[0];

wd[1] := (short)ws[1] - (short)wt[1];

.....

wd[7] := (short)ws[7] - (short)wt[7];

Requirements

Header:#include<msa.h>

2.1.55.2. v4i32 __builtin_msa_subsr_w (v8i16,v8i16)

Specific

Vector Subtract Right of Signed Values.

v4i32 wd = __builtin_msa_subsr_w (v8i16 ws,v8i16 wt);

Return Value

wd[0] := (int)ws[0] - (int)wt[0];

wd[1] := (int)ws[1] - (int)wt[1];

wd[2] := (int)ws[2] - (int)wt[2];

wd[3] := (int)ws[3] - (int)wt[3];

Requirements

Header:#include<msa.h>

2.1.56. SUBUL.df

2.1.56.1. v8i16 __builtin_msa_subul_h (v16u8,v16u8)

Specific

Vector Subtract Left of Unsigned Values.

v8i16 wd = __builtin_msa_subul_h (v16u8 ws,v16u8 wt);

Return Value

wd[0] := (unsigned short)ws[8] - (unsigned short)wt[8];

wd[1] := (unsigned short)ws[9] - (unsigned short)wt[9];

.....

wd[7] := (unsigned short)ws[15] - (unsigned short)wt[15];

Requirements

Header:#include<msa.h>

2.1.56.2. v4i32 __builtin_msa_subul_w (v8u16,v8u16)

Specific

Vector Subtract Left of Unsigned Values.

v4i32 wd = __builtin_msa_subul_w (v8u16 ws,v8u16 wt);

Return Value

wd[0] := (unsigned int)ws[4] - (unsigned int)wt[4];

wd[1] := (unsigned int)ws[5] - (unsigned int)wt[5];

wd[2] := (unsigned int)ws[6] - (unsigned int)wt[6];

wd[3] := (unsigned int)ws[7] - (unsigned int)wt[7];

Requirements

Header:#include<msa.h>

2.1.57. SUBUR.df

2.1.57.1. v8i16 __builtin_msa_subur_h (v16u8,v16u8)

Specific

Vector Subtract Right of Unsigned Values.

v8i16 wd = __builtin_msa_subur_h (v16u8 ws,v16u8 wt);

Return Value

wd[0] := (unsigned short)ws[0] - (unsigned short)wt[0];

wd[1] := (unsigned short)ws[1] - (unsigned short)wt[1];

.....

wd[7] := (unsigned short)ws[7] - (unsigned short)wt[7];

Requirements

Header:#include<msa.h>

2.1.57.2. v4i32 __builtin_msa_subur_w (v8u16,v8u16)

Specific

Vector Subtract Right of Unsigned Values.

v4i32 wd = __builtin_msa_subur_w (v8u16 ws,v8u16 wt);

Return Value

wd[0] := (unsigned int)ws[0] - (unsigned int)wt[0];

wd[1] := (unsigned int)ws[1] - (unsigned int)wt[1];

wd[2] := (unsigned int)ws[2] - (unsigned int)wt[2];

wd[3] := (unsigned int)ws[3] - (unsigned int)wt[3];

Requirements

Header:#include<msa.h>

2.1.58. MULSL.df

2.1.58.1. v8i16 __builtin_msa_mulsl_h (v16i8,v16i8)

Specific

Vector Multiply Left of signed Values.

v8i16 wd = __builtin_msa_mulsl_h (v16i8 ws,v16i8 wt);

Return Value

wd[0] := (short)ws[8] * (short)wt[8];

wd[1] := (short)ws[9] * (short)wt[9];

.....

wd[7] := (short)ws[15] * (short)wt[15];

Requirements

Header:#include<msa.h>

2.1.58.2. v4i32 __builtin_msa_mulsl_w (v8i16,v8i16)

Specific

Vector Multiply Left of signed Values.

v4i32 wd = __builtin_msa_mulsl_w (v8i16 ws,v8i16 wt);

Return Value

wd[0] := (int)ws[4] * (int)wt[4];

wd[1] := (int)ws[5] * (int)wt[5];

wd[2] := (int)ws[6] *(int) wt[6];

wd[3] := (int)ws[7] * (int)wt[7];

Requirements

Header:#include<msa.h>

2.1.59. MULSR.df

2.1.59.1. v8i16 __builtin_msa_mulsr_h (v16i8,v16i8)

Specific

Vector Multiply Right of Signed Values.

v8i16 wd = __builtin_msa_mulsr_h (v16i8 ws,v16i8 wt);

Return Value

wd[0] := (short)ws[0] * (short)wt[0];

wd[1] := (short)ws[1] * (short)wt[1];

.....

wd[7] := (short)ws[7] * (short)wt[7];

Requirements

Header:#include<msa.h>

2.1.59.2. v4i32 __builtin_msa_mulsr_w (v8i16,v8i16)

Specific

Vector Multiply Right of Signed Values.

v4i32 wd = __builtin_msa_mulsr_w (v8i16 ws,v8i16 wt);

Return Value

wd[0] := (int)ws[0] * (int)wt[0];

wd[1] := (int)ws[1] * (int)wt[1];

wd[2] := (int)ws[2] * (int)wt[2];

wd[3] := (int)ws[3] * (int)wt[3];

Requirements

Header:#include<msa.h>

2.1.60. MULUL.df

2.1.60.1. v8u16 __builtin_msa_mulul_h (v16u8,v16u8)

Specific

Vector Multiply Left of Unsigned Values.

v8u16 wd = __builtin_msa_mulul_h (v16u8 ws,v16u8 wt);

Return Value

wd[0] := (unsigned short)ws[8] * (unsigned short)wt[8];

wd[1] := (unsigned short)ws[9] * (unsigned short)wt[9];

.....

wd[7] := (unsigned short)ws[15] * (unsigned short)wt[15];

Requirements

Header:#include<msa.h>

2.1.60.2. v4u32 __builtin_msa_mulul_w (v8u16,v8u16)

Specific

Vector Multiply Left of Unsigned Values.

v4u32 wd = __builtin_msa_mulul_w (v8u16 ws,v8u16 wt);

Return Value

wd[0] := (unsigned int)ws[4] * (unsigned int)wt[4];

wd[1] := (unsigned int)ws[5] * (unsigned int)wt[5];

wd[2] := (unsigned int)ws[6] * (unsigned int)wt[6];

wd[3] := (unsigned int)ws[7] * (unsigned int)wt[7];

Requirements

Header:#include<msa.h>

2.1.61. MULUR.df

2.1.61.1. v8u16 __builtin_msa_mulur_h (v16u8,v16u8)

Specific

Vector Multiply Right of Unsigned Values.

v8u16 wd = __builtin_msa_mulur_h (v16u8 ws,v16u8 wt);

Return Value

wd[0] := (unsigned short)ws[0] * (unsigned short)wt[0];

wd[1] := (unsigned short)ws[1] * (unsigned short)wt[1];

.....

wd[7] := (unsigned short)ws[7] * (unsigned short)wt[7];

Requirements

Header:#include<msa.h>

2.1.61.2. v4u32 __builtin_msa_mulur_w (v8u16,v8u16)

Specific

Vector Multiply Right of Unsigned Values.

v4u32 wd = __builtin_msa_mulur_w (v8u16 ws,v8u16 wt);

Return Value

wd[0] := (unsigned int)ws[0] * (unsigned int)wt[0];

wd[1] := (unsigned int)ws[1] * (unsigned int)wt[1];

wd[2] := (unsigned int)ws[2] * (unsigned int)wt[2];

wd[3] := (unsigned int)ws[3] * (unsigned int)wt[3];

Requirements

Header:#include<msa.h>

2.2. Bitwise

2.2.1. AND.V

2.2.1.1. v16u8 __builtin_msa_and_v(v16u8,v16u8)

Specific

Bitwise And.

v16u8 wd = __builtin_msa_and_v(v16u8 ws,v16u8 wt);

Return Value

wd[0] := ws[0] & wt[0];

wd[1] := ws[1] & wt[1];

.....

wd[15] := ws[15] & wt[15];

Requirements

Header:#include<msa.h>

2.2.2. ANDI.B

2.2.2.1. v16u8 __builtin_msa_andi_b(v16u8,imm0_255)

Specific

Immediate Bitwise And.

v16u8 wd = __builtin_msa_andi_b(v16u8 ws,imm0_255 i8);

Return Value

wd[0] := ws[0] & i8;

wd[1] := ws[1] & i8;

.....

wd[15] := ws[15] & i8;

Requirements

Header:#include<msa.h>

2.2.3. BCLR.df

2.2.3.1. v16u8 __builtin_msa_bclr _b(v16u8,v16u8)

Specific

Vector Bit Clear.

v16u8 wd = __builtin_msa_bclr _b(v16u8 ws,v16u8 wt);

Return Value

wd[0] := ws[0] & (~(0x1<<wt[0]));

wd[1] := ws[1] & (~(0x1<<wt[1]));

.....

wd[15] := ws[15] & (~(0x<<wt[15]));

Requirements

Header:#include<msa.h>

eg:

bclr_test:

unsigned char a[16] ={0xff,0xff,0xff,0xff,

0xff,0xff,0xff,0xff,

```
                0xff,0xff,0xff,0xff,
                0xff,0xff,0xff,0xff};
unsigned char b[16] = {0,1,2,3
                4,5,6,7,
                0,1,2,3,
                4,5,6,7};
unsigned char c[16];
v16u8 va,vb;
v16u8 vc;
va = (v16u8)__builtin_msa_ld_b(a,0);
vb = (v16u8)__builtin_msa_ld_b(b,0);
vc = __builtin_msa_bclr_b(va,vb);
__builtin_msa_st_b((v16i8)vc,c,0);
result:
c={0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f,
  0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
```

2.2.3.2. v8u16 __builtin_msa_bclr_h(v8u16,v8u16)

Specific

Vector Bit Clear.

v8u16 wd = __builtin_msa_bclr_h(v8u16 ws,v8u16 wt);

Return Value

wd[0] := ws[0] & (~(0x1<<wt[0]));

wd[1] := ws[1] & (~(0x1<<wt[1]));

.....

wd[7] := ws[7] & (~(0x<<wt[7]));

Requirements

Header:#include<msa.h>

2.2.3.3. v4u32 __builtin_msa_bclr_w(v4u32,v4u32)

Specific

Vector Bit Clear.

v4u32 wd = __builtin_msa_bclr_w(v4u32 ws,v4u32 wt);

Return Value

wd[0] := ws[0] & (~(0x1<<wt[0]));

wd[1] := ws[1] & (~(0x1<<wt[1]));

wd[2] := ws[2] & (~(0x1<<wt[2]));

wd[3] := ws[3] & (~(0x1<<wt[3]));

Requirements

Header:#include<msa.h>

2.2.3.4. v2u64 __builtin_msa_bclr_d(v2u64,v2u64)

Specific

Vector Bit Clear.

v2u64 wd = __builtin_msa_bclr_d(v2u64 ws,v2u64 wt);

Return Value

wd[0] := ws[0] & (~(0x1<<wt[0]));

wd[1] := ws[1] & (~(0x1<<wt[1]));

Requirements

Header:#include<msa.h>

2.2.4. BCLRI.df

2.2.4.1. v16u8 __builtin_msa_bclr i_b(v16u8,imm0_7)

Specific

Immediate Bit Clear.

v16u8 wd = __builtin_msa_bclr i_b(v16u8 ws,imm0_7 m);

Return Value

wd[0] := ws[0] & (~(0x1<<m));

wd[1] := ws[1] & (~(0x1<<m));

.....

wd[15] := ws[15] & (~(0x<<m));

Requirements

Header:#include<msa.h>

2.2.4.2. v8u16 __builtin_msa_bclri_h(v8u16, imm0_15)

Specific

Immediate Bit Clear.

v8u16 wd = __builtin_msa_bclri_h(v8u16 ws, imm0_15 m);

Return Value

wd[0] := ws[0] & (~(0x1<<m));

wd[1] := ws[1] & (~(0x1<<m));

.....

wd[7] := ws[7] & (~(0x<<m));

Requirements

Header:#include<msa.h>

2.2.4.3. v4u32 __builtin_msa_bclri_w(v4u32, imm0_31)

Specific

Immediate Bit Clear.

v4u32 wd = __builtin_msa_bclri_w(v4u32 ws, imm0_31 m);

Return Value

wd[0] := ws[0] & (~(0x1<<m));

wd[1] := ws[1] & (~(0x1<<m));

wd[2] := ws[2] & (~(0x1<<m));

wd[3] := ws[3] & (~(0x1<<m));

Requirements

Header:#include<msa.h>

2.2.4.4. v2u64 __builtin_msa_bclri_d(v2u64, imm0_63)

Specific

Immediate Bit Clear.

v2u64 wd = __builtin_msa_bclri_d(v2u64 ws, imm0_63 m);

Return Value

wd[0] := ws[0] & (~(0x1<<m));

wd[1] := ws[1] & (~(0x1<<m));

Requirements

Header:#include<msa.h>

2.2.5. BINSL.df

2.2.5.1. v16u8 __builtin_msa_binsl _b(v16u8,v16u8,v16u8)

Specific

Vector Bit Insert Left.

v16u8 wd = __builtin_msa_binsl _b(v16u8 wd,v16u8 ws,v16u8 wt);

Return Value

wd[0] := binsl(ws[0],wt[0],wd[0],8);

wd[1] := binsl(ws[1],wt[1],wd[1],8);

.....

wd[15] := binsl(ws[15],wt[15],wd[15],8);

Requirements

Header:#include<msa.h>

eg:

test_binsl_b:

```
unsigned char a[16] ={0xff,0xff,0xff,0xff,
                      0xff,0xff,0xff,0xff,
                      0xff,0xff,0xff,0xff,
                      0xff,0xff,0xff,0xff};
unsigned char b[16] = {0,1,2,3,
                      4,5,6,7,
                      0,1,2,3,
                      4,5,6,7};
unsigned char c[16] = {0x00,0x00,0x00,0x00,
                      0x00,0x00,0x00,0x00,
                      0x00,0x00,0x00,0x00,
                      0x00,0x00,0x00,0x00};
v16u8 va,vb;
v16u8 vc;
va = (v16u8)__builtin_msa_ld_b(a,0);
vb = (v16u8)__builtin_msa_ld_b(b,0);
vc = (v16u8)__builtin_msa_ld_b(c,0);
vc = __builtin_msa_binsl_b(vc,va,vb);
__builtin_msa_st_b((v16i8)vc,c,0);
```

Result:

c={0x80,0xc0,0xe0,0xf0,0xf8,0xfc,0xfe,0xff,

0x80,0xc0,0xe0,0xf0,0xf8,0xfc,0xfe,0xff};

2.2.5.2. v8u16 __builtin_msa_binsl_h(v8u16,v8u16,v16u8)

Specific

Vector Bit Insert Left.

v8u16 wd = __builtin_msa_binsl_h(v8u16 wd,v8u16 ws,v16u8 wt);

Return Value

wd[0] := binsl(ws[0],wt[0],wd[0],16);

wd[1] := binsl(ws[1],wt[1],wd[1],16);

.....

wd[7] := binsl(ws[7],wt[7],wd[7],16);

Requirements

Header:#include<msa.h>

2.2.5.3. v4u32 __builtin_msa_binsl_w(v4u32,v4u32,v16u8)

Specific

Vector Bit Insert Left.

v4u32 wd = __builtin_msa_binsl_w(v4u32 wd,v4u32 ws,v16u8 wt);

Return Value

wd[0] := binsl(ws[0],wt[0],wd[0],32);

wd[1] := binsl(ws[1],wt[1],wd[1],32);

wd[2] := binsl(ws[2],wt[2],wd[2],32);

wd[3] := binsl(ws[3],wt[3],wd[3],32);

Requirements

Header:#include<msa.h>

2.2.5.4. v2u64 __builtin_msa_binsl_d(v2u64,v2u64,v16u8)

Specific

Vector Bit Insert Left.

v2u64 wd = __builtin_msa_binsl_d(v2u64 wd,v2u64 ws,v16u8 wt);

Return Value

wd[0] := binsl(ws[0],wt[0],wd[0],64);

wd[1] := binsl(ws[1],wt[1],wd[1],64);

Requirements

Header:#include<msa.h>

2.2.6. BINSLI.df

2.2.6.1. v16u8 __builtin_msa_binsli _b(v16u8,v16u8,imm0_7)

Specific

Immediate Bit Insert Left.

v16u8 wd = __builtin_msa_binsli _b(v16u8 wd,v16u8 ws,imm0_7 m);

Return Value

wd[0] := binsl(ws[0],m,wd[0],8);

wd[1] := binsl(ws[1],m,wd[1],8);

.....

wd[15] := binsl(ws[15],m,wd[15],8);

Requirements

Header:#include<msa.h>

2.2.6.2. v8u16 __builtin_msa_binsli_h(v8u16,v8u16,imm0_15)

Specific

Immediate Bit Insert Left.

v8u16 wd = __builtin_msa_binsli_h(v8u16 wd,v8u16 ws,imm0_15 m);

Return Value

wd[0] := binsl(ws[0],m,wd[0],16);

wd[1] := binsl(ws[1],m,wd[1],16);

.....

wd[7] := binsl(ws[7],m,wd[7],16);

Requirements

Header:#include<msa.h>

2.2.6.3. v4u32 __builtin_msa_binsli_w(v4u32,v4u32,imm0_31)

Specific

Immediate Bit Insert Left.

v4u32 wd = __builtin_msa_binsli_w(v4u32 wd,v4u32 ws,imm0_31 m);

Return Value

wd[0] := binsl(ws[0],m,wd[0],32);

wd[1] := binsl(ws[1],m,wd[1],32);

wd[2] := binsl(ws[2],m,wd[2],32);

wd[3] := binsl(ws[3],m,wd[3],32);

Requirements

Header:#include<msa.h>

2.2.6.4. v2u64 __builtin_msa_binsli_d(v2u64,v2u64,imm0_63)

Specific

Immediate Bit Insert Left.

v2u64 wd = __builtin_msa_binsli_d(v2u64 wd,v2u64 ws,imm0_63 m);

Return Value

wd[0] := binsl(ws[0],m,wd[0],64);

wd[1] := binsl(ws[1],m,wd[1],64);

Requirements

Header:#include<msa.h>

2.2.7. BINSR.df

2.2.7.1. v16u8 __builtin_msa_binsr _b(v16u8,v16u8,v16u8)

Specific

Vector Bit Insert Right.

v16u8 wd = __builtin_msa_binsr _b(v16u8 wd,v16u8 ws,v16u8 wt);

Return Value

wd[0] := binsr(ws[0],wt[0],wd[0],8);

wd[1] := binsr(ws[1],wt[1],wd[1],8);

.....

wd[15] := binsr(ws[15],wt[15],wd[15],8);

Requirements

Header:#include<msa.h>

2.2.7.2. v8u16 __builtin_msa_binsr_h(v8u16,v8u16,v16u8)

Specific

Vector Bit Insert Right.

v8u16 wd = __builtin_msa_binsr_h(v8u16 wd,v8u16 ws,v16u8 wt);

Return Value

wd[0] := binsr(ws[0],wt[0],wd[0],16);

wd[1] := binsr(ws[1],wt[1],wd[1],16);

.....

wd[7] := binsr(ws[7],wt[7],wd[7],16);

Requirements

Header:#include<msa.h>

2.2.7.3. v4u32 __builtin_msa_binsr_w(v4u32,v4u32,v16u8)

Specific

Vector Bit Insert Right.

v4u32 wd = __builtin_msa_binsr_w(v4u32 wd,v4u32 ws,v16u8 wt);

Return Value

wd[0] := binsr(ws[0],wt[0],wd[0],32);

wd[1] := binsr(ws[1],wt[1],wd[1],32);

wd[2] := binsr(ws[2],wt[2],wd[2],32);

wd[3] := binsr(ws[3],wt[3],wd[3],32);

Requirements

Header:#include<msa.h>

2.2.7.4. v2u64 __builtin_msa_binsr_d(v2u64,v2u64,v16u8)

Specific

Vector Bit Insert Right.

v2u64 wd = __builtin_msa_binsr_d(v2u64 wd,v2u64 ws,v16u8 wt);

Return Value

wd[0] := binsr(ws[0],wt[0],wd[0],64);

wd[1] := binsr(ws[1],wt[1],wd[1],64);

Requirements

Header:#include<msa.h>

2.2.8. BINSRI.df

2.2.8.1. v16u8 __builtin_msa_binsri _b(v16u8,v16u8,imm0_7)

Specific

Immediate Bit Insert Right.

v16u8 wd = __builtin_msa_binsri _b(v16u8 wd,v16u8 ws,imm0_7 m);

Return Value

wd[0] := binsr(ws[0],m,wd[0],8);

wd[1] := binsr(ws[1],m,wd[1],8);

.....

wd[15] := binsr(ws[15],m,wd[15],8);

Requirements

Header:#include<msa.h>

2.2.8.2. v8u16 __builtin_msa_binsri_h(v8u16,v8u16,imm0_15)

Specific

Immediate Bit Insert Right.

v8u16 wd = __builtin_msa_binsri_h(v8u16 wd,v8u16 ws,imm0_15 m);

Return Value

wd[0] := binsr(ws[0],m,wd[0],16);

wd[1] := binsr(ws[1],m,wd[1],16);

.....

wd[7] := binsr(ws[7],m,wd[7],16);

Requirements

Header:#include<msa.h>

2.2.8.3. v4u32 __builtin_msa_binsri_w(v4u32,v4u32,imm0_31)

Specific

Immediate Bit Insert Right.

v4u32 wd = __builtin_msa_binsri_w(v4u32 wd,v4u32 ws,imm0_31 m);

Return Value

wd[0] := binsr(ws[0],m,wd[0],32);

wd[1] := binsr(ws[1],m,wd[1],32);

wd[2] := binsr(ws[2],m,wd[2],32);

wd[3] := binsr(ws[3],m,wd[3],32);

Requirements

Header:#include<msa.h>

2.2.8.4. v2u64 __builtin_msa_binsri_d(v2u64,v2u64,imm0_63)

Specific

Immediate Bit Insert Right.

v2u64 wd = __builtin_msa_binsri_d(v2u64 wd,v2u64 ws,imm0_63 m);

Return Value

wd[0] := binsr(ws[0],m,wd[0],64);

wd[1] := binsr(ws[1],m,wd[1],64);

Requirements

Header:#include<msa.h>

2.2.9. BMNZ.V

2.2.9.1. v16u8 __builtin_msa_bmnz _v(v16u8,v16u8,v16u8)

Specific

Vector Bit Move if Not Zero.

v16u8 wd= __builtin_msa_bmnz _v(v16u8 wd,v16u8 ws,v16u8 wt);

Return Value

wd[0] := (ws[0] & wt[0]) || (wd[0] & (~wt[0]));

wd[1] := (ws[1] & wt[1]) || (wd[1] & (~wt[1]));

.....

wd[15] := (ws[15] & wt[15]) || (wd[15] & (~wt[15]));

Requirements

Header:#include<msa.h>

2.2.10. BMNZI.B

2.2.10.1. v16u8 __builtin_msa_bmnz i_b(v16u8,v16u8,imm0_255)

Specific

Immediate Bit Move If Not Zero.

v16u8 wd = __builtin_msa_bmnz i_b(v16u8 wd,v16u8 ws,imm0_255 i8);

Return Value

wd[0] := (ws[0] & i8) || (wd[0] & (~i8));

wd[1] := (ws[1] & i8) || (wd[1] & (~i8));

.....

wd[15] := (ws[15] & i8) || (wd[15] & (~i8));

Requirements

Header:#include<msa.h>

2.2.11. BMZ.V

2.2.11.1. v16u8 __builtin_msa_bmz_b(v16u8,v16u8,v16u8)

Specific

Vector Bit Move If Zero.

v16u8 wd = __builtin_msa_bmz_b(v16u8 wd,v16u8 ws,v16u8 wt);

Return Value

wd[0] := (ws[0] & (~wt[0])) || (wd[0] & wt[0]);

wd[1] := (ws[1] & (~wt[1])) || (wd[1] & wt[1]);

.....

wd[15] := (ws[15] & (~wt[15])) || (wd[15] & wt[15]);

Requirements

Header:#include<msa.h>

2.2.12. BMZI.B

2.2.12.1. v16u8 __builtin_msa_bmz i_b(v16u8,v16u8,imm0_255)

Specific

Immediate Bit Move If Zero.

v16u8 wd = __builtin_msa_bmz i_b(v16u8 wd,v16u8 ws,imm0_255 i8);

Return Value

wd[0] := (ws[0] & (~i8)) || (wd[0] & i8);

wd[1] := (ws[1] & (~i8)) || (wd[1] & i8);

.....

wd[15] := (ws[15] & (~i8)) || (wd[15] & i8);

Requirements

Header:#include<msa.h>

2.2.13. BNEG.df

2.2.13.1. v16u8 __builtin_msa_bneg _b(v16u8,v16u8)

Specific

Vector Bit Negate.

v16u8 wd = __builtin_msa_bneg _b(v16u8 ws,v16u8 wt);

Return Value

wd[0] := ws[0] ^ (0x1<<wt[0]);

wd[1] := ws[1] ^ (0x1<<wt[1]);

.....

wd[15] := ws[15] ^ (0x1<<wt[15]);

Requirements

Header:#include<msa.h>

test_bneg_b:

```
unsigned char a[16] = {0xf0,0xf0,0xf0,0xf0,
                       0xf0,0xf0,0xf0,0xf0,
                       0xff,0xff,0xff,0xff,
                       0xff,0xff,0xff,0xff};
unsigned char b[16] ={0,1,2,3,
                      4,5,6,7,
                      0,1,2,3,
                      4,5,6,7};
```

unsigned char c[16];

v16u8 va,vb;

v16u8 vc;

va = (v16u8)__builtin_msa_ld_b(a,0);

vb = (v16u8)__builtin_msa_ld_b(b,0);

vc = __builtin_msa_bneg_b(va,vb);

__builtin_msa_st_b((v16i8)vc,c,0);

Reslut:

c={0xf1,0xf2, 0xf4,0xf8,0xe0,0xd0,0xb0,0x70,

0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};

2.2.13.2. v8u16 __builtin_msa_bneg_h(v8u16,v8u16)

Specific

Vector Bit Negate.

v8u16 wd = __builtin_msa_bneg_h(v8u16 ws,v8u16 wt);

Return Value

wd[0] := ws[0] ^ (0x1<<wt[0]);

wd[1] := ws[1] ^ (0x1<<wt[1]);

.....

wd[7] := ws[7] ^ (0x1<<wt[7]);

Requirements

Header:#include<msa.h>

2.2.13.3. v4u32 __builtin_msa_bneg_w(v4u32,v4u32)

Specific

Vector Bit Negate.

v4u32 wd = __builtin_msa_bneg_w(v4u32 ws,v4u32 wt);

Return Value

wd[0] := ws[0] ^ (0x1<<wt[0]);

wd[1] := ws[1] ^ (0x1<<wt[1]);

wd[2] := ws[2] ^ (0x1<<wt[2]);

wd[3] := ws[3] ^ (0x1<<wt[3]);

Requirements

Header:#include<msa.h>

2.2.13.4. v2u64 __builtin_msa_bneg_d(v2u64,v2u64)

Specific

Vector Bit Negate.

v2u64 wd = __builtin_msa_bneg_d(v2u64 ws,v2u64 wt);

Return Value

wd[0] := ws[0] ^ (0x1<<wt[0]);

wd[1] := ws[1] ^ (0x1<<wt[1]);

Requirements

Header:#include<msa.h>

2.2.14. BNEGI.df

2.2.14.1. v16u8 __builtin_msa_bnegi _b(v16u8,imm0_7)

Specific

Immediate Bit Negate.

v16u8 wd = __builtin_msa_bnegi _b(v16u8 ws,imm0_7 m);

Return Value

wd[0] := ws[0] ^ (0x1<<m);

wd[1] := ws[1] ^ (0x1<<m);

.....

wd[15] := ws[15] ^ (0x1<<m);

Requirements

Header:#include<msa.h>

2.2.14.2. v8u16 __builtin_msa_bnegi_h(v8u16,imm0_15)

Specific

Immediate Bit Negate.

v8u16 wd = __builtin_msa_bnegi_h(v8u16 ws,imm0_15 m);

Return Value

wd[0] := ws[0] ^ (0x1<<m);

wd[1] := ws[1] ^ (0x1<<m);

.....

wd[7] := ws[7] ^ (0x1<<m);

Requirements

Header:#include<msa.h>

2.2.14.3. v4u32 __builtin_msa_bnegi_w(v4u32,imm0_31)

Specific

Immediate Bit Negate.

v4u32 wd = __builtin_msa_bnegi_w(v4u32 ws,imm0_31 m);

Return Value

wd[0] := ws[0] ^ (0x1<<m);

wd[1] := ws[1] ^ (0x1<<m);

wd[2] := ws[2] ^ (0x1<<m);

wd[3] := ws[3] ^ (0x1<<m);

Requirements

Header:#include<msa.h>

2.2.14.4. v2u64 __builtin_msa_bnegi_d(v2u64,imm0_63)

Specific

Immediate Bit Negate.

v2u64 wd = __builtin_msa_bnegi_d(v2u64 ws,imm0_63 m);

Return Value

wd[0] := ws[0] ^ (0x1<<m);

wd[1] := ws[1] ^ (0x1<<m);

Requirements

Header:#include<msa.h>

2.2.15. BSEL.V

2.2.15.1. v16u8 __builtin_msa_bsel_v_b(v16u8,v16u8,v16u8)

Specific

Vector Bit Select.

v16u8 wd = __builtin_msa_bsel_v_b(v16u8 wd,v16u8 ws,v16u8 wt);

Return Value

wd[0] := (ws[0] & (~wd[0])) || (wt[0] & wd[0]);

wd[1] := (ws[1] & (~wd[1])) || (wt[1] & wd[1]);

.....

wd[15] := (ws[15] & (~wd[15])) || (wt[15] & wd[15]);

Requirements

Header:#include<msa.h>

2.2.16. BSELI.B

2.2.16.1. v16u8 __builtin_msa_bseli_b(v16u8,v16u8,imm0_255)

Specific

Immediate Bit Select.

v16u8 wd = __builtin_msa_bseli_b(v16u8 wd,v16u8 ws,imm0_255 i8);

Return Value

wd[0] := (ws[0] & (~wd[0])) || (i8 & wd[0]);

wd[1] := (ws[1] & (~wd[1])) || (i8 & wd[1]);

.....

wd[15] := (ws[15] & (~wd[15])) || (i8 & wd[15]);

Requirements

Header:#include<msa.h>

2.2.17. BSET.df

2.2.17.1. v16u8 __builtin_msa_bset _b(v16u8,v16u8)

Specific

Vector Bit Set.

v16u8 wd = __builtin_msa_bset _b(v16u8 ws,v16u8 wt);

Return Value

wd[0] = ws[0] || (0x1 << wt[0]);

wd[1] = ws[1] || (0x1 << wt[1]);

.....

wd[15] = ws[15] || (0x1 << wt[15]);

Requirements

Header:#include<msa.h>

test_bset:

```
unsigned char a[16] = {0x00,0x00,0x00,0x00,
                       0x00,0x00,0x00,0x00,
                       0xff,0xff,0xff,0xff,
                       0xff,0xff,0xff,0xff};
unsigned char b[16] = {0,1,2,3,
                       4,5,6,7,
                   0,1,2,3,
                   4,5,6,7};
unsigned char c[16];
v16u8 va,vb,vc;
va = (v16u8)__builtin_msa_ld_b(a,0);
vb = (v16u8)__builtin_msa_ld_b(b,0);
vc = __builtin_msa_bset_b(va,vb);
__builtin_msa_st_b((v16i8)vc,c,0);
```

Result:

```
c[16] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,
         0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff};
```

2.2.17.2. v8u16 __builtin_msa_bset_h(v8u16,v8u16)

Specific

Vector Bit Set.

```
v8u16 wd = __builtin_msa_bset_h(v8u16 ws,v8u16 wt);
```

Return Value

wd[0] = ws[0] || (0x1 << wt[0]);

wd[1] = ws[1] || (0x1 << wt[1]);

.....

wd[7] = ws[7] || (0x1 << wt[7]);

Requirements

Header:#include<msa.h>

2.2.17.3. v4u32 __builtin_msa_bset_w(v4u32,v4u32)

Specific

Vector Bit Set.

v4u32 wd = __builtin_msa_bset_w(v4u32 ws,v4u32 wt);

Return Value

wd[0] = ws[0] || (0x1 << wt[0]);

wd[1] = ws[1] || (0x1 << wt[1]);

wd[2] = ws[2] || (0x1 << wt[2]);

wd[3] = ws[3] || (0x1 << wt[3]);

Requirements

Header:#include<msa.h>

2.2.17.4. v2u64 __builtin_msa_bset_d(v2u64,v2u64)

Specific

Vector Bit Set.

v2u64 wd = __builtin_msa_bset_d(v2u64 ws,v2u64 wt);

Return Value

wd[0] = ws[0] || (0x1 << wt[0]);

wd[1] = ws[1] || (0x1 << wt[1]);

Requirements

Header:#include<msa.h>

2.2.18. BSETI.df

2.2.18.1. v16u8 __builtin_msa_bseti _b(v16u8,imm0_7)

Specific

Immediate Bit Set.

v16u8 wd = __builtin_msa_bseti _b(v16u8 ws,imm0_7 m);

Return Value

wd[0] = ws[0] || (0x1 << m);

wd[1] = ws[1] || (0x1 << m);

.....

wd[15] = ws[15] || (0x1 << m);

Requirements

Header:#include<msa.h>

2.2.18.2. v8u16 __builtin_msa_bseti_h(v8u16,imm0_15)

Specific

Immediate Bit Set.

v8u16 wd = __builtin_msa_bseti_h(v8u16 ws,imm0_15 m);

Return Value

wd[0] = ws[0] || (0x1 << m);

wd[1] = ws[1] || (0x1 << m);

.....

wd[7] = ws[7] || (0x1 << m);

Requirements

Header:#include<msa.h>

2.2.18.3. v4u32 __builtin_msa_bseti_w(v4u32,imm0_31)

Specific

Immediate Bit Set.

v4u32 wd = __builtin_msa_bseti_w(v4u32 ws,imm0_31 m);

Return Value

wd[0] = ws[0] || (0x1 << m);

wd[1] = ws[1] || (0x1 << m);

wd[2] = ws[2] || (0x1 << m);

wd[3] = ws[3] || (0x1 << m);

Requirements

Header:#include<msa.h>

2.2.18.4. v2u64 __builtin_msa_bseti_d(v2u64,imm0_63)

Specific

Immediate Bit Set.

v2u64 wd = __builtin_msa_bseti_d(v2u64 ws,imm0_63 m);

Return Value

wd[0] = ws[0] || (0x1 << m);

wd[1] = ws[1] || (0x1 << m);

Requirements

Header:#include<msa.h>

2.2.19. NLOC.df

2.2.19.1. v16i8 __builtin_msa_nloc _b(v16i8)

Specific

Vector Leading Ones Count.

v16i8 wd = __builtin_msa_nloc _b(ws);

Return Value

wd[0] := loc(ws[0],8);

wd[1] := loc(ws[1],8);

.....

wd[15] := loc(ws[15],8);

Requirements

Header:#include<msa.h>

test_loc:

char a[16] = {0xff,0xfe,0xfc,0xf8,

0xf0,0xe0,0xc0,0x80,

0x00,0x01,0x03,0x07,

0x0f,0x1f,0x3f,0x7f};

char b[16];

v16i8 va,vb;

va = __builtin_msa_ld_b(a,0);

vb = __builtin_msa_nloc_b(va);

__builtin_msa_st_b((v16i8)vb,b,0);

Result:

b[16] = {8,7,6,5,

4,3,2,1,

0,0,0,0,

0,0,0,0};

2.2.19.2. v8i16 __builtin_msa_nloc_h(v8i16)

Specific

Vector Leading Ones Count.

v8i16 wd = __builtin_msa_nloc_h(v8i16 ws);

Return Value

wd[0] := loc(ws[0],16);

wd[1] := loc(ws[1],16);

.....

wd[7] := loc(ws[7],16);

Requirements

Header:#include<msa.h>

2.2.19.3. v4i32 __builtin_msa_nloc_w(v4i32)

Specific

Vector Leading Ones Count.

v4i32 wd = __builtin_msa_nloc_w(v4i32 ws);

Return Value

wd[0] := loc(ws[0],32);

wd[1] := loc(ws[1],32);

wd[2] := loc(ws[2],32);

wd[3] := loc(ws[3],32);

Requirements

Header:#include<msa.h>

2.2.19.4. v2i64 __builtin_msa_nloc_d(v2i64)

Specific

Vector Leading Ones Count.

v2i64 wd = __builtin_msa_nloc_d(v2i64 ws);

Return Value

wd[0] := loc(ws[0],64);

wd[1] := loc(ws[1],64);

Requirements

Header:#include<msa.h>

2.2.20. NLZC.df

2.2.20.1. v16i8 __builtin_msa_nlzc _b(v16i8)

Specific

Vector Leading Zeros Count.

v16i8 wd = __builtin_msa_nlzc _b(v16i8 ws);

Return Value

wd[0] := lzc(ws[0],8);

wd[1] := lzc(ws[1],8);

.....

wd[15] := lzc(ws[15],8);

Requirements

Header:#include<msa.h>

test_lzc:

```
char a[16] = {0xff,0xfe,0xfc,0xf8,
              0xf0,0xe0,0xc0,0x80,
              0x00,0x01,0x03,0x07,
              0x0f,0x1f,0x3f,0x7f};
```

char b[16];

v16i8 va,vb;

va = __builtin_msa_ld_b(a,0);

vb = __builtin_msa_nlzc_b(va);

__builtin_msa_st_b((v16i8)vb,b,0);

Result:

b[16] = {0,0,0,0,

0,0,0,0,

8,7,6,5,

4,3,2,1};

2.2.20.2. v8i16 __builtin_msa_nlzc_h(v8i16)

Specific

Vector Leading Zeros Count.

v8i16 wd = __builtin_msa_nlzc_h(v8i16 ws);

Return Value

wd[0] := lzc(ws[0],16);

wd[1] := lzc(ws[1],16);

.....

wd[7] := lzc(ws[7],16);

Requirements

Header:#include<msa.h>

2.2.20.3. v4i32 __builtin_msa_nlzc_w(v4i32)

Specific

Vector Leading Zeros Count.

v4i32 wd = __builtin_msa_nlzc_w(v4i32 ws);

Return Value

wd[0] := lzc(ws[0],32);

wd[1] := lzc(ws[1],32);

wd[2] := lzc(ws[2],32);

wd[3] := lzc(ws[3],32);

Requirements

Header:#include<msa.h>

2.2.20.4. v2i64 __builtin_msa_nlzc_d(v2i64)

Specific

Vector Leading Zeros Count.

v2i64 wd = __builtin_msa_nlzc_d(v2i64 ws);

Return Value

wd[0] := lzc(ws[0],64);

wd[1] := lzc(ws[1],64);

Requirements

Header:#include<msa.h>

2.2.21. NOR.V

2.2.21.1. v16u8 __builtin_msa_nor_v _b(v16u8,v16u8)

Specific

Vector Logical Negated Or.

v16u8 wd = __builtin_msa_nor_v _b(v16u8 ws,v16u8 wt);

Return Value

wd[0] := ~(ws[0] | wt[0]);

wd[1] := ~(ws[1] | wt[1]);

.....

wd[15] := ~(ws[15] | wt[15]);

Requirements

Header:#include<msa.h>

2.2.22. NORI.B

2.2.22.1. v16u8 __builtin_msa_nori_v _b(v16u8,imm0_255)

Specific

Immediate Logical Negated Or.

v16u8 wd = __builtin_msa_nori_v _b(v16u8 ws,imm0_255 i8);

Return Value

wd[0] := ~(ws[0] | i8);

wd[1] := ~(ws[1] | i8);

.....

wd[15] := ~(ws[15] | i8);

Requirements

Header:#include<msa.h>

2.2.23. PCNT.df

2.2.23.1. v16i8 __builtin_msa_pcnt _b(v16i8)

Specific

Vector Population Count.

v16i8 wd = __builtin_msa_pcnt _b(v16i8 ws);

Return Value

wd[0] := pcnt (ws[0],8);

wd[1] := pcnt (ws[1],8);

.....

wd[15] := pcnt (ws[15],8);

Requirements

Header:#include<msa.h>

test_pcnt:

signed char a[16] = {1,2,3,4,

5,6,7,8,

9,10,11,12,

13,14,15,16};

signed char b[16];

v16i8 va,vb;

va = __builtin_msa_ld_b(a,0);

vb = __builtin_msa_pcnt_b(va);

__builtin_msa_st_b((v16i8)vb,b,0);

Result:

b[16]={1,1,2,1,

2,2,3,1,

2,2,3,2,

3,3,4,1}

2.2.23.2. v8i16 __builtin_msa_pcnt_h(v8i16)

Specific

Vector Population Count.

v8i16 wd = __builtin_msa_pcnt_h(v8i16 ws);

Return Value

wd[0] := pcnt (ws[0],16);

wd[1] := pcnt (ws[1],16);

.....

wd[7] := pcnt (ws[7],16);

Requirements

Header:#include<msa.h>

2.2.23.3. v4i32 __builtin_msa_pcnt_w(v4i32)

Specific

Vector Population Count.

v4i32 wd = __builtin_msa_pcnt_w(v4i32 ws);

Return Value

wd[0] := pcnt (ws[0],32);

wd[1] := pcnt (ws[1],32);

wd[2] := pcnt (ws[2],32);

wd[3] := pcnt (ws[3],32);

Requirements

Header:#include<msa.h>

2.2.23.4. v2i64 __builtin_msa_pcnt_d(v2i64)

Specific

Vector Population Count.

v2i64 wd = __builtin_msa_pcnt_d(v2i64 ws);

Return Value

wd[0] := pcnt (ws[0],64);

wd[1] := pcnt (ws[1],64);

Requirements

Header:#include<msa.h>

2.2.24. OR.V

2.2.24.1. v16u8 __builtin_msa_or_v _b(v16u8,v16u8)

Specific

Vector Logical Or.

v16u8 wd = __builtin_msa_or_v _b(v16u8 ws,v16u8 wt);

Return Value

wd[0] := ws[0] | wt[0];

wd[1] := ws[1] | wt[1];

.....

wd[15] := ws[15] | wt[15];

Requirements

Header:#include<msa.h>

2.2.25. ORI.B

2.2.25.1. v16u8 __builtin_msa_ori_v _b(v16u8,imm0_255)

Specific

Immediate Logical Or.

v16u8 wd = __builtin_msa_ori_v _b(v16u8 ws,imm0_255 i8);

Return Value

wd[0] := ws[0] | i8;

wd[1] := ws[1] | i8;

.....

wd[15] := ws[15] | i8;

Requirements

Header:#include<msa.h>

2.2.26. SLL.df

2.2.26.1. v16i8 __builtin_msa_sll _b(v16i8,v16i8)

Specific

Vector Shift Left.

v16i8 wd = __builtin_msa_sll _b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := ws[0] << wt[0];

wd[1] := ws[1] << wt[1];

.....

wd[15] := ws[15] << wt[15];

Requirements

Header:#include<msa.h>

2.2.26.2. v8i16 __builtin_msa_sll_h(v8i16,v8i16)

Specific

Vector Shift Left.

v8i16 wd = __builtin_msa_sll_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := ws[0] << wt[0];

wd[1] := ws[1] << wt[1];

.....

wd[7] := ws[7] << wt[7];

Requirements

Header:#include<msa.h>

2.2.26.3. v4i32 __builtin_msa_sll_w(v4i32,v4i32)

Specific

Vector Shift Left.

v4i32 wd = __builtin_msa_sll_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := ws[0] << wt[0];

wd[1] := ws[1] << wt[1];

wd[2] := ws[2] << wt[2];

wd[3] := ws[3] << wt[3];

Requirements

Header:#include<msa.h>

2.2.26.4. v2i64 __builtin_msa_sll_d(v2i64,v2i64)

Specific

Vector Shift Left.

v2i64 wd = __builtin_msa_sll_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := ws[0] << wt[0];

wd[1] := ws[1] << wt[1];

Requirements

Header:#include<msa.h>

2.2.27. SLLI.df

2.2.27.1. v16i8 __builtin_msa_slli_b(v16i8,imm0_7)

Specific

Immediate Shift Left.

v16i8 wd = __builtin_msa_slli_b(v16i8 ws,imm0_7 m);

Return Value

wd[0] := ws[0] << m;

wd[1] := ws[1] << m;

.....

wd[15] := ws[15] << m;

Requirements

Header:#include<msa.h>

2.2.27.2. v8i16 __builtin_msa_slli_h(v8i16,imm0_15)

Specific

Immediate Shift Left.

v8i16 wd = __builtin_msa_slli_h(v8i16 ws,imm0_15 m);

Return Value

wd[0] := ws[0] << m;

wd[1] := ws[1] << m;

.....

wd[7] := ws[7] << m;

Requirements

Header:#include<msa.h>

2.2.27.3. v4i32 __builtin_msa_slli_w(v4i32,imm0_31)

Specific

Immediate Shift Left.

v4i32 wd = __builtin_msa_slli_w(v4i32 ws,imm0_31 m);

Return Value

wd[0] := ws[0] << m;

wd[1] := ws[1] << m;

wd[2] := ws[2] << m;

wd[3] := ws[3] << m;

Requirements

Header:#include<msa.h>

2.2.27.4. v2i64 __builtin_msa_slli_d(v2i64,imm0_63)

Specific

Immediate Shift Left.

v2i64 wd = __builtin_msa_slli_d(v2i64 ws,imm0_63 m);

Return Value

wd[0] := ws[0] << m;

wd[1] := ws[1] << m;

Requirements

Header:#include<msa.h>

2.2.28. SRA.df

2.2.28.1. v16i8 __builtin_msa_sra _b(v16i8,v16i8)

Specific

Vector Shift Right Arithmetic.

v16i8 wd=__builtin_msa_sra _b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := ws[0] >> wt[0];

wd[1] := ws[1] >> wt[1];

.....

wd[15] := ws[15] >> wt[15];

Requirements

Header:#include<msa.h>

2.2.28.2. v8i16 __builtin_msa_sra_h(v8i16,v8i16)

Specific

Vector Shift Right Arithmetic.

v8i16 wd = __builtin_msa_sra_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := ws[0] >> wt[0];

wd[1] := ws[1] >> wt[1];

.....

wd[7] := ws[7] >> wt[7];

Requirements

Header:#include<msa.h>

2.2.28.3. v4i32 __builtin_msa_sra_w(v4i32,v4i32)

Specific

Vector Shift Right Arithmetic.

v4i32 wd = __builtin_msa_sra_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := ws[0] >> wt[0];

wd[1] := ws[1] >> wt[1];

wd[2] := ws[2] >> wt[2];

wd[3] := ws[3] >> wt[3];

Requirements

Header:#include<msa.h>

2.2.28.4. v2i64 __builtin_msa_sra_d(v2i64,v2i64)

Specific

Vector Shift Right Arithmetic.

v2i64 wd = __builtin_msa_sra_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := ws[0] >> wt[0];

wd[1] := ws[1] >> wt[1];

Requirements

Header:#include<msa.h>

2.2.29. SRAI.df

2.2.29.1. v16i8 __builtin_msa_srai_b(v16i8,imm0_7)

Specific

Immediate Shift Right Arithmetic.

v16i8 wd = __builtin_msa_srai_b(v16i8 ws,imm0_7 m);

Return Value

wd[0] := ws[0] >> m;

wd[1] := ws[1] >> m;

.....

wd[15] := ws[15] >> m;

Requirements

Header:#include<msa.h>

2.2.29.2. v8i16 __builtin_msa_srai_h(v8i16,imm0_15)

Specific

Immediate Shift Right Arithmetic.

v8i16 wd = __builtin_msa_srai_h(v8i16 ws,imm0_15 m);

Return Value

wd[0] := ws[0] >> m;

wd[1] := ws[1] >> m;

.....

wd[7] := ws[7] >> m;

Requirements

Header:#include<msa.h>

2.2.29.3. v4i32 __builtin_msa_srai_w(v4i32,imm0_31)

Specific

Immediate Shift Right Arithmetic.

v4i32 wd = __builtin_msa_srai_w(v4i32 ws,imm0_31 m);

Return Value

wd[0] := ws[0] >> m;

wd[1] := ws[1] >> m;

wd[2] := ws[2] >> m;

wd[3] := ws[3] >> m;

Requirements

Header:#include<msa.h>

2.2.29.4. v2i64 __builtin_msa_srai_d(v2i64,imm0_63)

Specific

Immediate Shift Right Arithmetic.

v2i64 wd = __builtin_msa_srai_d(v2i64 ws,imm0_63 m);

Return Value

wd[0] := ws[0] >> m;

wd[1] := ws[1] >> m;

Requirements

Header:#include<msa.h>

2.2.30. SRAR.df

2.2.30.1. v16i8 __builtin_msa_srar _b(v16i8,v16i8)

Specific

Vector Shift Right Arithmetic Rounded.

v16i8 wd = __builtin_msa_srar _b(v16i8 ws,v16i8 wd);

Return Value

wd[0] := srar (ws[0] , wt[0] , 8);

wd[1] := srar (ws[1] , wt[1] , 8);

.....

wd[15] := srar (ws[15] , wt[15] , 8);

Requirements

Header:#include<msa.h>

2.2.30.2. v8i16 __builtin_msa_srar_h(v8i16,v8i16)

Specific

Vector Shift Right Arithmetic Rounded.

v8i16 wd = __builtin_msa_srar_h(v8i16 ws,v8i16 wd);

Return Value

wd[0] := srar (ws[0] , wt[0] , 16);

wd[1] := srar (ws[1] , wt[1] , 16);

.....

wd[7] := srar (ws[7] , wt[7] , 16);

Requirements

Header:#include<msa.h>

2.2.30.3. v4i32 __builtin_msa_srar_w(v4i32,v4i32)

Specific

Vector Shift Right Arithmetic Rounded.

v4i32 wd = __builtin_msa_srar_w(v4i32 ws,v4i32 wd);

Return Value

wd[0] := srar (ws[0] , wt[0] , 32);

wd[1] := srar (ws[1] , wt[1] , 32);

wd[2] := srar (ws[2] , wt[2] , 32);

wd[3] := srar (ws[3] , wt[3] , 32);

Requirements

Header:#include<msa.h>

2.2.30.4. v2i64 __builtin_msa_srar_d(v2i64,v2i64)

Specific

Vector Shift Right Arithmetic Rounded.

v2i64 wd = __builtin_msa_srar_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := srar (ws[0] , wt[0] , 64);

wd[1] := srar (ws[1] , wt[1] , 64);

Requirements

Header:#include<msa.h>

2.2.31. SRARI.df

2.2.31.1. v16i8 __builtin_msa_srari_b(v16i8,imm0_7)

Specific

Immediate Shift Right Arithmetic Rounded.

v16i8 wd = __builtin_msa_srari_b(v16i8 ws,imm0_7 m);

Return Value

wd[0] := srar (ws[0] , m , 8);

wd[1] := srar (ws[1] , m , 8);

.....

wd[15] := srar (ws[15] , m , 8);

Requirements

Header:#include<msa.h>

2.2.31.2. v8i16 __builtin_msa_srari_h(v8i16,imm0_15)

Specific

Immediate Shift Right Arithmetic Rounded.

v8i16 wd = __builtin_msa_srari_h(v8i16 ws,imm0_15 m);

Return Value

wd[0] := srar (ws[0] , m , 16);

wd[1] := srar (ws[1] , m , 16);

.....

wd[7] := srar (ws[7] , m , 16);

Requirements

Header:#include<msa.h>

2.2.31.3. v4i32 __builtin_msa_srari_w(v4i32,imm0_31)

Specific

Immediate Shift Right Arithmetic Rounded.

v4i32 wd = __builtin_msa_srari_w(v4i32 ws,imm0_31 m);

Return Value

wd[0] := srar (ws[0] , m , 32);

wd[1] := srar (ws[1] , m , 32);

wd[2] := srar (ws[2] , m , 32);

wd[3] := srar (ws[3] , m , 32);

Requirements

Header:#include<msa.h>

2.2.31.4. v2i64 __builtin_msa_srari_d(v2i64,imm0_63)

Specific

Immediate Shift Right Arithmetic Rounded.

v2i64 wd = __builtin_msa_srari_d(v2i64 ws,imm0_63 m);

Return Value

wd[0] := srar (ws[0] , m , 64);

wd[1] := srar (ws[1] , m , 64);

Requirements

Header:#include<msa.h>

2.2.32. SRL.df

2.2.32.1. v16i8 __builtin_msa_srl _b(v16i8,v16i8)

Specific

Vector Shift Right Logical.

v16i8 wd = __builtin_msa_srl _b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := ws[0] >> wt[0];

wd[1] := ws[1] >> wt[1];

.....

wd[15] := ws[15] >> wt[15];

Requirements

Header:#include<msa.h>

2.2.32.2. v8i16 __builtin_msa_srl_h(v8i16,v8i16)

Specific

Vector Shift Right Logical.

v8i16 wd = __builtin_msa_srl_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := ws[0] >> wt[0];

wd[1] := ws[1] >> wt[1];

.....

wd[7] := ws[7] >> wt[7];

Requirements

Header:#include<msa.h>

2.2.32.3. v4i32 __builtin_msa_srl_w(v4i32,v4i32)

Specific

Vector Shift Right Logical.

v4i32 wd = __builtin_msa_srl_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := ws[0] >> wt[0];

wd[1] := ws[1] >> wt[1];

wd[2] := ws[2] >> wt[2];

wd[3] := ws[3] >> wt[3];

Requirements

Header:#include<msa.h>

2.2.32.4. v2i64 __builtin_msa_srl_d(v2i64,v2i64)

Specific

Vector Shift Right Logical.

v2i64 wd = __builtin_msa_srl_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := ws[0] >> wt[0];

wd[1] := ws[1] >> wt[1];

Requirements

Header:#include<msa.h>

2.2.33. SRLI.df

2.2.33.1. v16i8 __builtin_msa_srli_b(v16i8,imm0_7)

Specific

Immediate Shift Right Logical.

v16i8 wd = __builtin_msa_srli_b(v16i8 ws,imm0_7 m);

Return Value

wd[0] := ws[0] >> m;

wd[1] := ws[1] >> m;

.....

wd[15] := ws[15] >> m;

Requirements

Header:#include<msa.h>

2.2.33.2. v8i16 __builtin_msa_srli_h(v8i16,imm0_15)

Specific

Immediate Shift Right Logical.

v8i16 wd = __builtin_msa_srli_h(v8i16 ws,imm0_15 m);

Return Value

wd[0] := ws[0] >> m;

wd[1] := ws[1] >> m;

.....

wd[7] := ws[7] >> m;

Requirements

Header:#include<msa.h>

2.2.33.3. v4i32 __builtin_msa_srli_w(v4i32,imm0_31)

Specific

Immediate Shift Right Logical.

v4i32 wd = __builtin_msa_srli_w(v4i32 ws,imm0_31 m);

Return Value

wd[0] := ws[0] >> m;

wd[1] := ws[1] >> m;

wd[2] := ws[2] >> m;

wd[3] := ws[3] >> m;

Requirements

Header:#include<msa.h>

2.2.33.4. v2i64 __builtin_msa_srli_d(v2i64,imm0_63)

Specific

Immediate Shift Right Logical.

v2i64 wd = __builtin_msa_srli_d(v2i64 ws,imm0_63 m);

Return Value

wd[0] := ws[0] >> m;

wd[1] := ws[1] >> m;

Requirements

Header:#include<msa.h>

2.2.34. SRLR.df

2.2.34.1. v16i8 __builtin_msa_srlr _b(v16i8,v16i8)

Specific

Vector Shift Right Logical Rounded.

v16i8 wd = __builtin_msa_srlr _b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := srlr (ws[0] , wt[0] , 8);

wd[1] := srlr (ws[1] , wt[1] , 8);

.....

wd[15] := srlr (ws[15] , wt[15] , 8);

Requirements

Header:#include<msa.h>

2.2.34.2. v8i16 __builtin_msa_srlr_h(v8i16,v8i16)

Specific

Vector Shift Right Logical Rounded.

v8i16 wd = __builtin_msa_srlr_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := srlr (ws[0] , wt[0] , 16);

wd[1] := srlr (ws[1] , wt[1] , 16);

.....

wd[7] := srlr (ws[7] , wt[7] , 16);

Requirements

Header:#include<msa.h>

2.2.34.3. v4i32 __builtin_msa_srlr_w(v4i32,v4i32)

Specific

Vector Shift Right Logical Rounded.

v4i32 wd = __builtin_msa_srlr_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := srlr (ws[0] , wt[0] , 32);

wd[1] := srlr (ws[1] , wt[1] , 32);

wd[2] := srlr (ws[2] , wt[2] , 32);

wd[3] := srlr (ws[3] , wt[3] , 32);

Requirements

Header:#include<msa.h>

2.2.34.4. v2i64 __builtin_msa_srlr_d(v2i64,v2i64)

Specific

Vector Shift Right Logical Rounded.

v2i64 wd = __builtin_msa_srlr_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := srlr (ws[0] , wt[0] , 64);

wd[1] := srlr (ws[1] , wt[1] , 64);

Requirements

Header:#include<msa.h>

2.2.35. SRLRI.df

2.2.35.1. v16i8 __builtin_msa_srlri_b(v16i8,imm0_7)

Specific

Immediate Shift Right Logical Rounded.

v16i8 wd = __builtin_msa_srlri_b(v16i8 ws,imm0_7 m);

Return Value

wd[0] := srlr (ws[0] , m , 8);

wd[1] := srlr (ws[1] , m , 8);

.....

wd[15] := srlr (ws[15] , m , 8);

Requirements

Header:#include<msa.h>

2.2.35.2. v8i16 __builtin_msa_srlri_h(v8i16,imm0_15)

Specific

Immediate Shift Right Logical Rounded.

v8i16 wd = __builtin_msa_srlri_h(v8i16 ws,imm0_15 m);

Return Value

wd[0] := srlr (ws[0] , m , 16);

wd[1] := srlr (ws[1] , m , 16);

.....

wd[7] := srlr (ws[7] , m , 16);

Requirements

Header:#include<msa.h>

2.2.35.3. v4i32 __builtin_msa_srlri_w(v4i32,imm0_31)

Specific

Immediate Shift Right Logical Rounded.

v4i32 wd = __builtin_msa_srlri_w(v4i32 ws,imm0_31 m);

Return Value

wd[0] := srlr (ws[0] , m , 32);

wd[1] := srlr (ws[1] , m , 32);

wd[2] := srlr (ws[2] , m , 32);

wd[3] := srlr (ws[3] , m , 32);

Requirements

Header:#include<msa.h>

2.2.35.4. v2i64 __builtin_msa_srlri_d(v2i64,imm0_63)

Specific

Immediate Shift Right Logical Rounded.

v2i64 wd = __builtin_msa_srlri_d(v2i64 ws,imm0_63 m);

Return Value

wd[0] := srlr (ws[0] , m , 64);

wd[1] := srlr (ws[1] , m , 64);

Requirements

Header:#include<msa.h>

2.2.36. XOR.V

2.2.36.1. v16u8 __builtin_msa_xor_v(v16u8,v16u8)

Specific

Vector Logical Exclusive Or.

v16u8 wd = __builtin_msa_xor_v(v16u8 ws,v16u8 wt);

Return Value

wd[0] := ws[0] ^ wt[0];

wd[1] := ws[1] ^ wt[1];

.....

wd[15] := ws[15] ^ wt[1]6;

Requirements

Header:#include<msa.h>

2.2.37. XORI.B

2.2.37.1. v16u8 __builtin_msa_xori_v(v16u8,imm0_255)

Specific

Immediate Logical Exclusive Or.

v16u8 wd = __builtin_msa_xori_v(v16u8 ws,imm0_255 i8);

Return Value

wd[0] := ws[0] ^ i8;

wd[1] := ws[1] ^ i8;

.....

wd[15] := ws[15] ^ i8;

Requirements

Header:#include<msa.h>

2.2.38. PCNT.V

2.2.38.1. v16i8 __builtin_msa_pcnt_v (v16i8)

Specific

Vector Population.

v16i8 wd = __builtin_msa_pcnt_v (v16i8 ws);

Return Value

wd[0] := pcnt (ws,128);

wd[1] := 0;

wd[2] := 0;

wd[3] := 0;

wd[4] := 0;

wd[5] := 0;

wd[6] := 0;

wd[7] := 0;

wd[8] := 0;

wd[9] := 0;

wd[10] := 0;

wd[11] := 0;

wd[12] := 0;

wd[13] := 0;

wd[14] := 0;

wd[15] := 0;

Requirements

Header:#include<msa.h>

2.2.39. BMR.V

2.2.39.1. v16i8 __builtin_msa_bmr_v _b(v16i8,v16i8,v16i8)

Specific

Vector Bit Move based on register.

v16i8 wd = __builtin_msa_bmr_v _b(v16i8 ws,v16i8 wt,v16i8 wr);

Return Value

wd[0] := (ws[0]&wr[0]) | (wt[0]&(~wr[0]));

wd[1] := (ws[1]&wr[1]) | (wt[1]&(~wr[1]));

.....

wd[15] := (ws[15]&wr[15]) | (wt[15]&(~wr[15]));

Requirements

Header:#include<msa.h>

2.2.40. BEXT.df

2.2.40.1. v16i8 __builtin_msa_bext_b (v16i8)

Specific

Vector Bit Extract.

v16i8 wd = __builtin_msa_bext_b (v16i8 ws);

Return Value

wd=0;

t=0;

t = t | (((ws[0]>>7)&0x1) <<0)

t = t | (((ws[1]>>7)&0x1) <<1)

.....

t = t | (((ws[15]>>7)&0x1) <<15)

(v8i16)wd[0]=t;

Requirements

Header:#include<msa.h>

Graphic example:

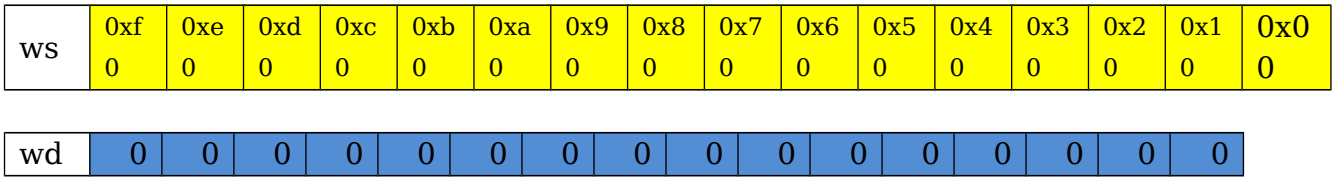
ws[16]={0x00,0x10,0x2f,0x30,

0x40,0x50,0x60,0x70,

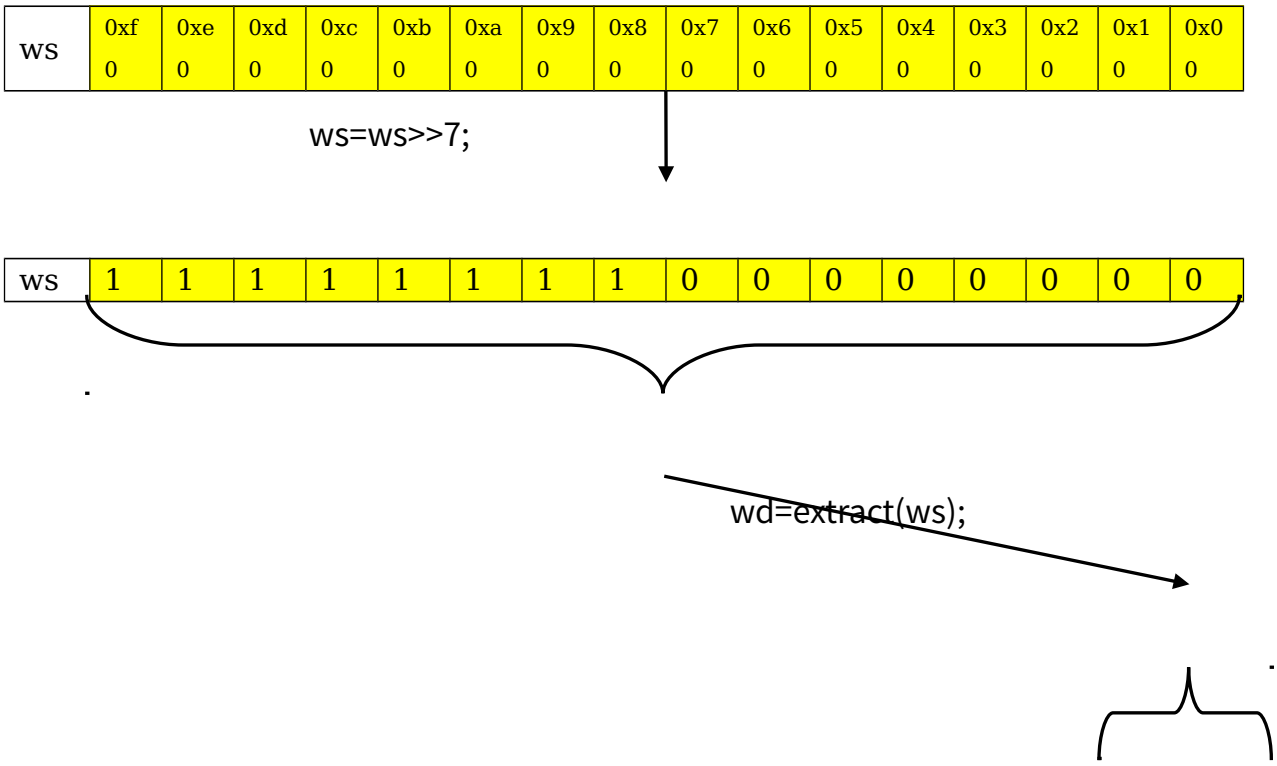
0x80,0x90,0xa0,0xb0,

0xc0,0xd0,0xe0,0xf0};

step1:



step2:



2.2.40.2. v8i16 __builtin_msa_bext_h (v8i16)

Specific

Vector Bit Extract.

v8i16 wd = __builtin_msa_bext_h (v8i16 ws);

Return Value

t=0;

t = t | (((ws[0]>>15)&0x1)<<0)

t = t | (((ws[1]>>15)&0x1)<<1)

.....

t = t | (((ws[7]>>15)&0x1)<<7)

(v16i8)wd[0]=t;

Requirements

Header:#include<msa.h>

2.2.40.3. v4i32 __builtin_msa_bext_w (v4i32)

Specific

Vector Bit Extract.

v4i32 wd = __builtin_msa_bext_w (v4i32 ws);

Return Value

t=0;

t = t | (((ws[0]>>31)&0x1)<<0)

t = t | (((ws[1]>>31)&0x1)<<1)

.....

t = t | (((ws[4]>>31)&0x1)<<4)

(v16i8)wd[0]=t;

Requirements

Header:#include<msa.h>

2.2.41. BEXP.df

2.2.41.1. v16i8 __builtin_msa_bexp_b (v16i8)

Specific

Vector Bit Expand.

v16i8 wd = __builtin_msa_bexp_b (v16i8 ws);

Return Value

wd[0] := (((v4i32)ws[0])&(0x1<<0)) ? one8 : zero8;

wd[1] := (((v4i32)ws[0])&(0x1<<1)) ? one8 : zero8;

.....

wd[15] := (((v4i32)ws[0])&(0x1<<15)) ? one8 : zero8;

Requirements

Header:#include<msa.h>

Graphic example:

ws[16]={0x0f,0x10,0x2f,0x30,

0x40,0x50,0x60,0x70,

0x80,0x90,0xa0,0xb0,

0xc0,0xd0,0xe0,0xf0};

step1:

ws	0xf0	0xe0	0xd0	0xc0	0xb0	0xa0	0x90	0x80	0x70	0x60	0x50	0x40	0x30	0x20	0x10	0x00
wd	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

step2:

ws=Expand(ws);

ws	0xf0	0xe0	0xd0	0xc0	0xb0	0xa0	0x90	0x80	0x70	0x60	0x50	0x40	0x30	0x20	0x10	0x0f

ws=expand(ws)

ws	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1

wd[n] = ws[n] ? 0xff : 0x00

wd	0x00	0x00	0x00	0xff	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0xff	0xff	0xff	0xff

2.2.41.2. v16i8 __builtin_msa_bexp_2b (v16i8)

Specific

Vector Bit Expand.

v16i8 wd = __builtin_msa_bexp_2b (v16i8 ws);

Return Value

wd = 0;

wd[0] := wd[0] | (((v2i64)ws[0])&(0x1<<0)) ? 0x03 : zero8;

wd[0] := wd[0] | (((v2i64)ws[0])&(0x1<<1)) ? 0x0c : zero8;

wd[0] := wd[0] | (((v2i64)ws[0])&(0x1<<2)) ? 0x30 : zero8;

wd[0] := wd[0] | (((v2i64)ws[0])&(0x1<<3)) ? 0xc0 : zero8;

wd[1] := wd[1] | (((v2i64)ws[0])&(0x1<<4)) ? 0x03: zero8;

wd[1] := wd[1] | (((v2i64)ws[0])&(0x1<<5)) ? 0x0c : zero8;

wd[1] := wd[1] | (((v2i64)ws[0])&(0x1<<6)) ? 0x30 : zero8;

wd[1] := wd[1] | (((v2i64)ws[0])&(0x1<<7)) ? 0xc0 : zero8;

.....

wd[15] := wd[15] | (((v2i64)ws[0])&(0x1<<60)) ? 0x03 : zero8;

wd[15] := wd[15] | (((v2i64)ws[0])&(0x1<<61)) ? 0x0c : zero8;

wd[15] := wd[15] | (((v2i64)ws[0])&(0x1<<62)) ? 0x30 : zero8;

wd[15] := wd[15] | (((v2i64)ws[0])&(0x1<<63)) ? 0xc0 : zero8;

Requirements

Header:#include<msa.h>

Graphic example:

ws[16]={0x00,0x10,0x2f,0x30,

0x40,0x50,0x60,0x70,

0x80,0x90,0xa0,0xb0,

0xc0,0xd0,0xe0,0xf0};

step1:

ws	0xf0	0xe0	0xd0	0xc0	0xb0	0xa0	0x90	0x80	0x70	0x60	0x50	0x40	0x30	0x20	0x10	0x00
wd	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

step2:

ws	0xf0	0xe0	0xd0	0xc0	0xb0	0xa0	0x90	0x80	0x70	0x60	0x50	0x40	0x30	0x20	0x10	0x00

wd=expand(ws);

wd	0x3f	0x00	0x3c	0x00	0x33	0x00	0x30	0x00	0x0f	0x00	0x0c	0x00	0x03	0x00	0x00	0x00

2.2.41.3. v16i8 __builtin_msa_bexp_4b (v16i8)

Specific

Vector Bit Expand.

v16i8 wd = __builtin_msa_bexp_4b (v16i8 ws);

Return Value

wd = 0;

wd[0] := wd[0] | (((v4i32)ws[0])&(0x1<<0)) ? 0x0f : zero8;

wd[0] := wd[0] | (((v4i32)ws[0])&(0x1<<1)) ? 0xf0 : zero8;

wd[1] := wd[1] | (((v4i32)ws[0])&(0x1<<2)) ? 0x0f : zero8;

wd[1] := wd[1] | (((v4i32)ws[0])&(0x1<<3)) ? 0xf0 : zero8;

.....

wd[15] := wd[15] | (((v4i32)ws[0])&(0x1<<30)) ? 0x0f : zero8;

wd[15] := wd[15] | (((v4i32)ws[0])&(0x1<<31)) ? 0xf0 : zero8;

Requirements

Header:#include<msa.h>

2.2.41.4. v8i16 __builtin_msa_bexp_h (v8i16)

Specific

Vector Bit Expand.

v8i16 wd = __builtin_msa_bexp_h (v8i16 ws);

Return Value

wd[0] := (((v4i32)ws[0])&(0x1<<0)) ? one16 : zero16;

wd[1] := (((v4i32)ws[0])&(0x1<<1)) ? one16 : zero16;

.....

wd[7] := (((v4i32)ws[0])&(0x1<<7)) ? one16 : zero16;

Requirements

Header:#include<msa.h>

2.2.41.5. v4i32 __builtin_msa_bexp_w (v4i32)

Specific

Vector Bit Expand.

v4i32 wd = __builtin_msa_bexp_w (v4i32 ws);

Return Value

wd[0] := (ws[0]&0x1) ? one32 : zero32;

wd[1] := (ws[0]&0x2) ? one32 : zero32;

wd[2] := (ws[0]&0x4) ? one32 : zero32;

wd[3] := (ws[0]&0x8) ? one32 : zero32;

Requirements

Header:#include<msa.h>

2.2.42. MOVBT.B

2.2.42.1. v16i8 __builtin_msa_movbt_b (v16i8,v16i8)

Specific

Move byte extract by bit-mask.

v16i8 wd = __builtin_msa_movbt_b (v16i8 ws,v16i8 wt);

Return Value

j=0;

wd = 0;

((v8i16)wt & (0x1 <<0)) ? (wd[j] = ws[0], j=j+1):default

((v8i16)wt & (0x1 <<1)) ? (wd[j] = ws[1], j=j+1):default

.....

((v8i16)wt & (0x1 <<15)) ? (wd[j] = ws[15], j=j+1):default

Requirements

Header:#include<msa.h>

test_movebt_b:

```
char a[16]={0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,
            0x00,0x00,0xff,0xaa};
char b[16]={0x00,0x01,0x02,0x03,
            0x04,0x05,0x06,0x07,
            0x08,0x09,0x10,0x11,
            0x12,0x13,0x14,0x15};
char c[16]=0;
v16i8 va,vb,vc;
va = __builtin_msa_ld_b(a,0);
```

vb = __builtin_msa_ld_b(b,0);

vc = __builtin_msa_movbt_b(vb,va);

__builtin_msa_st_b(vc,c,0);

Result:

c={0x01,0x03,0x05,0x07,

0x08,0x09,0x10,0x11,

0x12,0x13,0x14,0x15,

0x00,0x00,0x00,0x00};

Graphic example:

wt[16]={0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,

0x00,0x00,0xff,0xaa};

ws[16]={0x00,0x01,0x02,0x03,

0x04,0x05,0x06,0x07,

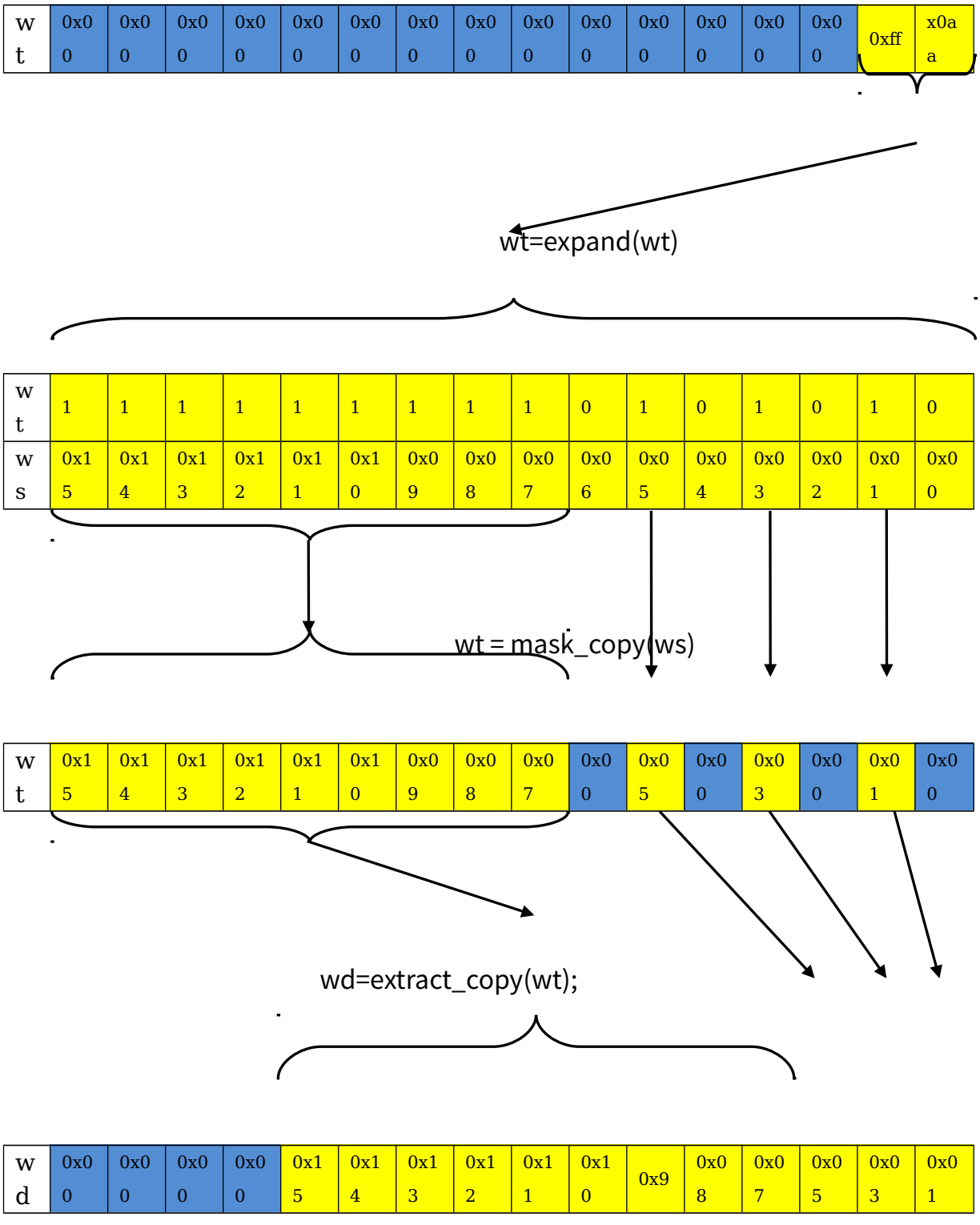
0x08,0x09,0x10,0x11,

0x12,0x13,0x14,0x15};

step1:

ws	0x15	0x14	0x13	0x12	0x11	0x10	0x09	0x08	0x07	0x06	0x05	0x04	0x03	0x02	0x01	0x00
wt	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0xff	0x00

step2:


wt
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xff x0aa
wt=expand(wt)
wt
1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0
ws
0x15 0x14 0x13 0x12 0x11 0x10 0x09 0x08 0x07 0x06 0x05 0x04 0x03 0x02 0x01 0x00
wt = mask_copy(ws)
wt
0x15 0x14 0x13 0x12 0x11 0x10 0x09 0x08 0x07 0x00 0x05 0x00 0x03 0x00 0x01 0x00
wd=extract_copy(wt);
wd
0x00 0x00 0x00 0x00 0x15 0x14 0x13 0x12 0x11 0x10 0x9 0x08 0x07 0x05 0x03 0x01

2.2.43. MOVBP.B

2.2.43.1. v16i8 __builtin_msa_movbp_b (v16i8,v16i8)

Specific

Move byte expand by bit-mask.

v16i8 wd = __builtin_msa_movbp_b (v16i8 ws,v16i8 wt);

Return Value

j=0;

wd=0;

((v8i16)wt & (0x1 <<0)) ? (wd[0] = ws[j], j=j+1):default

((v8i16)wt & (0x1 <<1)) ? (wd[1] = ws[j], j=j+1):default

.....

((v8i16)wt & (0x1 <<15)) ? (wd[15] = ws[j], j=j+1):default

Requirements

Header:#include<msa.h>

Graphic example:

wt[16]={0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,

0x00,0x00,0xff,0xaa};

ws[16]={0x00,0x01,0x02,0x03,

0x04,0x05,0x06,0x07,

0x08,0x09,0x10,0x11,

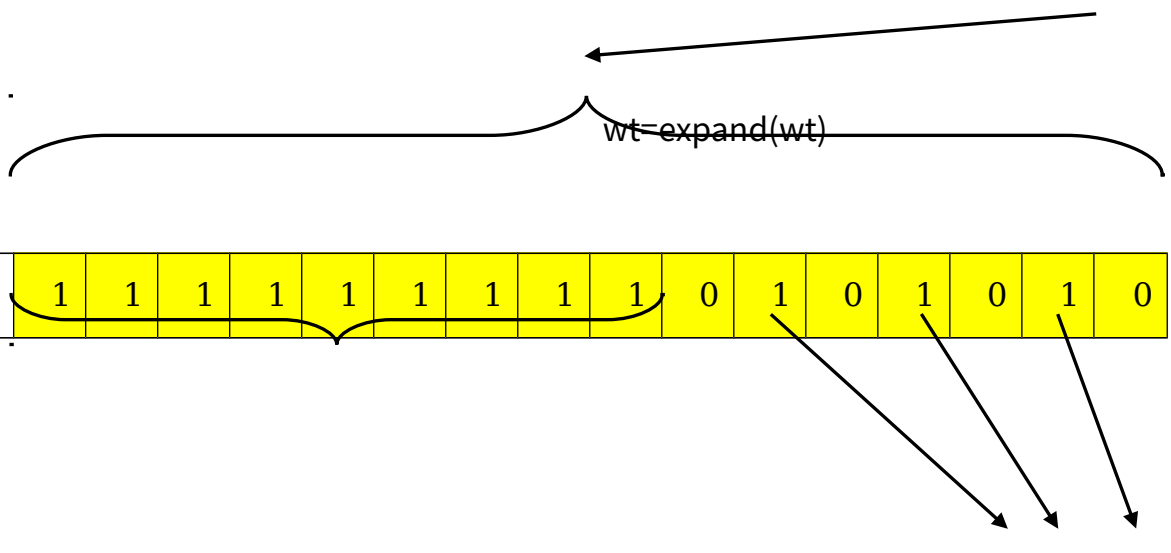
0x12,0x13,0x14,0x15};

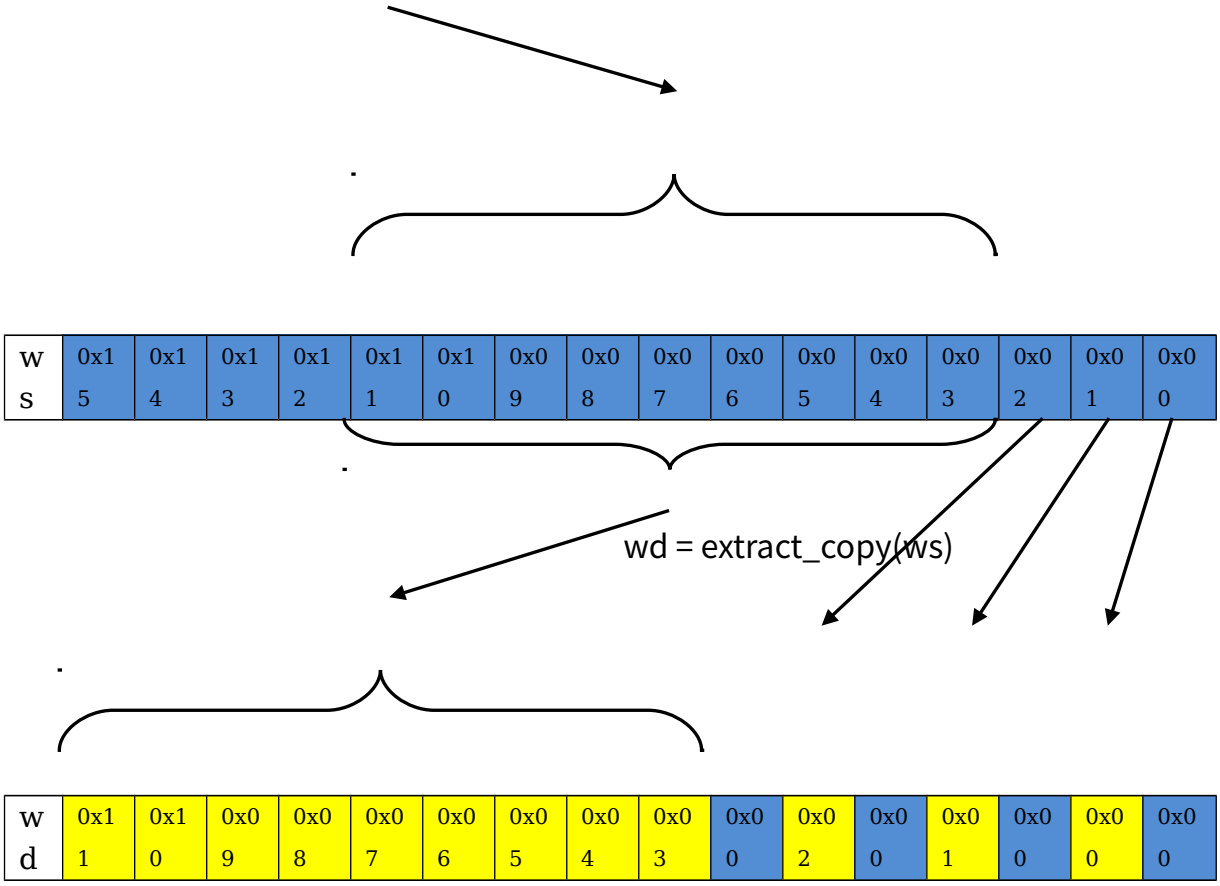
step1:

ws	0x15	0x14	0x13	0x12	0x11	0x10	0x09	0x08	0x07	0x06	0x05	0x04	0x03	0x02	0x01	0x00
wt	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0xff	0x00

step2:

wt	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0xff	x0aa





2.2.44. BMU.V

2.2.44.1. v16i8 __builtin_msa_bmu_v (v16i8,v16i8,v16i8)

Specific

Vector bit Mask Update.

v16i8 wd = __builtin_msa_bmu_v (v16i8 wd,v16i8 ws,v16i8 wt);

Return Value

wd :=bmu_v(ws,wd,wt[0]);

Requirements

Header:#include<msa.h>

test_bmu_v:

```
signed char a[16] = {0xfa,0xff,0x00,0x00,
                     0x00,0x00,0x00,0x00,
                     0x00,0x00,0x00,0x00,
                     0x00,0x00,0x00,0x00};
signed char b[16] = {0x55,0xff,0xff,0x00,
                     0x00,0x00,0x00,0x00,
                     0x00,0x00,0x00,0x00,
                     0x00,0x00,0x00,0x00};

signed char c[16] = {7,0,0,0,
                     0,0,0,0,
                     0,0,0,0,
                     0,0,0,0};
v16i8 va,vb,vc;
va =  __builtin_msa_ld_b(a,0);
vb =  __builtin_msa_ld_b(b,0);
vc =  __builtin_msa_ld_b(c,0);
va = __builtin_msa_bmu_v (va,vb,vc);
__builtin_msa_st_b(va,a,0);
```

Result:

```
a[16] = {0x52,0xff,0x00,0x00,
```

0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00};

Graphic example:

wd[16] = {0xfa,0xff,0x00,0x00,

0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00};

ws[16] = {0x55,0xff,0xff,0x00,

0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00};

wt[16] = {7,0,0,0,

0,0,0,0,

0,0,0,0,

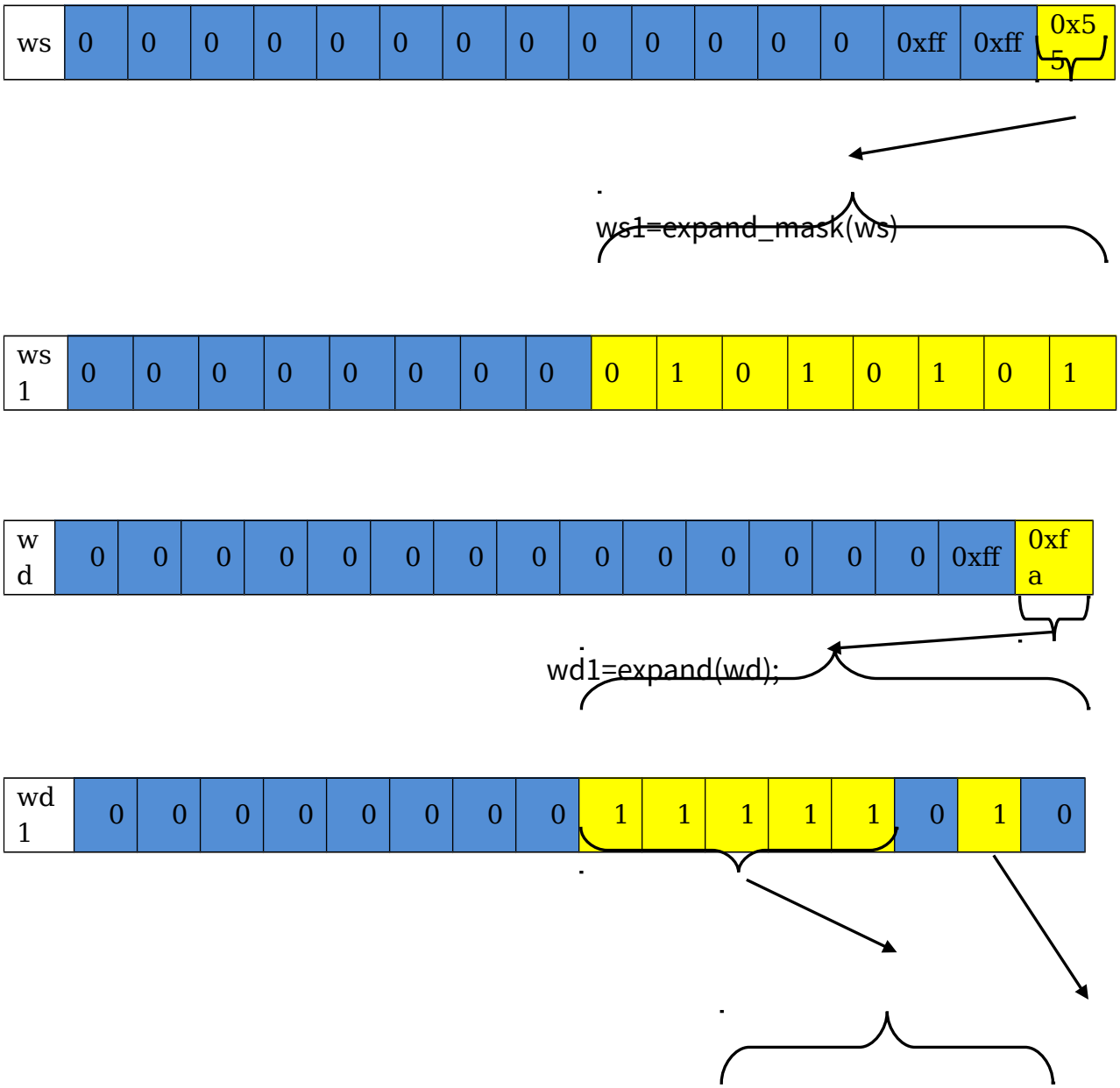
0,0,0,0};

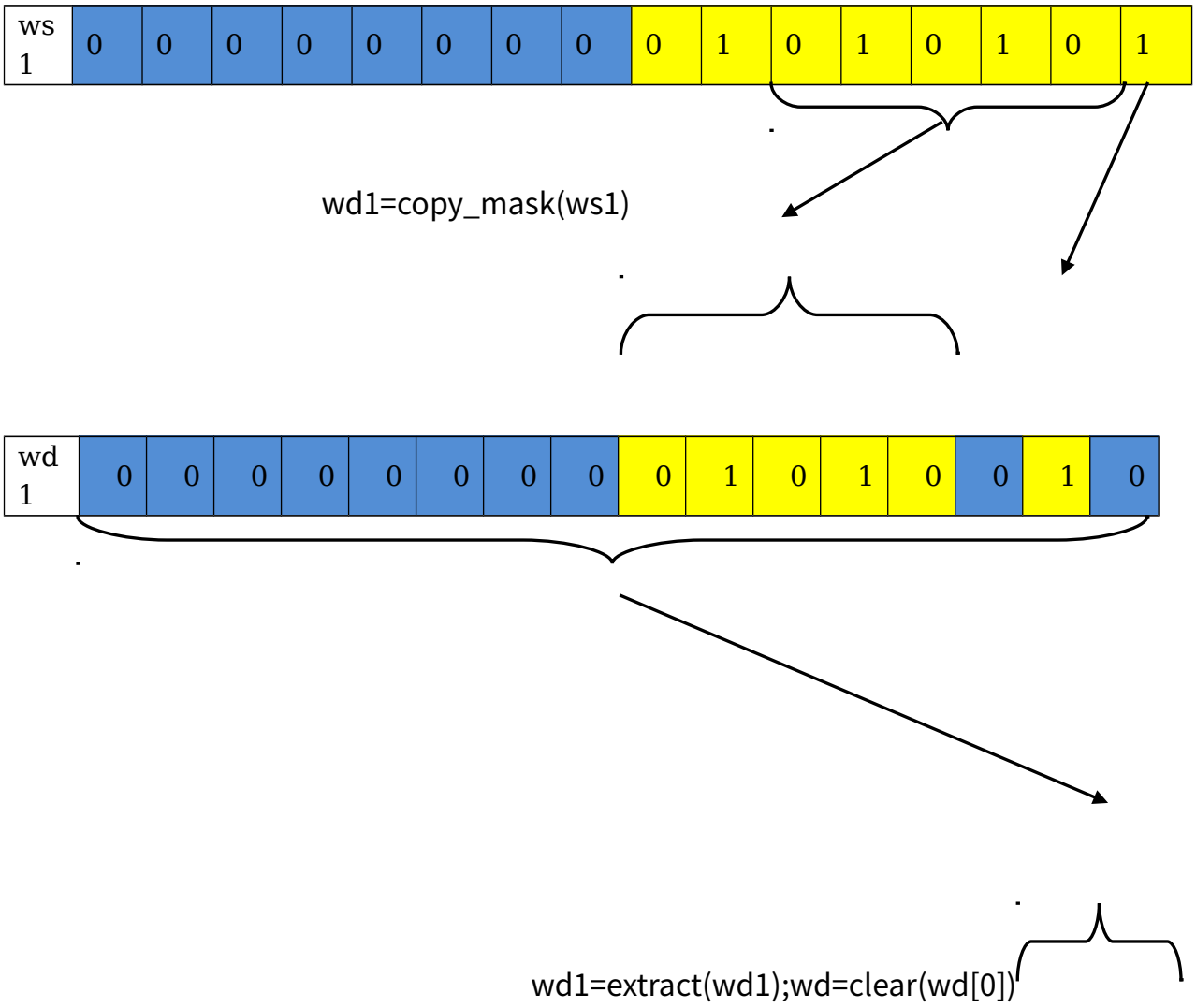
step1:

wd	0	0	0	0	0	0	0	0	0	0	0	0	0	0xff	0xfa
ws	0	0	0	0	0	0	0	0	0	0	0	0	0	0xff	0xff	0x55
wt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7

step2:

t=7;


ws
0 0 0 0 0 0 0 0 0 0 0 0 0 0xff 0xff 0x55
ws1=expand_mask(ws)
ws1
0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1
wd
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0xff 0xfa
wd1=expand(wd);
wd1
0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0


ws1
0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1
wd1=copy_mask(ws1)
wd1
0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0
wd1=extract(wd1);wd=clear(wd[0])
wd1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0x52
wd
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0xff 0x00
wd=wd|wd1
wd
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0xff 0x52

2.3. Floating-Point Arithmetic

2.3.1. FADD.df

2.3.1.1. v4f32 __builtin_msa_fadd_w(v4f32,v4f32)

Specific

Vector Floating-Point Addition.

v4f32 wd = __builtin_msa_fadd_w(v4f32ws,v4f32 wt);

Return Value

wd[0] := ws[0] + wt[0];

wd[1] := ws[1] + wt[1];

wd[2] := ws[2] + wt[2];

wd[3] := ws[3] + wt[3];

Requirements

Header:#include<msa.h>

2.3.1.2. v2f64 __builtin_msa_fadd_d(v2f64,v2f64)

Specific

Vector Floating-Point Addition.

v2f64 wd = __builtin_msa_fadd_d(v2f64 ws,v2f64 wt);

Return Value

wd[0] := ws[0] + wt[0];

wd[1] := ws[1] + wt[1];

Requirements

Header:#include<msa.h>

2.3.2. FDIV.df

2.3.2.1. v4f32 __builtin_msa_fdiv_w(v4f32,v4f32)

Specific

Vector Floating-Point Division.

v4f32 wd = __builtin_msa_fdiv_w(v4f32 ws,v4f32 wt);

Return Value

wd[0] := ws[0] / wt[0];

wd[1] := ws[1] / wt[1];

wd[2] := ws[2] / wt[2];

wd[3] := ws[3] / wt[3];

Requirements

Header:#include<msa.h>

2.3.2.2. v2f64 __builtin_msa_fdiv_d(v2f64,v2f64)

Specific

Vector Floating-Point Division.

v2f64 wd = __builtin_msa_fdiv_d(v2f64 ws,v2f64 wt);

Return Value

wd[0] := ws[0] / wt[0];

wd[1] := ws[1] / wt[1];

Requirements

Header:#include<msa.h>

2.3.3. FEXP2.df

2.3.3.1. v4f32 __builtin_msa_fexp2_w(v4f32,v4i32)

Specific

Vector Floating-Point Base 2 Exponentiation.

v4f32 wd = __builtin_msa_fexp2_w(v4f32 ws,v4i32 wt);

Return Value

wd[0] := ws[0] * $2^{wt[0]}$;

wd[1] := ws[1] * $2^{wt[1]}$;

wd[2] := ws[2] * $2^{wt[2]}$;

wd[3] := ws[3] * $2^{wt[3]}$;

Requirements

Header:#include<msa.h>

2.3.3.2. v2f64 __builtin_msa_fexp2_d(v2f64,v2i64)

Specific

Vector Floating-Point Base 2 Exponentiation.

v2f64 wd = __builtin_msa_fexp2_d(v2f64 ws,v2i64 wt);

Return Value

wd[0] := ws[0] * $2^{wt[0]}$;

wd[1] := ws[1] * $2^{wt[1]}$;

Requirements

Header:#include<msa.h>

2.3.4. FLOG2.df

2.3.4.1. v4f32 __builtin_msa_flog2_w(v4f32)

Specific

Vector Floating-Point Base 2 Logarithm.

v4f32 wd = __builtin_msa_flog2_w(v4f32 ws);

Return Value

wd[0] := log(ws[0]);

wd[1] := log(ws[1]);

wd[2] := log(ws[2]);

wd[3] := log(ws[3]);

Requirements

Header:#include<msa.h>

2.3.4.2. v2f64 __builtin_msa_flog2_d(v2f64)

Specific

Vector Floating-Point Base 2 Logarithm.

v2f64 wd = __builtin_msa_flog2_d(v2f64 ws);

Return Value

wd[0] := log(ws[0]);

wd[1] := log(ws[1]);

Requirements

Header:#include<msa.h>

2.3.5. FMADD.df

2.3.5.1. v4f32 __builtin_msa_fmadd_w(v4f32,v4f32,v4f32)

Specific

Vector Floating-Point Multiply-Add.

v4f32 wd = __builtin_msa_fmadd_w(v4f32 wd,v4f32 ws,v4f32 wt);

Return Value

wd[0] := wd[0] + ws[0]*wt[0];

wd[1] := wd[1] + ws[1]*wt[1];

wd[2] := wd[2] + ws[2]*wt[2];

wd[3] := wd[3] + ws[3]*wt[3];

Requirements

Header:#include<msa.h>

2.3.5.2. v2f64 __builtin_msa_fmadd_d(v2f64,v2f64,v2f64)

Specific

Vector Floating-Point Multiply-Add.

v2f64 wd = __builtin_msa_fmadd_d(v2f64 wd,v2f64 ws,v2f64 wt);

Return Value

wd[0] := wd[0] + ws[0]*wt[0];

wd[1] := wd[1] + ws[1]*wt[1];

Requirements

Header:#include<msa.h>

2.3.6. FMSUB.df

2.3.6.1. v4f32 __builtin_msa_fmsub_w(v4f32,v4f32,v4f32)

Specific

Vector Floating-Point Multiply-Sub.

v4f32 wd = __builtin_msa_fmsub_w(v4f32 wd,v4f32 ws,v4f32 wt);

Return Value

wd[0] := wd[0] - ws[0]*wt[0];

wd[1] := wd[1] - ws[1]*wt[1];

wd[2] := wd[2] - ws[2]*wt[2];

wd[3] := wd[3] - ws[3]*wt[3];

Requirements

Header:#include<msa.h>

2.3.6.2. v2f64 __builtin_msa_fmsub_d(v2f64,v2f64,v2f64)

Specific

Vector Floating-Point Multiply-Sub.

v2f64 wd = __builtin_msa_fmsub_d(v2f64 wd,v2f64 ws,v2f64 wt);

Return Value

wd[0] := wd[0] - ws[0]*wt[0];

wd[1] := wd[1] - ws[1]*wt[1];

Requirements

Header:#include<msa.h>

2.3.7. FMAX.df

2.3.7.1. v4f32 __builtin_msa_fmax_w(v4f32,v4f32)

Specific

Vector Floating-Point Maximum.

v4f32 wd = __builtin_msa_fmax_w(v4f32 ws,v4f32 wt);

Return Value

wd[0] := (ws[0]>wt[0]) ? ws[0]:wt[0];

wd[1] := (ws[1]>wt[1]) ? ws[1]:wt[1];

wd[2] := (ws[2]>wt[2]) ? ws[2]:wt[2];

wd[3] := (ws[3]>wt[3]) ? ws[3]:wt[3];

Requirements

Header:#include<msa.h>

2.3.7.2. v2f64 __builtin_msa_fmax_d(v2f64,v2f64)

Specific

Vector Floating-Point Maximum.

v2f64 wd = __builtin_msa_fmax_d(v2f64 ws,v2f64 wt);

Return Value

wd[0] := (ws[0]>wt[0]) ? ws[0]:wt[0];

wd[1] := (ws[1]>wt[1]) ? ws[1]:wt[1];

Requirements

Header:#include<msa.h>

2.3.8. FMIN.df

2.3.8.1. v4f32 __builtin_msa_fmin_w(v4f32,v4f32)

Specific

Vector Floating-Point Minimum.

v4f32 wd = __builtin_msa_fmin_w(v4f32 ws,v4f32 wt);

Return Value

wd[0] := (ws[0]<wt[0]) ? ws[0] :wt[0];

wd[1] := (ws[1]<wt[1]) ? ws[1] :wt[1];

wd[2] := (ws[2]<wt[2]) ? ws[2] :wt[2];

wd[3] := (ws[3]<wt[3]) ? ws[3] :wt[3];

Requirements

Header:#include<msa.h>

2.3.8.2. v2f64 __builtin_msa_fmin_d(v2f64,v2f64)

Specific

Vector Floating-Point Minimum.

v2f64 wd = __builtin_msa_fmin_d(v2f64 ws,v2f64 wt);

Return Value

wd[0] := (ws[0]<wt[0]) ? ws[0] :wt[0];

wd[1] := (ws[1]<wt[1]) ? ws[1] :wt[1];

Requirements

Header:#include<msa.h>

2.3.9. FMAX_A.df

2.3.9.1. v4f32 __builtin_msa_fmax_a_w(v4f32,v4f32)

Specific

Vector Floating-Point Maximum Based on Absolute Values.

v4f32 wd = __builtin_msa_fmax_a_w(v4f32 ws,v4f32 wt);

Return Value

wd[0] := (fabs(ws[0] , 32) > fabs(wt[0] , 32)) ? ws[0] : wt[0];

wd[1] := (fabs(ws[1] , 32) > fabs(wt[1] , 32)) ? ws[1] : wt[1];

wd[2] := (fabs(ws[2] , 32) > fabs(wt[2] , 32)) ? ws[2] : wt[2];

wd[3] := (fabs(ws[3] , 32) > fabs(wt[3] , 32)) ? ws[3] : wt[3];

Requirements

Header:#include<msa.h>

2.3.9.2. v2f64 __builtin_msa_fmax_a_d(v2f64,v2f64)

Specific

Vector Floating-Point Maximum Based on Absolute Values.

v2f64 wd = __builtin_msa_fmax_a_d(v2f64 ws,v2f64 wt);

Return Value

wd[0] := (fabs(ws[0] , 64) > fabs(wt[0] , 64)) ? ws[0] : wt[0];

wd[1] := (fabs(ws[1] , 64) > fabs(wt[1] , 64)) ? ws[1] : wt[1];

Requirements

Header:#include<msa.h>

2.3.10. FMIN_A.df

2.3.10.1. v4f32 __builtin_msa_fmin_a_w(v4f32,v4f32)

Specific

Vector Floating-Point Minimum Based on Absolute Values.

v4f32 wd = __builtin_msa_fmin_a_w(v4f32 ws,v4f32 wt);

Return Value

wd[0] := (fabs(ws[0] , 32) < fabs(wt[0] , 32)) ? ws[0] : wt[0];

wd[1] := (fabs(ws[1] , 32) < fabs(wt[1] , 32)) ? ws[1] : wt[1];

```
wd[2] := (fabs(ws[2] , 32) < fabs(wt[2] , 32)) ? ws[2] : wt[2];
```

wd[3] := (fabs(ws[3] , 32) < fabs(wt[3] , 32)) ? ws[3] : wt[3];

Requirements

Header:#include<msa.h>

2.3.10.2. v2f64 __builtin_msa_fmin_a_d(v2f64,v2f64)

Specific

Vector Floating-Point Minimum Based on Absolute Values.

v2f64 wd = __builtin_msa_fmin_a_d(v2f64 ws,v2f64 wt);

Return Value

wd[0] := (fabs(ws[0] , 64) < fabs(wt[0] , 64)) ? ws[0] : wt[0];

wd[1] := (fabs(ws[1] , 64) < fabs(wt[1] , 64)) ? ws[1] : wt[1];

Requirements

Header:#include<msa.h>

2.3.11. FMUL.df

2.3.11.1. v4f32 __builtin_msa_fmul_w(v4f32,v4f32)

Specific

Vector Floating-Point Multiplication.

v4f32 wd = __builtin_msa_fmul_w(v4f32 ws,v4f32 wt);

Return Value

wd[0] := ws[0] * wt[0];

wd[1] := ws[1] * wt[1];

wd[2] := ws[2] * wt[2];

wd[3] := ws[3] * wt[3];

Requirements

Header:#include<msa.h>

2.3.11.2. v2f64 __builtin_msa_fmul_d(v2f64,v2f64)

Specific

Vector Floating-Point Multiplication.

v2f64 wd = __builtin_msa_fmul_d(v2f64 ws,v2f64 wt);

Return Value

wd[0] := ws[0] * wt[0];

wd[1] := ws[1] * wt[1];

Requirements

Header:#include<msa.h>

2.3.12. FRCP.df

2.3.12.1. v4f32 __builtin_msa_frcp_w(v4f32)

Specific

Vector Approximate Floating-Point Reciprocal.

v4f32 wd = __builtin_msa_frcp_w(v4f32 ws);

Return Value

wd[0] := 1.0 / ws[0];

wd[1] := 1.0 / ws[1];

wd[2] := 1.0 / ws[2];

wd[3] := 1.0 / ws[3];

Requirements

Header:#include<msa.h>

2.3.12.2. v2f64 __builtin_msa_frcp_d(v2f64)

Specific

Vector Approximate Floating-Point Reciprocal.

v2f64 wd = __builtin_msa_frcp_d(v2f64 ws);

Return Value

wd[0] := 1.0 / ws[0];

wd[1] := 1.0 / ws[1];

Requirements

Header:#include<msa.h>

2.3.13. FRINT.df

2.3.13.1. v4f32 __builtin_msa_frint_w(v4f32)

Specific

Vector Floating-Point Round to Integer.

v4f32 wd = __builtin_msa_frint_w(v4f32 ws);

Return Value

wd[0] := single_rto_sint(ws[0]);

wd[1] := single_rto_sint(ws[1]);

wd[2] := single_rto_sint(ws[2]);

wd[3] := single_rto_sint(ws[3]);

Requirements

Header:#include<msa.h>

2.3.13.2. v2f64 __builtin_msa_frint_d(v2f64)

Specific

Vector Floating-Point Round to Integer.

v2f64 wd = __builtin_msa_frint_d(v2f64 ws);

Return Value

wd[0] := single_rto_sint(ws[0]);

wd[1] := single_rto_sint(ws[1]);

Requirements

Header:#include<msa.h>

2.3.14. FRSQRT.df

2.3.14.1. v4f32 __builtin_msa_frsqrt_w(v4f32)

Specific

Vector Approximate Floating-Point Reciprocal of Square Root.

v4f32 wd = __builtin_msa_frsqrt_w(v4f32 ws);

Return Value

wd[0] := 1.0 / sqrt(ws[0]);

wd[1] := 1.0 / sqrt(ws[1]);

wd[2] := 1.0 / sqrt(ws[2]);

wd[3] := 1.0 / sqrt(ws[3]);

Requirements

Header:#include<msa.h>

2.3.14.2. v2f64 __builtin_msa_frsqrt_d(v2f64)

Specific

Vector Approximate Floating-Point Reciprocal of Square Root.

v2f64 wd = __builtin_msa_frsqrt_d(v2f64 ws);

Return Value

wd[0] := 1.0 / sqrt(ws[0]);

wd[1] := 1.0 / sqrt(ws[1]);

Requirements

Header:#include<msa.h>

2.3.15. FSQRT.df

2.3.15.1. v4f32 __builtin_msa_fsqrt_w(v4f32)

Specific

Vector Floating-Point Square Root.

v4f32 wd = __builtin_msa_fsqrt_w(v4f32 ws);

Return Value

wd[0] := sqrt(ws[0]);

wd[1] := sqrt(ws[1]);

wd[2] := sqrt(ws[2]);

wd[3] := sqrt(ws[3]);

Requirements

Header:#include<msa.h>

2.3.15.2. v2f64 __builtin_msa_fsqrt_d(v2f64)

Specific

Vector Floating-Point Square Root.

v2f64 wd = __builtin_msa_fsqrt_d(v2f64 ws);

Return Value

wd[0] := sqrt(ws[0]);

wd[1] := sqrt(ws[1]);

Requirements

Header:#include<msa.h>

2.3.16. FSUB.df

2.3.16.1. v4f32 __builtin_msa_fsub_w(v4f32, v4f32)

Specific

Vector Floating-Point Subtraction.

v4f32 wd =__builtin_msa_fsub_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] := ws[0] - wt[0];

wd[1] := ws[1] - wt[1];

wd[2] := ws[2] - wt[2];

wd[3] := ws[3] - wt[3];

Requirements

Header:#include<msa.h>

2.3.16.2. v2f64 __builtin_msa_fsub_d(v2f64, v2f64)

Specific

Vector Floating-Point Subtraction.

v2f64 wd = __builtin_msa_fsub_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] := ws[0] - wt[0];

wd[1] := ws[1] - wt[1];

Requirements

Header:#include<msa.h>

2.3.17. FCMUL.W

2.3.17.1. v4f32 __builtin_msa_fcmul_w (v4f32,v4f32)

Specific

Vecoter Floating-Point Complex Multiplication.

v4f32 wd = __builtin_msa_fcmul_w (v4f32 ws,v4f32 wt);

Return Value

wd[0] := (v2f64)complex_mul((v2f64)ws[0] , (v2f64)wt[0]);

wd[1] := (v2f64)complex_mul((v2f64)ws[1], (v2f64)wt[1]);

Requirements

Header:#include<msa.h>

2.4. Floating-Point Non Arithmetic

2.4.1. FCLASS.df

2.4.1.1. v4f32 __builtin_msa_fclass_w(v4f32)

Specific

Vector Floating-Point Class Mask.

v4f32 wd = __builtin_msa_fclass_w(v4f32 ws);

Return Value

wd[0] := fclass(ws[0],32);

wd[1] := fclass(ws[1],32);

wd[2] := fclass(ws[2],32);

wd[3] := fclass(ws[3],32);

Requirements

Header:#include<msa.h>

2.4.1.2. v2f64 __builtin_msa_fclass_d(v2f64)

Specific

Vector Floating-Point Class Mask.

v2f64 wd = __builtin_msa_fclass_d(v2f64 ws);

Return Value

wd[0] := fclass(ws[0],64);

wd[1] := fclass(ws[1],64);

Requirements

Header:#include<msa.h>

2.5. Floating-Point Compare

2.5.1. FCAF.df

2.5.1.1. v4i32 __builtin_msa_fcaf_w(v4f32, v4f32)

Specific

Vector Floating-Point Quiet Compare Always False.

v4i32 wd = __builtin_msa_fcaf_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] := quiet_FALSE(ws[0],wt[0],32);

wd[1] := quiet_FALSE(ws[1],wt[1],32);

wd[2] := quiet_FALSE(ws[2],wt[2],32);

wd[3] := quiet_FALSE(ws[3],wt[3],32);

Requirements

Header:#include<msa.h>

2.5.1.2. v2i64 __builtin_msa_fcaf_d(v2f64, v2f64)

Specific

Vector Floating-Point Quiet Compare Always False.

v2i64 wd = __builtin_msa_fcaf_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] := quiet_FALSE(ws[0],wt[0],64);

wd[1] := quiet_FALSE(ws[1],wt[1],64);

Requirements

Header:#include<msa.h>

2.5.2. FCUN.df

2.5.2.1. v4i32 __builtin_msa_fcun_w(v4f32, v4f32)

Specific

Vector Floating-Point Quite Compare Unordered.

v4i32 wd = __builtin_msa_fcun_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] := ((ws[0]==NAN)||(wt[0]==NAN)) ? one32:zero32;

wd[1] := ((ws[1]==NAN)||(wt[1]==NAN)) ? one32:zero32;

wd[2] := ((ws[2]==NAN)||(wt[2]==NAN)) ? one32:zero32;

wd[3] := ((ws[3]==NAN)||(wt[3]==NAN)) ? one32:zero32;

Requirements

Header:#include<msa.h>

2.5.2.2. v2i64 __builtin_msa_fcun_d(v2f64, v2f64)

Specific

Vector Floating-Point Quite Compare Unordered.

v2i64 wd = __builtin_msa_fcun_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] := ((ws[0]==NAN)||(wt[0]==NAN)) ? one64:zero64;

wd[1] := ((ws[1]==NAN)||(wt[1]==NAN)) ? one64:zero64;

Requirements

Header:#include<msa.h>

2.5.3. FCOR.df

2.5.3.1. v4i32 __builtin_msa_fcor_w(v4f32, v4f32)

Specific

Vector Floating-Point Quiet Compare Ordered.

v4i32 wd = __builtin_msa_fcor_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] := ((ws[0]!=NAN)&& (wt[0]!=NAN)) ? one32:zero32

wd[1] := ((ws[1]!=NAN)&&(wt[1]!=NAN)) ? one32: zero32

wd[2] := ((ws[2]!=NAN)&&(wt[2]!=NAN)) ? one32: zero32

wd[3] := ((ws[3]!=NAN)&&(wt[3]!=NAN)) ? one32: zero32

Requirements

Header:#include<msa.h>

2.5.3.2. v2i64 __builtin_msa_fcor_d(v2f64, v2f64)

Specific

Vector Floating-Point Quiet Compare Ordered.

v2i64 wd = __builtin_msa_fcor_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] := ((ws[0]!=NAN)||(wt[0]!=NAN)) ? one64: zero64

wd[1] := ((ws[1]!=NAN)||(wt[1]!=NAN)) ? one64: zero64

Requirements

Header:#include<msa.h>

2.5.4. FCEQ.df

2.5.4.1. v4i32 __builtin_msa_fceq_w(v4f32, v4f32)

Specific

Vector Floating-Point Quiet Compare Equal.

v4i32 wd = __builtin_msa_fceq_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] :=((ws[0]==wt[0])&& ((ws[0]!=NAN)||(wt[0]!=NAN))) ? one32:zero32

wd[1] :=((ws[1]==wt[1])&&((ws[1]!=NAN)||(wt[1]!=NAN))) ? one32:zero32

wd[2] :=((ws[2]==wt[2])&&((ws[2]!=NAN)||(wt[2]!=NAN)))? one32:zero32

wd[3] :=((ws[3]==wt[3])&&((ws[3]!=NAN)||(wt[3]!=NAN))) ? one32:zero32

Requirements

Header:#include<msa.h>

2.5.4.2. v2i64 __builtin_msa_fceq_d(v2f64, v2f64)

Specific

Vector Floating-Point Quiet Compare Equal.

v2i64 wd = __builtin_msa_fceq_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] :=((ws[0]==wt[0]) &&((ws[0]!=NAN)|| (wt[0]!=NAN))) ? one64:zero64

wd[1] :=((ws[1]==wt[1]) &&((ws[1]!=NAN)||(wt[1]!=NAN))) ? one64:zero64

Requirements

Header:#include<msa.h>

2.5.5. FCUNE.df

2.5.5.1. v4i32 __builtin_msa_fcune_w(v4f32, v4f32)

Specific

Vector Floating-Point Quiet Compare Unordered or Not Equal.

v4i32 wd = __builtin_msa_fcune_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] := ((ws[0]!=wt[0]) || ((ws[0]==NAN)||(wt[0]==NAN)))?one32:zero32

wd[1] := ((ws[1]!=wt[1]) || ((ws[1]==NAN)||(wt[1]==NAN)))?one32:zero32

wd[2] := ((ws[2]!=wt[2]) || ((ws[2]==NAN)||(wt[2]==NAN)))?one32:zero32

wd[3] := ((ws[3]!=wt[3]) || ((ws[3]==NAN)||(wt[3]==NAN)))?one32:zero32

Requirements

Header:#include<msa.h>

2.5.5.2. v2i64 __builtin_msa_fcune_d(v2f64, v2f64)

Specific

Vector Floating-Point Quiet Compare Unordered or Not Equal.

v2i64 wd = __builtin_msa_fcune_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] := ((ws[0]!=wt[0]) || ((ws[0]==NAN)||(wt[0]==NAN)))?one64:zero64

wd[1] := ((ws[1]!=wt[1]) || ((ws[1]==NAN)||(wt[1]==NAN)))?one64:zero64

Requirements

Header:#include<msa.h>

2.5.6. FCUEQ.df

2.5.6.1. v4i32 __builtin_msa_fcueq_w(v4f32, v4f32)

Specific

Vector Floating-Point Quiet Compare Unordered or Equal.

v4i32 wd = __builtin_msa_fcueq_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] := ((ws[0]==wt[0]) || ((ws[0]==NAN)|| (wt[0]==NAN)))?one32:zero32

wd[1] := ((ws[1]==wt[1]) || ((ws[1]==NAN)||(wt[1]==NAN)))? one32: zero32

wd[2] := ((ws[2]==wt[2]) || ((ws[2]==NAN)||(wt[2]==NAN)))? one32: zero32

wd[3] := ((ws[3]==wt[3]) || ((ws[3]==NAN)||(wt[3]==NAN)))? one32: zero32

Requirements

Header:#include<msa.h>

2.5.6.2. v2i64 __builtin_msa_fcueq_d(v2f64, v2f64)

Specific

Vector Floating-Point Quiet Compare Unordered or Equal.

v2i64 wd = __builtin_msa_fcueq_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] := ((ws[0]==wt[0]) || ((ws[0]==NAN)|| (wt[0]==NAN)))?one64:zero64

wd[1] := ((ws[1]==wt[1]) || ((ws[1]==NAN)||(wt[1]==NAN)))? one64: zero64

Requirements

Header:#include<msa.h>

2.5.7. FCNE.df

2.5.7.1. v4i32 __builtin_msa_fcne_w(v4f32, v4f32)

Specific

Vector Floating-Point Quiet Compare Not Equal.

v4i32 wd = __builtin_msa_fcne_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] :=((ws[0]!=wt[0]) &&((ws[0]!=NAN)|| (wt[0]!=NAN)))?one32:zero32

wd[1] :=((ws[1]!=wt[1]) &&((ws[1]!=NAN)||(wt[1]!=NAN)))? one32:zero32

wd[2] :=((ws[2]!=wt[2]) &&((ws[2]!=NAN)||(wt[2]!=NAN)))? one32:zero32

wd[3] :=((ws[3]!=wt[3]) &&((ws[3]!=NAN)||(wt[3]!=NAN)))? one32:zero32

Requirements

Header:#include<msa.h>

2.5.7.2. v2i64 __builtin_msa_fcne_d(v2f64, v2f64)

Specific

Vector Floating-Point Quiet Compare Not Equal.

v2i64 wd = __builtin_msa_fcne_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] :=((ws[0]!=wt[0]) && ((ws[0]!=NAN)|| (wt[0]!=NAN)))?one64:zero64

wd[1] :=((ws[1]!=wt[1]) && ((ws[1]!=NAN)||(wt[1]!=NAN)))?one64:zero64

Requirements

Header:#include<msa.h>

2.5.8. FCLT.df

2.5.8.1. v4i32 __builtin_msa_fclt_w(v4f32, v4f32)

Specific

Vector Floating-Point Quiet Compare Less Than.

v4i32 wd = __builtin_msa_fclt_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] :=((ws[0]<wt[0]) || (ws[0]!=NAN))?one32:zero32

wd[1] :=((ws[1]<wt[1]) || (ws[1]!=NAN))?one32:zero32

wd[2] :=((ws[2]<wt[2]) || (ws[2]!=NAN))?one32:zero32

wd[3] :=((ws[3]<wt[3]) || (ws[3]!=NAN))?one32:zero32

Requirements

Header:#include<msa.h>

2.5.8.2. v2i64 __builtin_msa_fclt_d(v2f64, v2f64)

Specific

Vector Floating-Point Quiet Compare Less Than.

v2i64 wd = __builtin_msa_fclt_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] :=((ws[0]<wt[0]) || (ws[0]!=NAN))?one64:zero64

wd[1] :=((ws[1]<wt[1]) || (ws[0]!=NAN))?one64:zero64

Requirements

Header:#include<msa.h>

2.5.9. FCULT.df

2.5.9.1. v4i32 __builtin_msa_fcult_w(v4f32, v4f32)

Specific

Vector Floating-Point Quiet Compare Unordered or Less Than.

v4i32 wd = __builtin_msa_fcult_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] := ((ws[0]<wt[0]) || (ws[0]==NAN))?one32:zero32

wd[1] := ((ws[1]<wt[1]) || (ws[1]==NAN))?one32:zero32

wd[2] := ((ws[2]<wt[2]) || (ws[2]==NAN))?one32:zero32

wd[3] := ((ws[3]<wt[3]) || (ws[3]= =NAN))?one32:zero32

Requirements

Header:#include<msa.h>

2.5.9.2. v2i64 __builtin_msa_fcult_d(v2f64, v2f64)

Specific

Vector Floating-Point Quiet Compare Unordered or Less Than.

v2i64 wd = __builtin_msa_fcult_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] := ((ws[0]<wt[0]) || (ws[0]==NAN))?one64:zero64

wd[1] := ((ws[1]<wt[1]) || (ws[1]==NAN))?one64:zero64

Requirements

Header:#include<msa.h>

2.5.10. FCLE.df

2.5.10.1. v4i32 __builtin_msa_fcle_w(v4f32, v4f32)

Specific

Vector Floating-Point Quiet Compare Less or Equal.

v4i32 wd = __builtin_msa_fcle_w(v4f32 ws, v4f32 wt)

Return Value

wd[0] := ((ws[0]<=wt[0])&& (ws[0]!=NAN))?one32:zero32

wd[1] := ((ws[1]<=wt[1]) && (ws[1]!=NAN))?one32:zero32

wd[2] := ((ws[2]<=wt[2]) && (ws[2]!=NAN))?one32:zero32

wd[3] := ((ws[3]<=wt[3]) && (ws[3]! =NAN))?one32:zero32

Requirements

Header:#include<msa.h>

2.5.10.2. v2i64 __builtin_msa_fcle_d(v2f64, v2f64)

Specific

Vector Floating-Point Quiet Compare Less or Equal.

v2i64 wd = __builtin_msa_fcle_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] := ((ws[0]<=wt[0]) && (ws[0]!=NAN))?one64:zero64

wd[1] := ((ws[1]<=wt[1]) && (ws[1]!=NAN))?one64:zero64

Requirements

Header:#include<msa.h>

2.5.11. FCULE.df

2.5.11.1. v4i32 __builtin_msa_fcule_w(v4f32, v4f32)

Specific

Vector Floating-Point Quiet Compare Unordered or Less or Equal.

v4i32 wd = __builtin_msa_fcule_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] := ((ws[0]<=wt[0]) || (ws[0]==NAN))?one32:zero32

wd[1] := ((ws[1]<=wt[1]) || (ws[1]==NAN))?one32:zero32

wd[2] := ((ws[2]<=wt[2]) || (ws[2]==NAN))?one32:zero32

wd[3] := ((ws[3]<=wt[3]) || (ws[3]= =NAN))?one32:zero32

Requirements

Header:#include<msa.h>

2.5.11.2. v2i64 __builtin_msa_fcule_d(v2f64, v2f64)

Specific

Vector Floating-Point Quiet Compare Unordered or Less or Equal.

v2i64 wd = __builtin_msa_fcule_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] := ((ws[0]<=wt[0]) || (ws[0]==NAN))?one64:zero64

wd[1] := ((ws[1]<=wt[1]) || (ws[1]==NAN))?one64:zero64

Requirements

Header:#include<msa.h>

2.5.12. FSAF.df

2.5.12.1. v4i32 __builtin_msa_fsaf_w(v4f32, v4f32)

Specific

Vector Floating-Point Signaling Compare Always False.

v4i32 wd = __builtin_msa_fsaf_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] := signaling_FALSE(ws[0],wt[0],32);

wd[1] := signaling_FALSE(ws[1],wt[1],32);

wd[2] := signaling_FALSE(ws[2],wt[2],32);

wd[3] := signaling_FALSE(ws[3],wt[3],32);

Requirements

Header:#include<msa.h>

2.5.12.2. v2i64 __builtin_msa_fsaf_d(v2f64, v2f64)

Specific

Vector Floating-Point Signaling Compare Always False.

v2i64 wd = __builtin_msa_fsaf_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] := signaling_FALSE(ws[0],wt[0],64);

wd[1] := signaling_FALSE(ws[1],wt[1],64);

Requirements

Header:#include<msa.h>

2.5.13. FSUN.df

2.5.13.1. v4i32 __builtin_msa_fsun_w(v4f32, v4f32)

Specific

Vector Floating-Point Signaling Compare Unordered.

v4i32 wd = __builtin_msa_fsun_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] := ((ws[0]==NAN)||(wt[0]==NAN)) ? one32:zero32;

wd[1] := ((ws[1]==NAN)||(wt[1]==NAN)) ? one32:zero32;

wd[2] := ((ws[2]==NAN)||(wt[2]==NAN)) ? one32:zero32;

wd[3] := ((ws[3]==NAN)||(wt[3]==NAN)) ? one32:zero32;

Requirements

Header:#include<msa.h>

2.5.13.2. v2i64 __builtin_msa_fsun_d(v2f64, v2f64)

Specific

Vector Floating-Point Signaling Compare Unordered.

v2i64 wd = __builtin_msa_fsun_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] := ((ws[0]==NAN)||(wt[0]==NAN)) ? one64:zero64;

wd[1] := ((ws[1]==NAN)||(wt[1]==NAN)) ? one64:zero64;

Requirements

Header:#include<msa.h>

2.5.14. FSOR.df

2.5.14.1. v4i32 __builtin_msa_fsor_w(v4f32, v4f32)

Specific

Vector Floating-Point Signaling Compare Ordered.

v4i32 wd = __builtin_msa_fsor_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] := ((ws[0]!=NAN)&&(wt[0]!=NAN)) ? one32:zero32

wd[1] := ((ws[1]!=NAN)&&(wt[1]!=NAN)) ? one32:zero32

wd[2] := ((ws[2]!=NAN)&&(wt[2]!=NAN)) ? one32:zero32

wd[3] := ((ws[3]!=NAN)&&(wt[3]!=NAN)) ? one32:zero32

Requirements

Header:#include<msa.h>

2.5.14.2. v2i64 __builtin_msa_fsor_d(v2f64, v2f64)

Specific

Vector Floating-Point Signaling Compare Ordered.

v2i64 wd = __builtin_msa_fsor_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] := ((ws[0]!=NAN)&&(wt[0]!=NAN)) ? one64:zero64

wd[1] := ((ws[1]!=NAN)&&(wt[1]!=NAN)) ? one64:zero64

Requirements

Header:#include<msa.h>

2.5.15. FSEQ.df

2.5.15.1. v4i32 __builtin_msa_fseq_w(v4f32, v4f32)

Specific

Vector Floating-Point Signaling Compare Equal.

v4i32 wd = __builtin_msa_fseq_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] := (ws[0]==wt[0]) ? one32:zero32

wd[1] := (ws[1]==wt[1])? one32:zero32

wd[2] := (ws[2]==wt[2])? one32:zero32

wd[3] := (ws[3]==wt[3]) ? one32:zero32

Requirements

Header:#include<msa.h>

2.5.15.2. v2i64 __builtin_msa_fseq_d(v2f64, v2f64)

Specific

Vector Floating-Point Signaling Compare Equal.

v2i64 wd = __builtin_msa_fseq_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] :=(ws[0]==wt[0]) ? one64:zero64

wd[1] :=(ws[1]==wt[1]) ? one64:zero64

Requirements

Header:#include<msa.h>

2.5.16. FSUNE.df

2.5.16.1. v4i32 __builtin_msa_fsune_w(v4f32, v4f32)

Specific

Vector Floating-Point Signaling Compare Unordered or Not Equal.

v4i32 wd = __builtin_msa_fsune_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] := ((ws[0]!=wt[0]) || ((ws[0]==NAN)||(wt[0]==NAN)))?one32:zero32

wd[1] := ((ws[1]!=wt[1]) || ((ws[1]==NAN)||(wt[1]==NAN)))?one32:zero32

wd[2] := ((ws[2]!=wt[2]) || ((ws[2]==NAN)||(wt[2]==NAN)))?one32:zero32

wd[3] := ((ws[3]!=wt[3]) || ((ws[3]==NAN)||(wt[3]==NAN)))?one32:zero32

Requirements

Header:#include<msa.h>

2.5.16.2. v2i64 __builtin_msa_fsune_d(v2f64, v2f64)

Specific

Vector Floating-Point Signaling Compare Unordered or Not Equal.

v2i64 wd = __builtin_msa_fsune_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] := ((ws[0]!=wt[0]) || ((ws[0]==NAN)||(wt[0]==NAN)))?one64:zero64

wd[1] := ((ws[1]!=wt[1]) || ((ws[1]==NAN)||(wt[1]==NAN)))?one64:zero64

Requirements

Header:#include<msa.h>

2.5.17. FSUEQ.df

2.5.17.1. v4i32 __builtin_msa_fsueq_w(v4f32, v4f32)

Specific

Vector Floating-Point Signaling Compare Unordered or Equal.

v4i32 wd = __builtin_msa_fsueq_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] := ((ws[0]==wt[0]) || ((ws[0]==NAN)||(wt[0]==NAN)))?one32:zero32

wd[1] := ((ws[1]==wt[1]) || ((ws[1]==NAN)||(wt[1]==NAN)))?one32:zero32

wd[2] := ((ws[2]==wt[2]) || ((ws[2]==NAN)||(wt[2]==NAN)))?one32:zero32

wd[3] := ((ws[3]==wt[3]) || ((ws[3]==NAN)||(wt[3]==NAN)))?one32:zero32

Requirements

Header:#include<msa.h>

2.5.17.2. v2i64 __builtin_msa_fsueq_d(v2f64, v2f64)

Specific

Vector Floating-Point Signaling Compare Unordered or Equal.

v2i64 wd = __builtin_msa_fsueq_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] := ((ws[0]==wt[0]) || ((ws[0]==NAN)||(wt[0]==NAN)))?one64:zero64

wd[1] := ((ws[1]==wt[1]) || ((ws[1]==NAN)||(wt[1]==NAN)))?one64:zero64

Requirements

Header:#include<msa.h>

2.5.18. FSNE.df

2.5.18.1. v4i32 __builtin_msa_fsne_w(v4f32, v4f32)

Specific

Vector Floating-Point Signaling Compare Not Equal.

v4i32 wd = __builtin_msa_fsne_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] :=(ws[0]!=wt[0])?one32:zero32

wd[1] :=(ws[1]!=wt[1])?one32:zero32

wd[2] :=(ws[2]!=wt[2])?one32:zero32

wd[3] :=(ws[3]!=wt[3])?one32:zero32

Requirements

Header:#include<msa.h>

2.5.18.2. v2i64 __builtin_msa_fsne_d(v2f64, v2f64)

Specific

Vector Floating-Point Signaling Compare Not Equal.

v2i64 wd = __builtin_msa_fsne_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] :=(ws[0]!=wt[0])?one64:zero64

wd[1] :=(ws[1]!=wt[1])?one64:zero64

Requirements

Header:#include<msa.h>

2.5.19. FSLT.df

2.5.19.1. v4i32 __builtin_msa_fslt_w(v4f32, v4f32)

Specific

Vector Floating-Point Signaling Compare Less Than.

v4i32 wd = __builtin_msa_fslt_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] :=(ws[0]<wt[0])?one32:zero32

wd[1] :=(ws[1]<wt[1])?one32:zero32

wd[2] :=(ws[2]<wt[2])?one32:zero32

wd[3] :=(ws[3]<wt[3])?one32:zero32

Requirements

Header:#include<msa.h>

2.5.19.2. v2i64 __builtin_msa_fslt_d(v2f64, v2f64)

Specific

Vector Floating-Point Signaling Compare Less Than.

v2i64 wd = __builtin_msa_fslt_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] :=(ws[0]<wt[0])?one64:zero64

wd[1] :=(ws[1]<wt[1])?one64:zero64

Requirements

Header:#include<msa.h>

2.5.20. FSULT.df

2.5.20.1. v4i32 __builtin_msa_fsult_w(v4f32, v4f32)

Specific

Vector Floating-Point Signaling Compare Unordered or Less Than.

v4i32 wd = __builtin_msa_fsult_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] := ((ws[0]<wt[0]) || ((ws[0]==NAN)||(wt[0]==NAN)))?one32:zero32

wd[1] := ((ws[1]<wt[1]) || ((ws[1]==NAN)||(wt[1]==NAN)))?one32:zero32

wd[2] := ((ws[2]<wt[2]) || ((ws[2]==NAN)||(wt[2]==NAN)))?one32:zero32

wd[3] := ((ws[3]<wt[3]) || ((ws[3]= =NAN)||(wt[3]==NAN)))?one32:zero32

Requirements

Header:#include<msa.h>

2.5.20.2. v2i64 __builtin_msa_fsult_d(v2f64, v2f64)

Specific

Vector Floating-Point Signaling Compare Unordered or Less Than.

v2i64 wd = __builtin_msa_fsult_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] := ((ws[0]<wt[0]) || ((ws[0]==NAN)||(wt[0]==NAN)))?one64:zero64

wd[1] := ((ws[1]<wt[1]) || ((ws[1]==NAN)||(wt[1]==NAN)))?one64:zero64

Requirements

Header:#include<msa.h>

2.5.21. FSLE.df

2.5.21.1. v4i32 __builtin_msa_fsle_w(v4f32, v4f32)

Specific

Vector Floating-Point Signaling Compare Less or Equal.

v4i32 wd = __builtin_msa_fsle_w(v4f32 ws, v4f32 wt)

Return Value

wd[0] := (ws[0]<=wt[0])?one32:zero32

wd[1] := (ws[1]<=wt[1])?one32:zero32

wd[2] := (ws[2]<=wt[2])?one32:zero32

wd[3] := (ws[3]<=wt[3])?one32:zero32

Requirements

Header:#include<msa.h>

2.5.21.2. v2i64 __builtin_msa_fsle_d(v2f64, v2f64)

Specific

Vector Floating-Point Signaling Compare Less or Equal.

v2i64 wd = __builtin_msa_fsle_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] := (ws[0]<=wt[0])?one64:zero64

wd[1] := (ws[1]<=wt[1])?one64:zero64

Requirements

Header:#include<msa.h>

2.5.22. FSULE.df

2.5.22.1. v4i32 __builtin_msa_fsule_w(v4f32, v4f32)

Specific

Vector Floating-Point Signaling Compare Unordered or Less or Equal.

v4i32 wd = __builtin_msa_fsule_w(v4f32 ws, v4f32 wt);

Return Value

wd[0] := ((ws[0]<=wt[0]) || ((ws[0]==NAN)||(wt[0]==NAN)))?one32:zero32

wd[1] := ((ws[1]<=wt[1]) || ((ws[1]==NAN)||(wt[1]==NAN)))?one32:zero32

wd[2] := ((ws[2]<=wt[2]) || ((ws[2]==NAN)||(wt[2]==NAN)))?one32:zero32

wd[3] := ((ws[3]<=wt[3]) || ((ws[3]= =NAN)||(wt[3]==NAN)))?one32:zero32

Requirements

Header:#include<msa.h>

2.5.22.2. v2i64 __builtin_msa_fsule_d(v2f64, v2f64)

Specific

Vector Floating-Point Signaling Compare Unordered or Less or Equal.

v2i64 wd = __builtin_msa_fsule_d(v2f64 ws, v2f64 wt);

Return Value

wd[0] := ((ws[0]<=wt[0]) || ((ws[0]==NAN)||(wt[0]==NAN)))?one64:zero64

wd[1] := ((ws[1]<=wt[1]) || ((ws[1]==NAN)||(wt[1]==NAN)))?one64:zero64

Requirements

Header:#include<msa.h>

2.6. Floating-Point Conversion

2.6.1. FEXDO.df

2.6.1.1. v8i16 __builtin_msa_fexdo_h(v4f32, v4f32)

Specific

Vector Floating-Point Down-Convert Interchange Format.

v8i16 wd = __builtin_msa_fexdo_h(v4f32 ws, v4f32 wt);

Return Value

wd[0] := (f16)wt[0];

wd[1] := (f16)ws[0];

wd[2] := (f16)wt[1];

wd[3] := (f16)ws[1];

.....

wd[6] := (f16)wt[3];

wd[7] := (f16)ws[3];

Requirements

Header:#include<msa.h>

2.6.1.2. v4f32 __builtin_msa_fexdo_w(v2f64, v2f64)

Specific

Vector Floating-Point Down-Convert Interchange Format.

v4f32 wd = __builtin_msa_fexdo_w(v2f64 ws, v2f64 wt);

Return Value

wd[0] := (float)wt[0];

wd[1] := (float)ws[0];

wd[2] := (float)wt[1];

wd[3] := (float)ws[1];

Requirements

Header:#include<msa.h>

2.6.2. FEXUPL.df

2.6.2.1. v4f32 __builtin_msa_fexupl_w(v8i16)

Specific

Vector Floating-Point Up-Convert Interchange Format Left.

v4f32 wd = __builtin_msa_fexupl_w(v8i16 ws);

Return Value

wd[0] := (float)ws[0];

wd[1] := (float)ws[2];

wd[2] := (float)ws[4];

wd[3] := (float)ws[6];

Requirements

Header:#include<msa.h>

2.6.2.2. v2f64 __builtin_msa_fexupl_d(v4f32)

Specific

Vector Floating-Point Up-Convert Interchange Format Left.

v2f64 wd = __builtin_msa_fexupl_d(v4f32 ws);

Return Value

wd[0] := (double)ws[0];

wd[1] := (double)ws[2];

Requirements

Header:#include<msa.h>

2.6.3. FEXUPR.df

2.6.3.1. v4f32 __builtin_msa_fexupr_w(v8i16)

Specific

Vector Floating-Point Up-Convert Interchange Format Right.

v4f32 wd = __builtin_msa_fexupr_w(v8i16 ws);

Return Value

wd[0] := (float)ws[1];

wd[1] := (float)ws[3];

wd[2] := (float)ws[5];

wd[3] := (float)ws[7];

Requirements

Header:#include<msa.h>

2.6.3.2. v2f64 __builtin_msa_fexupr_d(v4f32)

Specific

Vector Floating-Point Up-Convert Interchange Format Right.

v2f64 wd = __builtin_msa_fexupr_d(v4f32 ws);

Return Value

wd[0] := (double)ws[1];

wd[1] := (double)ws[3];

Requirements

Header:#include<msa.h>

2.6.4. FFINT_S.df

2.6.4.1. v4f32 __builtin_msa_ffint_s_w(v4i32)

Specific

Vector Floating-Point Round and Convert from Signed Integer.

v4f32 wd = __builtin_msa_ffint_s_w(v4i32 ws);

Return Value

wd[0] := (float)ws[0];

wd[1] := (float)ws[1];

wd[2] := (float)ws[2];

wd[3] := (float)ws[3];

Requirements

Header:#include<msa.h>

2.6.4.2. v2f64 __builtin_msa_ffint_s_d(v2i64)

Specific

Vector Floating-Point Round and Convert from Signed Integer.

v2f64 wd = __builtin_msa_ffint_s_d(v2i64 ws);

Return Value

wd[0] :=(double)ws[0];

wd[1] := (double)ws[1];

Requirements

Header:#include<msa.h>

2.6.5. FFINT_U.df

2.6.5.1. v4f32 __builtin_msa_ffint_u_w(v4u32)

Specific

Vector Floating-Point Convert from Unsigned Integer.

v4f32 wd = __builtin_msa_ffint_u_w(v4u32 ws);

Return Value

wd[0] := (float)ws[0];

wd[1] := (float)ws[1];

wd[2] := (float)ws[2];

wd[3] := (float)ws[3];

Requirements

Header:#include<msa.h>

2.6.5.2. v2f64 __builtin_msa_ffint_u_d(v2u64)

Specific

Vector Floating-Point Convert from Unsigned Integer.

v2f64 wd = __builtin_msa_ffint_u_d(v2u64 ws);

Return Value

wd[0] := (double)ws[0];

wd[1] := (double)ws[1];

Requirements

Header:#include<msa.h>

2.6.6. FFQL.df

2.6.6.1. v4f32 __builtin_msa_ffql_w(v8i16)

Specific

Vector Floating-Point Convert from Fixed-Point Left;

v4f32 wd = __builtin_msa_ffql_w(v8i16 ws);

Return Value

wd[0] :=(float)ws[4];

wd[1] :=(float)ws[5];

wd[2] :=(float)ws[6];

wd[3] :=(float)ws[7];

Requirements

Header:#include<msa.h>

2.6.6.2. v2f64 __builtin_msa_ffql_d(v4i32)

Specific

Vector Floating-Point Convert from Fixed-Point Left;

v2f64 wd = __builtin_msa_ffql_d(v4i32 ws);

Return Value

wd[0] :=(double)ws[2];

wd[1] :=(double)ws[3];

Requirements

Header:#include<msa.h>

2.6.7. FFQR.df

2.6.7.1. v4f32 __builtin_msa_ffqr_w(v8i16)

Specific

Vector Floating-Point Convert from Fixed-Point Right.

v4f32 wd = __builtin_msa_ffqr_w(v8i16 ws);

Return Value

wd[0] :=(float)ws[0];

wd[1] :=(float)ws[1];

wd[2] :=(float)ws[2];

wd[3] :=(float)ws[3];

Requirements

Header:#include<msa.h>

2.6.7.2. v2f64 __builtin_msa_ffqr_d(v4i32)

Specific

Vector Floating-Point Convert from Fixed-Point Right.

v2f64 wd = __builtin_msa_ffqr_d(v4i32 ws);

Return Value

wd[0] :=(double)ws[0];

wd[1] :=(double)ws[1];

Requirements

Header:#include<msa.h>

2.6.8. FTINT_S.df

2.6.8.1. v4i32 __builtin_msa_ftint_s_w(v4f32)

Specific

Vector Floating-Point Convert to Signed Integer.

v4i32 wd = __builtin_msa_ftint_s_w(v4f32 ws);

Return Value

wd[0] :=(int)ws[0];

wd[1] :=(int)ws[1];

wd[2] :=(int)ws[2];

wd[3] :=(int)ws[3];

Requirements

Header:#include<msa.h>

2.6.8.2. v2i64 __builtin_msa_ ftint_s _d(v2f64)

Specific

Vector Floating-Point Convert to Signed Integer.

v2i64 wd = __builtin_msa_ ftint_s _d(v2f64 ws);

Return Value

wd[0] := (long long)ws[0];

wd[1] := (long long)ws[1];

Requirements

Header:#include<msa.h>

2.6.9. FTINT_U.df

2.6.9.1. v4u32 __builtin_msa_ftint_u_w(v4f32)

Specific

Vector Floating-Point Round and Convert to Unsigned Integer.

v4u32 wd = __builtin_msa_ftint_u_w(v4f32 ws);

Return Value

wd[0] := (unsigned int)ws[0];

wd[1] := (unsigned int)ws[1];

wd[2] := (unsigned int)ws[2];

wd[3] := (unsigned int)ws[3];

Requirements

Header:#include<msa.h>

2.6.9.2. v2u64 __builtin_msa_ ftint_u _d(v2f64)

Specific

Vector Floating-Point Round and Convert to Unsigned Integer.

v2u64 wd = __builtin_msa_ ftint_u _d(v2f64 ws);

Return Value

wd[0] := (unsigned long long)ws[0];

wd[1] := (unsigned long long)ws[1];

Requirements

Header:#include<msa.h>

2.6.10. FTRUNC_S.df

2.6.10.1. v4i32 __builtin_msa_ftrunc_s_w(v4f32)

Specific

Vector Floating-Point Truncate and Convert to Signed Integer.

v4i32 wd = __builtin_msa_ftrunc_s_w(v4f32 ws);

Return Value

wd[0] := (int)ws[0];

wd[1] := (int)ws[1];

wd[2] := (int)ws[2];

wd[3] := (int)ws[3];

Requirements

Header:#include<msa.h>

2.6.10.2. v2i64 __builtin_msa_ftrunc_s_d(v2f64)

Specific

Vector Floating-Point Truncate and Convert to Signed Integer.

v2i64 wd = __builtin_msa_ftrunc_s_d(v2f64 ws);

Return Value

wd[0] := (long long)ws[0];

wd[1] := (long long)ws[1];

Requirements

Header:#include<msa.h>

2.6.11. FTRUNC_U.df

2.6.11.1. v4u32 __builtin_msa_ftrunc_u_w(v4f32)

Specific

Vector Floating-Point Truncate and Convert to Unsigned Integer.

v4u32 wd = __builtin_msa_ftrunc_u_w(v4f32 ws);

Return Value

wd[0] := (unsigned int)ws[0];

wd[1] := (unsigned int)ws[1];

wd[2] := (unsigned int)ws[2];

wd[3] := (unsigned int)ws[3];

Requirements

Header:#include<msa.h>

2.6.11.2. v2u64 __builtin_msa_ftrunc_u_d(v2f64)

Specific

Vector Floating-Point Truncate and Convert to Unsigned Integer.

v2u64 wd = __builtin_msa_ftrunc_u_d(v2f64 ws);

Return Value

wd[0] := (unsigned long long)ws[0];

wd[1] := (unsigned long long)ws[1];

Requirements

Header:#include<msa.h>

2.6.12. FTQ.df

2.6.12.1. v8i16 __builtin_msa_ftq_h(v4f32, v4f32)

Specific

Vector Floating-Point Convert to Fixed-Point.

v8i16 wd = __builtin_msa_ftq_h(v4f32 ws, v4f32 wt);

Return Value

wd[0] := (Q15)wt[0];

wd[1] := (Q15)ws[0];

wd[2] := (Q15)wt[1];

wd[3] := (Q15)ws[1];

.....

wd[6] := (Q15)wt[3];

wd[7] := (Q15)ws[3];

Requirements

Header:#include<msa.h>

2.6.12.2. v4i32 __builtin_msa_ftq_w(v2f64, v2f64)

Specific

Vector Floating-Point Convert to Fixed-Point.

v4i32 wd = __builtin_msa_ftq_w(v2f64 ws, v2f64 wt);

Return Value

wd[0] := (Q31)wt[0];

wd[1] := (Q31)ws[0];

wd[2] := (Q31)wt[1];

wd[3] := (Q31)ws[1];

Requirements

Header:#include<msa.h>

2.6.13. FEXUPLC.df

2.6.13.1. v4f32 __builtin_msa_fexuplc_w (v8i16,v8i16)

Specific

Configurable Vector Floating-Point Up-Convert Interchange Format Left;

v4f32 wd = __builtin_msa_fexuplc_w (v8i16 ws,v8i16 wt);

Return Value

wd[0] := half_float_convert (ws[4],wt[4]);

wd[1] := half_float_convert (ws[5],wt[5]);

wd[2] := half_float_convert (ws[6],wt[6]);

wd[3] := half_float_convert(ws[7],wt[7]);

Requirements

Header:#include<msa.h>

2.6.14. FEXUPRC.df

2.6.14.1. v4f32 __builtin_msa_fexuprc_w (v8i16,v8i16)

Specific

Configurable Vector Floating-Point Up-Convert Interchange Format Right.

v4f32 wd = __builtin_msa_fexuprc_w (v8i16 ws,v8i16 wt);

Return Value

wd[0] := half_float_convert (ws[0],wt[0]);

wd[1] := half_float_convert (ws[1],wt[1]);

wd[2] := half_float_convert (ws[2],wt[2]);

wd[3] := half_float_convert(ws[3],wt[3]);

Requirements

Header:#include<msa.h>

2.7. Fixed-Point

2.7.1. MADD_Q.df

2.7.1.1. v8i16 __builtin_msa_madd_q _h(v8i16, v8i16, v8i16)

Specific

Vector Fixed-Point Multiply and Add.

v8i16 wd = __builtin_msa_madd_q _h(v8i16 wd, v8i16 ws, v8i16 wt);

Return Value

wd[0] := sats (wd[0] + ws[0] * wt[0] , 16);

wd[1] := sats (wd[1] + ws[1] * wt[1] , 16);

.....

wd[7] := sats (wd[7] + ws[7] * wt[7] , 16);

Requirements

Header:#include<msa.h>

2.7.1.2. v4i32 __builtin_msa_madd_q _w(v4i32, v4i32, v4i32)

Specific

Vector Fixed-Point Multiply and Add.

v4i32 wd = __builtin_msa_madd_q _w(v4i32 wd, v4i32 ws, v4i32 wt);

Return Value

wd[0] := sats (wd[0] + ws[0] * wt[0] , 32);

wd[1] := sats (wd[1] + ws[1] * wt[1] , 32);

wd[2] := sats (wd[2] + ws[2] * wt[2] , 32);

wd[3] := sats (wd[3] + ws[3] * wt[3] , 32);

Requirements

Header:#include<msa.h>

2.7.2. MADDR_Q.df

2.7.2.1. v8i16 __builtin_msa_maddr_q _h(v8i16, v8i16, v8i16)

Specific

Vector Fixed-Point Multiply and Add Rounded.

v8i16 wd = __builtin_msa_maddr_q _h(v8i16 wd, v8i16 ws, v8i16 wt);

Return Value

wd[0] := sats (round_add(wd[0] , ws[0]*wt[0] , 16) , 16);

wd[1] := sats (round_add(wd[1] , ws[1]*wt[1] , 16) , 16);

.....

wd[7] := sats (round_add(wd[7] , ws[7]*wt[7] , 16) , 16);

Requirements

Header:#include<msa.h>

2.7.2.2. v4i32 __builtin_msa_maddr_q _w(v4i32, v4i32, v4i32)

Specific

Vector Fixed-Point Multiply and Add Rounded.

v4i32 wd = __builtin_msa_maddr_q _w(v4i32 wd, v4i32 ws, v4i32 wt);

Return Value

wd[0] := sats (round_add(wd[0] , ws[0]*wt[0] , 32) , 32);

wd[1] := sats (round_add(wd[1] , ws[1]*wt[1] , 32) , 32);

wd[2] := sats (round_add(wd[2] , ws[2]*wt[2] , 32) , 32);

wd[3] := sats (round_add(wd[3] , ws[3]*wt[3] , 32) , 32);

Requirements

Header:#include<msa.h>

2.7.3. MSUB_Q.df

2.7.3.1. v8i16 __builtin_msa_msub_q _h(v8i16, v8i16, v8i16)

Specific

Vector Fixed-Point Multiply and Subtract.

v8i16 wd = __builtin_msa_msub_q _h(v8i16 wd, v8i16 ws, v8i16 wt);

Return Value

wd[0] := sats (wd[0] - ws[0]*wt[0] , 16);

wd[1] := sats (wd[1] - ws[1]*wt[1] , 16);

.....

wd[7] := sats (wd[7] - ws[7]*wt[7] , 16);

Requirements

Header:#include<msa.h>

2.7.3.2. v4i32 __builtin_msa_msub_q _w(v4i32, v4i32, v4i32)

Specific

Vector Fixed-Point Multiply and Subtract.

v4i32 wd = __builtin_msa_msub_q _w(v4i32 wd, v4i32 ws, v4i32 wt);

Return Value

wd[0] := sats (wd[0] - ws[0]*wt[0] , 32);

wd[1] := sats (wd[1] - ws[1]*wt[1] , 32);

wd[2] := sats (wd[2] - ws[2]*wt[2] , 32);

wd[3] := sats (wd[3] - ws[3]*wt[3] , 32);

Requirements

Header:#include<msa.h>

2.7.4. MSUBR_Q.df

2.7.4.1. v8i16 __builtin_msa_msubr_q _h(v8i16, v8i16, v8i16)

Specific

Vector Fixed-Point Multiply and Subtract Rounded.

v8i16 wd = __builtin_msa_msubr_q _h(v8i16 wd, v8i16 ws, v8i16 wt);

Return Value

wd[0] := sats (round_sub(wd[0] , ws[0]*wt[0] , 16) , 16);

wd[1] := sats (round_sub(wd[1] , ws[1]*wt[1] , 16) , 16);

.....

wd[7] := sats (round_sub(wd[7] , ws[7]*wt[7] , 16) , 16);

Requirements

Header:#include<msa.h>

2.7.4.2. v4i32 __builtin_msa_msubr_q _w(v4i32, v4i32, v4i32)

Specific

Vector Fixed-Point Multiply and Subtract Rounded.

v4i32 wd = __builtin_msa_msubr_q _w(v4i32 wd, v4i32 ws, v4i32 wt);

Return Value

wd[0] := sats (round_sub(wd[0] , ws[0]*wt[0] , 32) , 32);

wd[1] := sats (round_sub(wd[1] , ws[1]*wt[1] , 32) , 32);

wd[2] := sats (round_sub(wd[2] , ws[2]*wt[2] , 32) , 32);

wd[3] := sats (round_sub(wd[3] , ws[3]*wt[3] , 32) , 32);

Requirements

Header:#include<msa.h>

2.7.5. MUL_Q.df

2.7.5.1. v8i16 __builtin_msa_mul_q _h(v8i16, v8i16)

Specific

Vector Fixed-Point Multiply.

v8i16 wd = __builtin_msa_mul_q _h(v8i16 ws, v8i16 wt);

Return Value

wd[0] := ws[0] * wt[0];

wd[1] := ws[1] * wt[1];

.....

wd[7] := ws[7] * wt[7];

Requirements

Header:#include<msa.h>

2.7.5.2. v4i32 __builtin_msa_mul_q _w(v4i32, v4i32)

Specific

Vector Fixed-Point Multiply.

v4i32 wd = __builtin_msa_mul_q _w(v4i32 ws, v4i32 wt);

Return Value

wd[0] := ws[0] * wt[0];

wd[1] := ws[1] * wt[1];

wd[2] := ws[2] * wt[2];

wd[3] := ws[3] * wt[3];

Requirements

Header:#include<msa.h>

2.7.6. MULR_Q.df

2.7.6.1. v8i16 __builtin_msa_mulr_q _h(v8i16, v8i16)

Specific

Vector Fixed-Point Multiply Rounded.

v8i16 wd = __builtin_msa_mulr_q _h(v8i16 ws, v8i16 wt);

Return Value

wd[0] := round_mul (ws[0] , wt[0] , 16);

wd[1] := round_mul (ws[1] , wt[1] , 16);

.....

wd[7] := round_mul (ws[7] , wt[7] , 16);

Requirements

Header:#include<msa.h>

2.7.6.2. v4i32 __builtin_msa_mulr_q _w(v4i32, v4i32)

Specific

Vector Fixed-Point Multiply Rounded.

v4i32 wd = __builtin_msa_mulr_q _w(v4i32 ws, v4i32 wt);

Return Value

wd[0] := round_mul (ws[0] , wt[0] , 32);

wd[1] := round_mul (ws[1] , wt[1] , 32);

wd[2] := round_mul (ws[2] , wt[2] , 32);

wd[3] := round_mul (ws[3] , wt[3] , 32);

Requirements

Header:#include<msa.h>

2.8. Branch and Compare

2.8.1. BNZ.df

2.8.1.1. i32 __builtin_msa_bnz_b(v16u8)

Specific

Immediate Branch if All Elements Are Not Zero.

i32 t =__builtin_msa_bnz_b(v16u8 ws);

Return Value

int temp = 1;

temp = temp && ws[0];

temp = temp && ws[1];

.....

temp = temp && ws[15];

t = temp;

Requirements

Header:#include<msa.h>

test_bnz_b:

unsigned char a[16] = {0,1,2,3,

4,5,6,7,

8,9,10,11,

12,13,14,15};

v16i8 va;

va = __builtin_msa_ld_b(a,0);

int t;

t = __builtin_msa_bnz_b((v16u8)va);

if(t)

.....

else

.....

Result:

t=0

2.8.1.2. i32 __builtin_msa_bnz_h(v8u16)

Specific

Immediate Branch if All Elements Are Not Zero.

i32 t = __builtin_msa_bnz_h(v8u16 ws);

Return Value

int temp = 1;

temp = temp && ws[0];

temp = temp && ws[1];

.....

temp = temp && ws[7];

t = temp;

Requirements

Header:#include<msa.h>

2.8.1.3. i32 __builtin_msa_bnz_w(v4u32)

Specific

Immediate Branch if All Elements Are Not Zero.

i32 t = __builtin_msa_bnz_w(v4u32 ws);

Return Value

int temp = 1;

temp = temp && ws[0];

temp = temp && ws[1];

temp = temp && ws[2];

temp = temp && ws[3];

t = temp;

Requirements

Header:#include<msa.h>

2.8.1.4. i32 __builtin_msa_bnz_d(v2u64)

Specific

Immediate Branch if All Elements Are Not Zero.

i32 t = __builtin_msa_bnz_d(v2u64 ws);

Return Value

int temp = 1;

temp = temp && ws[0];

temp = temp && ws[1];

t = temp;

Requirements

Header:#include<msa.h>

2.8.2. BNZ.V

2.8.2.1. i32 __builtin_msa_bnz_v(v16u8)

Specific

Immediate Branch If Not Zero(At Least One Element of Any Format Is Not Zero).

i32 t = __builtin_msa_bnz_v(v16u8 ws);

Return Value

int temp = 0;

temp = temp || ws[0];

temp = temp || ws[1];

.....

temp = temp || ws[15];

t = temp;

Requirements

Header:#include<msa.h>

test_bnz_b:

unsigned char a[16] = {0,0,0,0,

0,0,0,0,

0,0,0,0,

0,0,0,1};

v16i8 va;

va = __builtin_msa_ld_b(a,0);

int t;

t = __builtin_msa_bnz_v((v16u8)va);

if(t)

.....

else

.....

Result:

t=1

2.8.3. BZ.df

2.8.3.1. i32 __builtin_msa_bz _b(v16u8)

Specific

Immediate Branch If At Least One Element Is Zero.

i32 t = __builtin_msa_bz _b(v16u8 ws);

Return Value

int temp = 0;

temp = temp || !ws[0];

temp = temp || !ws[1];

.....

temp = temp || !ws[15];

t = temp;

Requirements

Header:#include<msa.h>

test_bz_b:

unsigned char a[16] = {0,1,2,3,

4,5,6,7,

8,9,10,11,

12,13,14,15};

v16i8 va;

va = __builtin_msa_ld_b(a,0);

int t;

t = __builtin_msa_bz_b((v16u8)va);

if(t)

.....

else

.....

Result:

t=1;

2.8.3.2. i32 __builtin_msa_bz _h(v8u16)

Specific

Immediate Branch If At Least One Element Is Zero.

i32 t = __builtin_msa_bz _h(v8u16 ws);

Return Value

int temp = 0;

temp = temp || !ws[0];

temp = temp || !ws[1];

.....

temp = temp || !ws[7];

t = temp;

Requirements

Header:#include<msa.h>

2.8.3.3. i32 __builtin_msa_bz _w(v4u32)

Specific

Immediate Branch If At Least One Element Is Zero.

i32 t = __builtin_msa_bz _w(v4u32 ws);

Return Value

int temp = 0;

temp = temp || !ws[0];

temp = temp || !ws[1];

temp = temp || !ws[2];

temp = temp || !ws[3];

t = temp;

Requirements

Header:#include<msa.h>

2.8.3.4. i32 __builtin_msa_bz _d(v2u64)

Specific

Immediate Branch If At Least One Element Is Zero.

i32 t = __builtin_msa_bz _d(v2u64 ws);

Return Value

int temp = 0;

temp = temp || !ws[0];

temp = temp || !ws[1];

t = temp;

Requirements

Header:#include<msa.h>

2.8.4. BZ.V

2.8.4.1. i32 __builtin_msa_bz _v(v16u8)

Specific

Immediate Branch If zero(All Elements of Any Format Are Zero).

i32 t = __builtin_msa_bz _v(v16u8 ws);

Return Value

int temp = 1;

temp = temp && !ws[0];

temp = temp && !ws[1];

.....

temp = temp && !ws[15];

t = temp;

Requirements

Header:#include<msa.h>

test_bz_v:

unsigned char a[16] = {0,0,0,0,

0,0,0,0,

0,0,0,0,

0,0,0,0};

v16i8 va;

va = __builtin_msa_ld_b(a,0);

int t;

t = __builtin_msa_bz_v((v16u8)va);

if(t)

.....

else

.....

Result:

t=1

2.8.5. CEQ.df

2.8.5.1. v16i8 __builtin_msa_ceq _b(v16i8,v16i8)

Specific

Vector Compare Equal.

v16i8 wd = __builtin_msa_ceq _b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := (ws[0]==wt[0]) ? one8 : zero8;

wd[1] := (ws[1]==wt[1]) ? one8 : zero8;

.....

wd[15] := (ws[15]==wt[15]) ? one8 : zero8;

Requirements

Header:#include<msa.h>

2.8.5.2. v8i16 __builtin_msa_ceq_h(v8i16,v8i16)

Specific

Vector Compare Equal.

v8i16 wd = __builtin_msa_ceq_h(v8i16 ws,v8i16 wt)

Return Value

wd[0] := (ws[0]==wt[0]) ? one16 : zero16;

wd[1] := (ws[1]==wt[1]) ? one16 : zero16;

.....

wd[7] := (ws[7]==wt[7]) ? one16 : zero16;

Requirements

Header:#include<msa.h>

2.8.5.3. v4i32 __builtin_msa_ceq_w(v4i32,v4i32)

Specific

Vector Compare Equal.

v4i32 wd = __builtin_msa_ceq_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := (ws[0]==wt[0]) ? one32 : zero32;

wd[1] := (ws[1]==wt[1]) ? one32 : zero32;

wd[2] := (ws[2]==wt[2]) ? one32 : zero32;

wd[3] := (ws[3]==wt[3]) ? one32 : zero32;

Requirements

Header:#include<msa.h>

2.8.5.4. v2i64 __builtin_msa_ceq_d(v2i64,v2i64)

Specific

Vector Compare Equal.

v2i64 wd = __builtin_msa_ceq_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := (ws[0]==wt[0]) ? one64 : zero64;

wd[1] := (ws[1]==wt[1]) ? one64 : zero64;

Requirements

Header:#include<msa.h>

2.8.6. CEQI.df

2.8.6.1. v16i8 __builtin_msa_ceq _b(v16i8,imm_n16_15)

Specific

Immediate Compare Equal.

v16i8 wd = __builtin_msa_ceq _b(v16i8 ws,imm_n16_15 s5);

Return Value

wd[0] := (ws[0]==s5) ? one8 : zero8;

wd[1] := (ws[1]==s5) ? one8 : zero8;

.....

wd[15] := (ws[15]==s5) ? one8 : zero8;

Requirements

Header:#include<msa.h>

2.8.6.2. v8i16 __builtin_msa_ceq_h(v8i16, imm_n16_15)

Specific

Immediate Compare Equal.

v8i16 wd = __builtin_msa_ceq_h(v8i16 ws, imm_n16_15 s5);

Return Value

wd[0] := (ws[0]==s5) ? one16 : zero16;

wd[1] := (ws[1]==s5) ? one16 : zero16;

.....

wd[7] := (ws[7]==s5) ? one16 : zero16;

Requirements

Header:#include<msa.h>

2.8.6.3. v4i32 __builtin_msa_ceq_w(v4i32, imm_n16_15)

Specific

Immediate Compare Equal.

v4i32 wd = __builtin_msa_ceq_w(v4i32 ws, imm_n16_15 s5);

Return Value

wd[0] := (ws[0]==s5) ? one32 : zero32;

wd[1] := (ws[1]==s5) ? one32 : zero32;

wd[2] := (ws[2]==s5) ? one32 : zero32;

wd[3] := (ws[3]==s5) ? one32 : zero32;

Requirements

Header:#include<msa.h>

2.8.6.4. v2i64 __builtin_msa_ceq_d(v2i64, imm_n16_15)

Specific

Immediate Compare Equal.

v2i64 wd = __builtin_msa_ceq_d(v2i64 ws, imm_n16_15 s5);

Return Value

wd[0] := (ws[0]==s5) ? one64 : zero64;

wd[1] := (ws[1]==s5) ? one64 : zero64;

Requirements

Header:#include<msa.h>

2.8.7. CLE_S.df

2.8.7.1. v16i8 __builtin_msa_cle_s_b(v16i8,v16i8)

Specific

Vector Compare Signed Less Than or Equal.

v16i8 wd = __builtin_msa_cle_s_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := (ws[0] <= wt[0]) ? one8 : zero8;

wd[1] := (ws[1] <= wt[1]) ? one8 : zero8;

.....

wd[15] := (ws[15] <= wt[15]) ? one8 : zero8;

Requirements

Header:#include<msa.h>

2.8.7.2. v8i16 __builtin_msa_cle_s_h(v8i16,v8i16)

Specific

Vector Compare Signed Less Than or Equal.

v8i16 wd = __builtin_msa_cle_s_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := (ws[0] <= wt[0]) ? one16 : zero16;

wd[1] := (ws[1] <= wt[1]) ? one16 : zero16;

.....

wd[7] := (ws[7] <= wt[7]) ? one16 : zero16;

Requirements

Header:#include<msa.h>

2.8.7.3. v4i32 __builtin_msa_cle_s_w(v4i32,v4i32)

Specific

Vector Compare Signed Less Than or Equal.

v4i32 wd = __builtin_msa_cle_s_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := (ws[0] <= wt[0]) ? one32 : zero32;

wd[1] := (ws[1] <= wt[1]) ? one32 : zero32;

wd[2] := (ws[2] <= wt[2]) ? one32 : zero32;

wd[3] := (ws[3] <= wt[3]) ? one32 : zero32;

Requirements

Header:#include<msa.h>

2.8.7.4. v2i64 __builtin_msa_cle_s_d(v2i64,v2i64)

Specific

Vector Compare Signed Less Than or Equal.

v2i64 wd = __builtin_msa_cle_s_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := (ws[0] <= wt[0]) ? one64 : zero64;

wd[1] := (ws[1] <= wt[1]) ? one64 : zero64;

Requirements

Header:#include<msa.h>

2.8.8. CLEI_S.df

2.8.8.1. v16i8 __builtin_msa_clei_s_b(v16i8,imm_n16_15)

Specific

Immediate Compare Signed Less Than or Equal.

v16i8 wd = __builtin_msa_clei_s_b(v16i8 ws,imm_n16_15 s5);

Return Value

wd[0] := (ws[0] <= s5) ? one8 : zero8;

wd[1] := (ws[1] <= s5) ? one8 : zero8;

.....

wd[15] := (ws[15] <= s5) ? one8 : zero8;

Requirements

Header:#include<msa.h>

2.8.8.2. v8i16 __builtin_msa_clei_s_h(v8i16,imm_n16_15)

Specific

Immediate Compare Signed Less Than or Equal.

v8i16 wd = __builtin_msa_clei_s_h(v8i16 ws,imm_n16_15 s5);

Return Value

wd[0] := (ws[0] <= s5) ? one16 : zero16;

wd[1] := (ws[1] <= s5) ? one16 : zero16;

.....

wd[7] := (ws[7] <= s5) ? one16 : zero16;

Requirements

Header:#include<msa.h>

2.8.8.3. v4i32 __builtin_msa_clei_s_w(v4i32,imm_n16_15)

Specific

Immediate Compare Signed Less Than or Equal.

v4i32 wd = __builtin_msa_clei_s_w(v4i32 ws,imm_n16_15 s5);

Return Value

wd[0] := (ws[0] <= s5) ? one32 : zero32;

wd[1] := (ws[1] <= s5) ? one32 : zero32;

wd[2] := (ws[2] <= s5) ? one32 : zero32;

wd[3] := (ws[3] <= s5) ? one32 : zero32;

Requirements

Header:#include<msa.h>

2.8.8.4. v2i64 __builtin_msa_clei_s_d(v2i64,imm_n16_15)

Specific

Immediate Compare Signed Less Than or Equal.

v2i64 wd = __builtin_msa_clei_s_d(v2i64 ws,imm_n16_15 s5);

Return Value

wd[0] := (ws[0] <= s5) ? one64 : zero64;

wd[1] := (ws[1] <= s5) ? one64 : zero64;

Requirements

Header:#include<msa.h>

2.8.9. CLE_U.df

2.8.9.1. v16i8 __builtin_msa_cle_u_b(v16u8,v16u8)

Specific

Vector Compare Unsigned Less Than or Equal.

v16i8 wd = __builtin_msa_cle_u_b(v16u8 ws,v16u8 wt);

Return Value

wd[0] := (ws[0] <= wt[0]) ? one8 : zero8;

wd[1] := (ws[1] <= wt[0]) ? one8 : zero8;

.....

wd[15] := (ws[15] <= wt[15]) ? one8 : zero8;

Requirements

Header:#include<msa.h>

2.8.9.2. v8i16 __builtin_msa_cle_u_h(v8u16,v8u16)

Specific

Vector Compare Unsigned Less Than or Equal.

v8i16 wd = __builtin_msa_cle_u_h(v8u16 ws,v8u16 wt);

Return Value

wd[0] := (ws[0] <= wt[0]) ? one16 : zero16;

wd[1] := (ws[1] <= wt[0]) ? one16 : zero16;

.....

wd[7] := (ws[7] <= wt[7]) ? one16 : zero16;

Requirements

Header:#include<msa.h>

2.8.9.3. v4i32 __builtin_msa_cle_u_w(v4u32,v4u32)

Specific

Vector Compare Unsigned Less Than or Equal.

v4i32 wd = __builtin_msa_cle_u_w(v4u32 ws,v4u32 wt);

Return Value

wd[0] := (ws[0] <= wt[0]) ? one32 : zero32;

wd[1] := (ws[1] <= wt[0]) ? one32 : zero32;

wd[2] := (ws[2] <= wt[2]) ? one32 : zero32;

wd[3] := (ws[3] <= wt[3]) ? one32 : zero32;

Requirements

Header:#include<msa.h>

2.8.9.4. v2i64 __builtin_msa_cle_u_d(v2u64,v2u64)

Specific

Vector Compare Unsigned Less Than or Equal.

v2i64 wd = __builtin_msa_cle_u_d(v2u64 ws,v2u64 wt);

Return Value

wd[0] := (ws[0] <= wt[0]) ? one64 : zero64;

wd[1] := (ws[1] <= wt[0]) ? one64 : zero64;

Requirements

Header:#include<msa.h>

2.8.10. CLEI_U.df

2.8.10.1. v16i8 __builtin_msa_clei_u_b(v16u8,imm0_31)

Specific

Immediate Compare Unsigned Less Than or Equal.

v16i8 wd = __builtin_msa_clei_u_b(v16u8 ws,imm0_31 u5);

Return Value

wd[0] := (ws[0] <= u5) ? one8: zero8;

wd[1] := (ws[1] <= u5) ? one8 : zero8;

.....

wd[15] := (ws[15] <= u5) ? one8 : zero8;

Requirements

Header:#include<msa.h>

2.8.10.2. v8i16 __builtin_msa_clei_u_h(v8u16,imm0_31)

Specific

Immediate Compare Unsigned Less Than or Equal.

Return Value

wd[0] := (ws[0] <= u5) ? one16 : zero16;

wd[1] := (ws[1] <= u5) ? one16 : zero16;

.....

wd[7] := (ws[7] <= u5) ? one16 : zero16;

Requirements

Header:#include<msa.h>

2.8.10.3. v4i32 __builtin_msa_clei_u_w(v4u32,imm0_31)

Specific

Immediate Compare Unsigned Less Than or Equal.

v4i32 wd = __builtin_msa_clei_u_w(v4u32 ws,imm0_31 u5);

Return Value

wd[0] := (ws[0] <= u5) ? one32 : zero32;

wd[1] := (ws[1] <= u5) ? one32 : zero32;

wd[2] := (ws[2] <= u5) ? one32 : zero32;

wd[3] := (ws[3] <= u5) ? one32 : zero32;

Requirements

Header:#include<msa.h>

2.8.10.4. v2i64 __builtin_msa_clei_u_d(v2u64,imm0_31)

Specific

Immediate Compare Unsigned Less Than or Equal.

v2i64 wd = __builtin_msa_clei_u_d(v2u64 ws,imm0_31 u5);

Return Value

wd[0] := (ws[0] <= u5) ? one64 : zero64;

wd[1] := (ws[1] <= u5) ? one64 : zero64;

Requirements

Header:#include<msa.h>

2.8.11. CLT_S.df

2.8.11.1. v16i8 __builtin_msa_clt_s_b(v16i8,v16i8)

Specific

Vector Compare Signed Less Than.

v16i8 wd = __builtin_msa_clt_s_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := (ws[0] < wt[0]) ? one8 : zero8;

wd[1] := (ws[1] < wt[1]) ? one8 : zero8;

.....

wd[15] := (ws[15] < wt[15]) ? one8 : zero8;

Requirements

Header:#include<msa.h>

2.8.11.2. v8i16 __builtin_msa_clt_s_h(v8i16,v8i16)

Specific

Vector Compare Signed Less Than.

v8i16 wd = __builtin_msa_clt_s_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := (ws[0] < wt[0]) ? one16 : zero16;

wd[1] := (ws[1] < wt[1]) ? one16 : zero16;

.....

wd[7] := (ws[7] < wt[7]) ? one16 : zero16;

Requirements

Header:#include<msa.h>

2.8.11.3. v4i32 __builtin_msa_clt_s_w(v4i32,v4i32)

Specific

Vector Compare Signed Less Than.

v4i32 wd = __builtin_msa_clt_s_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := (ws[0] < wt[0]) ? one32 : zero32;

wd[1] := (ws[1] < wt[1]) ? one32 : zero32;

wd[2] := (ws[2] < wt[2]) ? one32 : zero32;

wd[3] := (ws[3] < wt[3]) ? one32 : zero32;

Requirements

Header:#include<msa.h>

2.8.11.4. v2i64 __builtin_msa_clt_s_d(v2i64,v2i64)

Specific

Vector Compare Signed Less Than.

v2i64 wd = __builtin_msa_clt_s_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := (ws[0] < wt[0]) ? one64 : zero64;

wd[1] := (ws[1] < wt[1]) ? one64 : zero64;

Requirements

Header:#include<msa.h>

2.8.12. CLTI_S.df

2.8.12.1. v16i8 __builtin_msa_clti_s_b(v16i8,imm_n16_15)

Specific

Immediate Compare Signed Less Than.

v16i8 wd = __builtin_msa_clti_s_b(v16i8 ws,imm_n16_15 s5);

Return Value

wd[0] := (ws[0] < s5) ? one8 : zero8;

wd[1] := (ws[1] < s5) ? one8 : zero8;

.....

wd[15] := (ws[15] < s5) ? one8 : zero8;

Requirements

Header:#include<msa.h>

2.8.12.2. v8i16 __builtin_msa_clti_s_h(v8i16,imm_n16_15)

Specific

Immediate Compare Signed Less Than.

v8i16 wd = __builtin_msa_clti_s_h(v8i16 ws,imm_n16_15 s5);

Return Value

wd[0] := (ws[0] < s5) ? one16 : zero16;

wd[1] := (ws[1] < s5) ? one16 : zero16;

.....

wd[7] := (ws[7] < s5) ? one16 : zero16;

Requirements

Header:#include<msa.h>

2.8.12.3. v4i32 __builtin_msa_clti_s_w(v4i32,imm_n16_15)

Specific

Immediate Compare Signed Less Than.

v4i32 wd = __builtin_msa_clti_s_w(v4i32 ws,imm_n16_15 s5);

Return Value

wd[0] := (ws[0] < s5) ? one32 : zero32;

wd[1] := (ws[1] < s5) ? one32 : zero32;

wd[2] := (ws[2] < s5) ? one32 : zero32;

wd[3] := (ws[3] < s5) ? one32 : zero32;

Requirements

Header:#include<msa.h>

2.8.12.4. v2i64 __builtin_msa_clti_s_d(v2i64,imm_n16_15)

Specific

Immediate Compare Signed Less Than.

v2i64 wd = __builtin_msa_clti_s_d(v2i64 ws,imm_n16_15 s5);

Return Value

wd[0] := (ws[0] < s5) ? one64 : zero64;

wd[1] := (ws[1] < s5) ? one64 : zero64;

Requirements

Header:#include<msa.h>

2.8.13. CLT_U.df

2.8.13.1. v16i8 __builtin_msa_clt_u_b(v16u8,v16u8)

Specific

Vector Compare Unsigned Less Than.

v16i8 wd = __builtin_msa_clt_u_b(v16u8 ws,v16u8 wt);

Return Value

wd[0] := (ws[0] < wt[0]) ? one8 : zero8;

wd[1] := (ws[1] < wt[1]) ? one8 : zero8;

.....

wd[15] := (ws[15] < wt[15]) ? one8 : zero8;

Requirements

Header:#include<msa.h>

2.8.13.2. v8i16 __builtin_msa_clt_u_h(v8u16,v8u16)

Specific

Vector Compare Unsigned Less Than.

v8i16 wd = __builtin_msa_clt_u_h(v8u16 ws,v8u16 wt);

Return Value

wd[0] := (ws[0] < wt[0]) ? one16 : zero16;

wd[1] := (ws[1] < wt[1]) ? one16 : zero16;

.....

wd[7] := (ws[7] < wt[7]) ? one16 : zero16;

Requirements

Header:#include<msa.h>

2.8.13.3. v4i32 __builtin_msa_clt_u_w(v4u32,v4u32)

Specific

Vector Compare Unsigned Less Than.

v4i32 wd = __builtin_msa_clt_u_w(v4u32 ws,v4u32 wt);

Return Value

wd[0] := (ws[0] < wt[0]) ? one32 : zero32;

wd[1] := (ws[1] < wt[1]) ? one32 : zero32;

wd[2] := (ws[2] < wt[2]) ? one32 : zero32;

wd[3] := (ws[3] < wt[3]) ? one32 : zero32;

Requirements

Header:#include<msa.h>

2.8.13.4. v2i64 __builtin_msa_clt_u_d(v2u64,v2u64)

Specific

Vector Compare Unsigned Less Than.

v2i64 wd = __builtin_msa_clt_u_d(v2u64 ws,v2u64 wt);

Return Value

wd[0] := (ws[0] < wt[0]) ? one64 : zero64;

wd[1] := (ws[1] < wt[1]) ? one64 : zero64;

Requirements

Header:#include<msa.h>

2.8.14. CLTI_U.df

2.8.14.1. v16i8 __builtin_msa_clti_u_b(v16u8,imm0_31)

Specific

Immediate Compare Unsigned Less Than.

v16i8 wd = __builtin_msa_clti_u_b(v16u8 ws,imm0_31 u5);

Return Value

wd[0] := (ws[0] < u5) ? one8 : zero8;

wd[1] := (ws[1] < u5) ? one8 : zero8;

.....

wd[15] := (ws[15] < u5) ? one8 : zero8;

Requirements

Header:#include<msa.h>

2.8.14.2. v8i16 __builtin_msa_clti_u_h(v8u16,imm0_31)

Specific

Immediate Compare Unsigned Less Than.

v8i16 wd = __builtin_msa_clti_u_h(v8u16 ws,imm0_31 u5);

Return Value

wd[0] := (ws[0] < u5) ? one16 : zero16;

wd[1] := (ws[1] < u5) ? one16 : zero16;

.....

wd[7] := (ws[7] < u5) ? one16 : zero16;

Requirements

Header:#include<msa.h>

2.8.14.3. v4i32 __builtin_msa_clti_u_w(v4u32,imm0_31)

Specific

Immediate Compare Unsigned Less Than.

v4i32 wd = __builtin_msa_clti_u_w(v4u32 ws,imm0_31 u5);

Return Value

wd[0] := (ws[0] < u5) ? one32 : zero32;

wd[1] := (ws[1] < u5) ? one32 : zero32;

wd[2] := (ws[2] < u5) ? one32 : zero32;

wd[3] := (ws[3] < u5) ? one32 : zero32;

Requirements

Header:#include<msa.h>

2.8.14.4. v2i64 __builtin_msa_clti_u_d(v2u64,imm0_31)

Specific

Immediate Compare Unsigned Less Than.

v2i64 wd = __builtin_msa_clti_u_d(v2u64 ws,imm0_31 u5);

Return Value

wd[0] := (ws[0] < u5) ? one64 : zero64;

wd[1] := (ws[1] < u5) ? one64 : zero64;

Requirements

Header:#include<msa.h>

2.9. Load/Store and Move

2.9.1. CFCMSA

2.9.1.1. i32 __builtin_msa_cfcmsa(imm0_31)

Specific

GPR Copy from MSA Control Register.

i32 rd = __builtin_msa_cfcmsa(imm0_31 cs);

Return Value

rd := cs;/*the content of MSA control register cs id copied to GPR rd*/

Requirements

Header:#include<msa.h>

2.9.2. CTCMSA

2.9.2.1. void __builtin_msa_ctcmsa(imm0_31,i32)

Specific

GPR Copy to MSA Control Register.

void __builtin_msa_ctcmsa(imm0_31 cd ,i32 rs);

Return Value

cd := rs;/*The content of GPR rs is copied to MSA control register cd*/

Requirements

Header:#include<msa.h>

2.9.3. LD.df

2.9.3.1. v16i8 __builtin_msa_ld_b(void *,imm_n512_511)

Specific

Vector Load.

v16i8 wd = __builtin_msa_ld_b(void * rs,imm_n512_511 s10);

Return Value

wd[0] = memory(rs + (s10 + 0) * 1);

wd[1] = memory(rs + (s10 + 1) * 1);

.....

wd[15] = memory(rs + (s10 + 15) * 1);

Requirements

Header:#include<msa.h>

Notes

2.9.3.2. v8i16 __builtin_msa_ld_h(void *,imm_n1024_1022)

Specific

Vector Load.

v8i16 wd = __builtin_msa_ld_h(void * rs,imm_n1024_1022 s10);

Return Value

wd[0] = memory(rs + (s10 + 0) * 2);

wd[1] = memory(rs + (s10 + 1) * 2);

.....

wd[7] = memory(rs + (s10 + 7) * 2);

Requirements

Header:#include<msa.h>

2.9.3.3. v4i32 __builtin_msa_ld_w(void *,imm_n2048_2044)

Specific

Vector Load.

v4i32 wd = __builtin_msa_ld_w(void * rs,imm_n2048_2044 s10);

Return Value

wd[0] = memory(rs + (s10 + 0) * 4);

wd[1] = memory(rs + (s10 + 1) * 4);

wd[2] = memory(rs + (s10 + 2) * 4);

wd[3] = memory(rs + (s10 + 3) * 4);

Requirements

Header:#include<msa.h>

2.9.3.4. v2i64 __builtin_msa_ld_d(void *,imm_n4096_4088)

Specific

Vector Load.

v2i64 wd = __builtin_msa_ld_d(void * ws,imm_n4096_4088 s10);

Return Value

wd[0] = memory(rs + (s10 + 0) * 8);

wd[1] = memory(rs + (s10 + 1) * 8);

Requirements

Header:#include<msa.h>

2.9.4. LDI.df

2.9.4.1. v16i8 __builtin_msa_ldi_b(imm_n512_511)

Specific

Immediate Load.

v16i8 wd = __builtin_msa_ldi_b(imm_n512_511 s10);

Return Value

wd[0] := s10;

wd[1] := s10;

.....

wd[15] := s10;

Requirements

Header:#include<msa.h>

2.9.4.2. v8i16 __builtin_msa_ldi_h(imm_n512_511)

Specific

Immediate Load.

v8i16 wd = __builtin_msa_ldi_h(imm_n512_511 s10);

Return Value

wd[0] := s10;

wd[1] := s10;

.....

wd[7] := s10;

Requirements

Header:#include<msa.h>

2.9.4.3. v4i32 __builtin_msa_ldi_w(imm_n512_511)

Specific

Immediate Load.

v4i32 wd = __builtin_msa_ldi_w(imm_n512_511 s10);

Return Value

wd[0] := s10;

wd[1] := s10;

wd[2] := s10;

wd[3] := s10;

Requirements

Header:#include<msa.h>

2.9.4.4. v2i64 __builtin_msa_ldi_d(imm_n512_511)

Specific

Immediate Load.

v2i64 wd = __builtin_msa_ldi_d(imm_n512_511 s10);

Return Value

wd[0] := s10;

wd[1] := s10;

Requirements

Header:#include<msa.h>

2.9.5. MOVE.V

2.9.5.1. v16i8 __builtin_msa_move_v(v16i8)

Specific

Vector Move.

v16i8 wd = __builtin_msa_move_v(v16i8 ws);

Return Value

wd[0] := ws[0];

wd[1] := ws[1];

.....

wd[15] := ws[15];

Requirements

Header:#include<msa.h>

2.9.6. SPLAT.df

2.9.6.1. v16i8 __builtin_msa_splat _b(v16i8,i32)

Specific

GPR Element Splat.

v16i8 wd = __builtin_msa_splat _b(v16i8 ws,i32 rt);

Return Value

wd[0] := ws[rt];

wd[1] := ws[rt];

.....

wd[15] := ws[rt];

Requirements

Header:#include<msa.h>

2.9.6.2. v8i16 __builtin_msa_splat _h(v8i16,i32)

Specific

GPR Element Splat.

v8i16 wd = __builtin_msa_splat _h(v8i16 ws ,i32 rt);

Return Value

wd[0] := ws[rt];

wd[1] := ws[rt];

.....

wd[7] := ws[rt];

Requirements

Header:#include<msa.h>

2.9.6.3. v4i32 __builtin_msa_splat _w(v4i32,i32)

Specific

GPR Element Splat.

v4i32 wd = __builtin_msa_splat _w(v4i32 ws,i32 rt);

Return Value

wd[0] := ws[rt];

wd[1] := ws[rt];

wd[2] := ws[rt];

wd[3] := ws[rt];

Requirements

Header:#include<msa.h>

2.9.6.4. v2i64 __builtin_msa_splat _d(v2i64,i32)

Specific

GPR Element Splat.

v2i64 wd = __builtin_msa_splat _d(v2i64 ws,i32 rt);

Return Value

wd[0] := ws[rt];

wd[1] := ws[rt];

Requirements

Header:#include<msa.h>

2.9.7. SPLATI.df

2.9.7.1. v16i8 __builtin_msa_splati _b(v16i8,imm0_15)

Specific

Immediate Element Splat.

v16i8 wd = __builtin_msa_splati _b(v16i8 ws,imm0_15 n);

Return Value

wd[0] := ws[n];

wd[1] := ws[n];

.....

wd[15] := ws[n];

Requirements

Header:#include<msa.h>

2.9.7.2. v8i16 __builtin_msa_splati _h(v8i16,imm0_7)

Specific

Immediate Element Splat.

v8i16 wd = __builtin_msa_splati _h(v8i16 ws,imm0_7 n);

Return Value

wd[0] := ws[n];

wd[1] := ws[n];

.....

wd[7] := ws[n];

Requirements

Header:#include<msa.h>

2.9.7.3. v4i32 __builtin_msa_splati _w(v4i32,imm0_3)

Specific

Immediate Element Splat.

v4i32 wd = __builtin_msa_splati _w(v4i32 ws,imm0_3 n);

Return Value

wd[0] := ws[n];

wd[1] := ws[n];

wd[2] := ws[n];

wd[3] := ws[n];

Requirements

Header:#include<msa.h>

2.9.7.4. v2i64 __builtin_msa_splati _d(v2i64,imm0_1)

Specific

Immediate Element Splat.

v2i64 wd = __builtin_msa_splati _d(v2i64 ws,imm0_1 n);

Return Value

wd[0] := ws[n];

wd[1] := ws[n];

Requirements

Header:#include<msa.h>

2.9.8. FILL.df

2.9.8.1. v16i8 __builtin_msa_fill_b(i32)

Specific

Vector Fill from GPR.

v16i8 wd = __builtin_msa_fill_b(i32 rs);

Return Value

wd[0] := rs;

wd[1] := rs;

.....

wd[15] := rs;

Requirements

Header:#include<msa.h>

2.9.8.2. v8i16 __builtin_msa_fill_h(i32)

Specific

Vector Fill from GPR.

v8i16 wd = __builtin_msa_fill_h(i32 rs);

Return Value

wd[0] := rs;

wd[1] := rs;

.....

wd[7] := rs;

Requirements

Header:#include<msa.h>

2.9.8.3. v4i32 __builtin_msa_fill_w(i32)

Specific

Vector Fill from GPR.

v4i32 wd = __builtin_msa_fill_w(i32 rs);

Return Value

wd[0] := rs;

wd[1] := rs;

wd[2] := rs;

wd[3] := rs;

Requirements

Header:#include<msa.h>

2.9.8.4. v2i64 __builtin_msa_fill_d(i64)

Specific

Vector Fill from GPR.

v2i64 wd = __builtin_msa_fill_d(i64 rs);

Return Value

wd[0] := rs;

wd[1] := rs;

Requirements

Header:#include<msa.h>

2.9.9. INSERT.df

2.9.9.1. v16i8 __builtin_msa_insert_b(v16i8,imm0_15,i32)

Specific

GPR Insert Element.

v16i8 wd = __builtin_msa_insert_b(v16i8 wd,imm0_15 n,i32 rs);

Return Value

0<=n<=15;

wd[n] := rs;

Requirements

Header:#include<msa.h>

2.9.9.2. v8i16 __builtin_msa_insert_h(v8i16,imm0_7,i32)

Specific

GPR Insert Element.

v8i16 wd = __builtin_msa_insert_h(v8i16 wd,imm0_7 n,i32 rs);

Return Value

0<=n<=7;

wd[n] := rs;

Requirements

Header:#include<msa.h>

2.9.9.3. v4i32 __builtin_msa_insert_w(v4i32,imm0_3,i32)

Specific

GPR Insert Element.

v4i32 wd = __builtin_msa_insert_w(v4i32 wd,imm0_3 n,i32 rs);

Return Value

0<=n<=3;

wd[n] := rs;

Requirements

Header:#include<msa.h>

2.9.9.4. v2i64 __builtin_msa_insert_d(v2i64,imm0_1,i64)

Specific

GPR Insert Element.

v2i64 wd = __builtin_msa_insert_d(v2i64 wd,imm0_1 n,i64 rs);

Return Value

0<=n<=1;

wd[n] := rs;

Requirements

Header:#include<msa.h>

2.9.10. INSVE.df

2.9.10.1. v16i8 __builtin_msa_insve_b(v16i8,imm0_15,v16i8)

Specific

Element Insert Element.

v16i8 wd = __builtin_msa_insve_b(v16i8 wd,imm0_15 n,v16i8 ws);

Return Value

0<=n<=15;

wd[n] := ws[0];

Requirements

Header:#include<msa.h>

2.9.10.2. v8i16 __builtin_msa_insve_h(v8i16,imm0_7,v8i16)

Specific

Element Insert Element.

v8i16 wd = __builtin_msa_insve_h(v8i16 wd,imm0_7 n,v8i16 ws);

Return Value

0<=n<=7;

wd[n] := ws[0];

Requirements

Header:#include<msa.h>

2.9.10.3. v4i32 __builtin_msa_insve_w(v4i32,imm0_3,v4i32)

Specific

Element Insert Element.

v4i32 wd = __builtin_msa_insve_w(v4i32 wd,imm0_3 n,v4i32 ws);

Return Value

0<=n<=3;

wd[n] := ws[0];

Requirements

Header:#include<msa.h>

2.9.10.4. v2i64 __builtin_msa_insve_d(v2i64,imm0_1,v2i64)

Specific

Element Insert Element.

v2i64 wd = __builtin_msa_insve_d(v2i64 wd,imm0_1 n,v2i64 ws);

Return Value

0<=n<=1;

wd[n] := ws[0];

Requirements

Header:#include<msa.h>

2.9.11. COPY_S.df

2.9.11.1. i32 __builtin_msa_copy_s _b(v16i8,imm0_15)

Specific

Element Copy to GPR Signed.

i32 rd = __builtin_msa_copy_s _b(v16i8 ws,imm0_15 n);

Return Value

0<=n<=15;

rd := ws[n];

Requirements

Header:#include<msa.h>

2.9.11.2. i32 __builtin_msa_copy_s_h(v8i16,imm0_7)

Specific

Element Copy to GPR Signed.

i32 rd = __builtin_msa_copy_s_h(v8i16 ws,imm0_7 n);

Return Value

0<=n<=7;

rd := ws[n];

Requirements

Header:#include<msa.h>

2.9.11.3. i32 __builtin_msa_copy_s_w(v4i32,imm0_3)

Specific

Element Copy to GPR Signed.

i32 rd = __builtin_msa_copy_s_w(v4i32 ws,imm0_3 n);

Return Value

0<=n<=3;

rd := ws[n];

Requirements

Header:#include<msa.h>

2.9.12. COPY_U.df

2.9.12.1. u32 __builtin_msa_copy_u _b(v16i8,imm0_15)

Specific

Element Copy to GPR Unsigned.

u32 wd = __builtin_msa_copy_u _b(v16i8 ws,imm0_15 n);

Return Value

0<=n<=15;

rd := ws[n];

Requirements

Header:#include<msa.h>

2.9.12.2. u32 __builtin_msa_copy_u_h(v8i16,imm0_7)

Specific

Element Copy to GPR Unsigned.

u32 wd = __builtin_msa_copy_u_h(v8i16 ws,imm0_7 n);

Return Value

0<=n<=7;

rd := ws[n];

Requirements

Header:#include<msa.h>

2.9.12.3. u32 __builtin_msa_copy_u_w(v4i32,imm0_3)

Specific

Element Copy to GPR Unsigned.

u32 wd = __builtin_msa_copy_u_w(v4i32 ws,imm0_3 n);

Return Value

0<=n<=3;

rd := ws[n];

Requirements

Header:#include<msa.h>

2.9.13. ST.df

2.9.13.1. void __builtin_msa_st_b(v16i8,void*,imm_n512_511)

Specific

Vector Store.

void __builtin_msa_st_b(v16i8 wd,void* rs,imm_n512_511 s10);

Return Value

memory(rs + (s10 + 0) * 1) := wd[0];

memory(rs + (s10 + 1) * 1) := wd[1];

.....

memory(rs + (s10 + 15) * 1) := wd[15];

Requirements

Header:#include<msa.h>

2.9.13.2. void __builtin_msa_st_h(v8i16,void*,imm_n1024_1022)

Specific

Vector Store.

void __builtin_msa_st_h(v8i16 wd,void* rs,imm_n1024_1022 s10);

Return Value

memory(rs + (s10 + 0) * 2) := wd[0];

memory(rs + (s10 + 1) * 2) := wd[1];

.....

memory(rs + (s10 + 7) * 2) := wd[7];

Requirements

Header:#include<msa.h>

2.9.13.3. void __builtin_msa_st_w(v4i32,void*,imm_n2048_2044)

Specific

Vector Store.

void __builtin_msa_st_w(v4i32 wd,void* rs,imm_n2048_2044 s10);

Return Value

memory(rs + (s10 + 0) * 4) := wd[0];

memory(rs + (s10 + 1) * 4) := wd[1];

memory(rs + (s10 + 2) * 4) := wd[2];

memory(rs + (s10 + 3) * 4) := wd[3];

Requirements

Header:#include<msa.h>

2.9.13.4. void __builtin_msa_st_d(v2i64,void*,imm_n4096_4088)

Specific

Vector Store.

void __builtin_msa_st_d(v2i64 wd,void* rs,imm_n4096_4088 s10);

Return Value

memory(rs + (s10 + 0) * 8) := wd[0];

memory(rs + (s10 + 1) * 8) := wd[1];

Requirements

Header:#include<msa.h>

2.9.14. LDINSS.W

2.9.14.1. v4i32 __builtin_msa_ldinss_w(v4i32,imm0_3,void *,int)

Specific

Load element insert element from sparse memory.

v4i32 wd = __builtin_msa_ldinss_w(v4i32 wd,imm0_3 n,void *base,int index);

Return Value

n=0,1,2,3;

offset = ((char*) index)[n]*4;

addr = base+offset;

if(addr&0x3)

    SignalException(AddressError);

wd[n] := memory(addr);

Requirements

Header:#include<msa.h>

2.9.15. LDINSSU.W

2.9.15.1. v4i32 __builtin_msa_ldinssu_w(v4i32,imm0_3,void *,int)

Specific

Load element insert element and update base address.

v4i32 wd = __builtin_msa_ldinssu_w(v4i32 wd,imm0_3 n,void *base,int index);

Return Value

n=0,1,2,3;

offset = ((char*) index)[n]*4;

addr = base+offset;

if(addr&0x3)

SignalException(AddressError);

wd[n] := memory(addr);

base := addr;

Requirements

Header:#include<msa.h>

2.9.16. LDINSSU2.W

2.9.16.1. v4i32 __builtin_msa_ldinssu2_w (v4i32,imm0_3,void *, int)

Specific

Load element insert element and update base address.

v4i32 wd = __builtin_msa_ldinssu2_w (v4i32 wd,imm0_3 n,void * bsae, int index);

Return Value

n=0,1,2,3,4,5,6,7;

offset =((index <<(32-(4n+4)))>>28)*4;

addr = base+offset;

if(addr&0x3)

SignalException(AddressError);

wd[n%4] := memory(addr);

base := addr;

Requirements

Header:#include<msa.h>

2.9.17. LDINS.df

2.9.17.1. v4i32 __builtin_msa_ldins_w(v4i32,imm0_3,void *,int)

Specific

Load element insert element.

v4i32 wd = __builtin_msa_ldins_w(v4i32 wd,imm0_3 n,void * base,int index);

Return Value

n=0,1,2,3;

offset=index;

addr = base+offset;

if(addr&0x3)

    SignalException(AddressError);

wd[n] := memory (addr);

Requirements

Header:#include<msa.h>

2.9.17.2. v2i64 __builtin_msa_ldins_d(v2i64,imm0_3,void *,int)

Specific

Load element insert element.

v2i64 wd = __builtin_msa_ldins_d(v2i64 wd,imm0_3 n,void *base,int index);

Return Value

n=0,1;

offset=index;

addr = base+offset;

if(addr&0x7)

SignalException(AddressError);

wd[n] := memory (addr);

Requirements

Header:#include<msa.h>

2.9.18. LDINSU.df

2.9.18.1. v4i32 __builtin_msa_ldinsu_w(v4i32,imm0_3,void *,int)

Specific

Load element insert element and update base address.

v4i32 wd = __builtin_msa_ldinsu_w(v4i32 wd,imm0_3 n,void *base,int index);

Return Value

n=0,1,2,3;

offset=index;

addr = base+offset;

if(addr&0x3)

SignalException(AddressError);

wd[n] := memory (addr);

base := addr;

Requirements

Header:#include<msa.h>

2.9.18.2. v2i64 __builtin_msa_ldinsu_d(v2i64,imm0_3,void *,int)

Specific

Load element insert element and update base address.

v2i64 wd = __builtin_msa_ldinsu_d(v2i64 wd,imm0_3 n,void *base,int index);

Return Value

n=0,1;

offset=index;

addr = base+offset;

if(addr&0x7)

SignalException(AddressError);

wd[n] := memory (addr);

base := addr;

Requirements

Header:#include<msa.h>

2.9.19. STEXT.df

2.9.19.1. void __builtin_msa_stext _w(v4i32,imm0_3,void *,int)

Specific

Store extracted element.

void __builtin_msa_stext _w(v4i32 wd,imm0_3 n,void *base,int index);

Return Value

n=0,1,2,3;

offset=index;

addr = base+offset;

if(addr&0x3)

SignalException(AddressError);

memory(addr) := wd[n];

Requirements

Header:#include<msa.h>

2.9.19.2. void __builtin_msa_stext_d(v2i64,imm0_3,void *,int)

Specific

Store extracted element.

void __builtin_msa_stext_d(v2i64 wd,imm0_3 n,void *base,int index);

Return Value

n=0,1;

offset=index;

addr = base+offset;

if(addr&0x7)

SignalException(AddressError);

memory(addr) := wd[n];

Requirements

Header:#include<msa.h>

2.9.20. STEXTU.df

2.9.20.1. void __builtin_msa_stextu _w(v4i32,imm0_3,void *,int)

Specific

Store extracted element and update base address.

void __builtin_msa_stextu _w(v4i32 wd,imm0_3 n,void *base,int index);

Return Value

n=0,1,2,3;

offset=index;

addr = base+offset;

if(addr&0x3)

SignalException(AddressError);

memory(addr) := wd[n];

base := addr;

Requirements

Header:#include<msa.h>

2.9.20.2. void __builtin_msa_stextu_d(v2i64,imm0_3,void *,int)

Specific

Store extracted element and update base address.

void __builtin_msa_stextu_d(v2i64 wd,imm0_3 n,void *base,int index);

Return Value

n=0,1;

offset=index;

addr = base+offset;

if(addr&0x7)

SignalException(AddressError);

memory(addr) := wd[n];

base := addr;

Requirements

Header:#include<msa.h>

2.9.21. MOVE.W

2.9.21.1. v4i32 __builtin_msa_move_w(v4i32,imm0_3,v4i32,imm0_3)

Specific

Vector Move.

v4i32 wd = ltin_msa_move_w(v4i32 wd,imm0_3 m,4i32 ws,imm0_3 n);

Return Value

m=0,1,2,3;

n=0,1,2,3;

wd[m] := ws[n];

Requirements

Header:#include<msa.h>

2.10. Element Permute

2.10.1. ILVEV.df

2.10.1.1. v16i8 __builtin_msa_ilvev_b(v16i8,v16i8)

Specific

Vector Interleave Even.

v16i8 wd = __builtin_msa_ilvev_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := wt[0];

wd[1] := ws[0];

wd[2] := wt[2];

wd[3] := ws[2];

.....

wd[14] := wt[14];

wd[15] := ws[14];

Requirements

Header:#include<msa.h>

2.10.1.2. v8i16 __builtin_msa_ilvev_h(v8i16,v8i16)

Specific

Vector Interleave Even.

v8i16 wd = __builtin_msa_ilvev_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := wt[0];

wd[1] := ws[0];

wd[2] := wt[2];

wd[3] := ws[2];

.....

wd[6] := wt[6];

wd[7] := ws[6];

Requirements

Header:#include<msa.h>

2.10.1.3. v4i32 __builtin_msa_ilvev_w(v4i32,v4i32)

Specific

Vector Interleave Even.

v4i32 wd = __builtin_msa_ilvev_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := wt[0];

wd[1] := ws[0];

wd[2] := wt[2];

wd[3] := ws[2];

Requirements

Header:#include<msa.h>

2.10.1.4. v2i64 __builtin_msa_ilvev_d(v2i64,v2i64)

Specific

Vector Interleave Even.

v2i64 wd = __builtin_msa_ilvev_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := wt[0];

wd[1] := ws[0];

Requirements

Header:#include<msa.h>

2.10.2. ILVOD.df

2.10.2.1. v16i8 __builtin_msa_ilvod_b(v16i8,v16i8)

Specific

Vector Interleave Odd.

v16i8 wd = __builtin_msa_ilvod_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := wt[1];

wd[1] := ws[1];

wd[2] := wt[3];

wd[3] := ws[3];

.....

wd[14] := wt[15];

wd[15] := ws[15];

Requirements

Header:#include<msa.h>

2.10.2.2. v8i16 __builtin_msa_ilvod_h(v8i16,v8i16)

Specific

Vector Interleave Odd.

v8i16 wd = __builtin_msa_ilvod_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := wt[1];

wd[1] := ws[1];

wd[2] := wt[3];

wd[3] := ws[3];

.....

wd[6] := wt[7];

wd[7] := ws[7];

Requirements

Header:#include<msa.h>

2.10.2.3. v4i32 __builtin_msa_ilvod_w(v4i32,v4i32)

Specific

Vector Interleave Odd.

v4i32 wd = __builtin_msa_ilvod_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := wt[1];

wd[1] := ws[1];

wd[2] := wt[3];

wd[3] := ws[3];

Requirements

Header:#include<msa.h>

2.10.2.4. v2i64 __builtin_msa_ilvod_d(v2i64,v2i64)

Specific

Vector Interleave Odd.

v2i64 wd = __builtin_msa_ilvod_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := wt[1];

wd[1] := ws[1];

Requirements

Header:#include<msa.h>

2.10.3. ILVL.df

2.10.3.1. v16i8 __builtin_msa_ilvl_b(v16i8,v16i8)

Specific

Vector Interleave Left.

v16i8 wd = __builtin_msa_ilvl_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := wt[8];

wd[1] := ws[8];

wd[2] := wt[9];

wd[3] := ws[9];

.....

wd[14] := wt[15];

wd[15] := ws[15];

Requirements

Header:#include<msa.h>

2.10.3.2. v8i16 __builtin_msa_ilvl_h(v8i16,v8i16)

Specific

Vector Interleave Left.

v8i16 wd = __builtin_msa_ilvl_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := wt[4];

wd[1] := ws[4];

wd[2] := wt[5];

wd[3] := ws[5];

.....

wd[6] := wt[7];

wd[7] := ws[7];

Requirements

Header:#include<msa.h>

2.10.3.3. v4i32 __builtin_msa_ilvl_w(v4i32,v4i32)

Specific

Vector Interleave Left.

v4i32 wd = __builtin_msa_ilvl_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := wt[2];

wd[1] := ws[2];

wd[2] := wt[3];

wd[3] := ws[3];

Requirements

Header:#include<msa.h>

2.10.3.4. v2i64 __builtin_msa_ilvl_d(v2i64,v2i64)

Specific

Vector Interleave Left.

v2i64 wd = __builtin_msa_ilvl_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := wt[1];

wd[1] := ws[1];

Requirements

Header:#include<msa.h>

2.10.4. ILVR.df

2.10.4.1. v16i8 __builtin_msa_ilvr_b(v16i8,v16i8)

Specific

Vector Interleave Right.

v16i8 wd = __builtin_msa_ilvr_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := wt[0];

wd[1] := ws[0];

wd[2] := wt[1];

wd[3] := ws[1];

.....

wd[14] := wt[7];

wd[15] := ws[7];

Requirements

Header:#include<msa.h>

2.10.4.2. v8i16 __builtin_msa_ilvr_h(v8i16,v8i16)

Specific

Vector Interleave Right.

v8i16 wd = __builtin_msa_ilvr_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := wt[0];

wd[1] := ws[0];

wd[2] := wt[1];

wd[3] := ws[1];

.....

wd[6] := wt[3];

wd[7] := ws[3];

Requirements

Header:#include<msa.h>

2.10.4.3. v4i32 __builtin_msa_ilvr_w(v4i32,v4i32)

Specific

Vector Interleave Right.

v4i32 wd = __builtin_msa_ilvr_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := wt[0];

wd[1] := ws[0];

wd[2] := wt[1];

wd[3] := ws[1];

Requirements

Header:#include<msa.h>

2.10.4.4. v2i64 __builtin_msa_ilvr_d(v2i64,v2i64)

Specific

Vector Interleave Right.

v2i64 wd = __builtin_msa_ilvr_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := wt[0];

wd[1] := ws[0];

Requirements

Header:#include<msa.h>

2.10.5. PCKEV.df

2.10.5.1. v16i8 __builtin_msa_pckev _b(v16i8,v16i8)

Specific

Vector Pack Even.

v16i8 wd = __builtin_msa_pckev _b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := wt[0];

wd[1] := wt[2];

wd[2] := wt[4];

wd[3] := wt[6];

wd[4] := wt[8];

wd[5] := wt[10];

wd[6] := wt[12];

wd[7] := wt[14];

wd[8] := ws[0];

wd[9] := ws[2];

wd[10] := ws[4];

wd[11] := ws[6];

wd[12] := ws[8];

wd[13] := ws[10];

wd[14] := ws[12];

wd[15] := ws[14];

Requirements

Header:#include<msa.h>

2.10.5.2. v8i16 __builtin_msa_pckev_h(v8i16,v8i16)

Specific

Vector Pack Even.

v8i16 wd = __builtin_msa_pckev_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := wt[0];

wd[1] := wt[2];

wd[2] := wt[4];

wd[3] := wt[6];

wd[4] := ws[0];

wd[5] := ws[2];

wd[6] := ws[4];

wd[7] := ws[6];

Requirements

Header:#include<msa.h>

2.10.5.3. v4i32 __builtin_msa_pckev_w(v4i32,v4i32)

Specific

Vector Pack Even.

v4i32 wd = __builtin_msa_pckev_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := wt[0];

wd[1] := wt[2];

wd[2] := ws[0];

wd[3] := ws[2];

Requirements

Header:#include<msa.h>

2.10.5.4. v2i64 __builtin_msa_pckev_d(v2i64,v2i64)

Specific

Vector Pack Even.

v2i64 wd = __builtin_msa_pckev_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := wt[0];

wd[1] := ws[0];

Requirements

Header:#include<msa.h>

2.10.6. PCKOD.df

2.10.6.1. v16i8 __builtin_msa_pckod_b(v16i8,v16i8)

Specific

Vector Pack Odd.

v16i8 wd = __builtin_msa_pckod_b(v16i8 ws,v16i8 wt);

Return Value

wd[0] := wt[1];

wd[1] := wt[3];

wd[2] := wt[5];

wd[3] := wt[7];

wd[4] := wt[9];

wd[5] := wt[11];

wd[6] := wt[13];

wd[7] := wt[15];

wd[8] := ws[1];

wd[9] := ws[3];

wd[10] := ws[5];

wd[11] := ws[7];

wd[12] := ws[9];

wd[13] := ws[11];

wd[14] := ws[13];

wd[15] := ws[15];

Requirements

Header:#include<msa.h>

2.10.6.2. v8i16 __builtin_msa_pckod_h(v8i16,v8i16)

Specific

Vector Pack Odd.

v8i16 wd = __builtin_msa_pckod_h(v8i16 ws,v8i16 wt);

Return Value

wd[0] := wt[1];

wd[1] := wt[3];

wd[2] := wt[5];

wd[3] := wt[7];

wd[4] := ws[1];

wd[5] := ws[3];

wd[6] := ws[5];

wd[7] := ws[7];

Requirements

Header:#include<msa.h>

2.10.6.3. v4i32 __builtin_msa_pckod_w(v4i32,v4i32)

Specific

Vector Pack Odd.

v4i32 wd = __builtin_msa_pckod_w(v4i32 ws,v4i32 wt);

Return Value

wd[0] := wt[1];

wd[1] := wt[3];

wd[2] := ws[1];

wd[3] := ws[3];

Requirements

Header:#include<msa.h>

2.10.6.4. v2i64 __builtin_msa_pckod_d(v2i64,v2i64)

Specific

Vector Pack Odd.

v2i64 wd = __builtin_msa_pckod_d(v2i64 ws,v2i64 wt);

Return Value

wd[0] := wt[1];

wd[1] := ws[1];

Requirements

Header:#include<msa.h>

2.10.7. SHF.df

2.10.7.1. v16i8 __builtin_msa_shf_b(v16i8,imm0_255)

Specific

Immediate Set Shuffle Elements.

v16i8 wd = __builtin_msa_shf_b(v16i8 ws,imm0_255 i8);

Return Value

wd[0] := ws[shf_index(i8,0)];

wd[1] := ws[shf_index(i8,1)];

.....

wd[15] := ws[shf_index(i8,15)];

Requirements

Header:#include<msa.h>

2.10.7.2. v8i16 __builtin_msa_shf_h(v8i16,imm0_255)

Specific

Immediate Set Shuffle Elements.

v8i16 wd = __builtin_msa_shf_h(v8i16 ws,imm0_255 i8);

Return Value

wd[0] := ws[shf_index(i8,0)];

wd[1] := ws[shf_index(i8,1)];

.....

wd[7] := ws[shf_index(i8,7)];

Requirements

Header:#include<msa.h>

2.10.7.3. v4i32 __builtin_msa_shf_w(v4i32,imm0_255)

Specific

Immediate Set Shuffle Elements.

v4i32 wd = __builtin_msa_shf_w(v4i32 ws,imm0_255 i8);

Return Value

wd[0] := ws[shf_index(i8,0)];

wd[1] := ws[shf_index(i8,1)];

wd[2] := ws[shf_index(i8,2)];

wd[3] := ws[shf_index(i8,3)];

Requirements

Header:#include<msa.h>

2.10.8. SLD.df

2.10.8.1. v16i8 __builtin_msa_sld_b(v16i8,v16i8,i32)

Specific

GPR Columns Slide.

v16i8 wd = __builtin_msa_sld_b(v16i8 wd,v16i8 ws,i32 rt);

Return Value

rt=rt%16;

wd[0] := (0+rt) <=15 ? ws[0+rt] : wd[0+rt-16];

wd[1] := (1+rt) <=15 ? ws[1+rt] : wd[1+rt-16];

.....

wd[15] := (15+rt) <=15 ? ws[15+rt] : wd[15+rt-16]

Requirements

Header:#include<msa.h>

Graphic example:

wd[16]={0,1,2,3,

4,5,6,7,

8,9,10,11,

12,13,14,15};

ws[16]={16,17,18,19,

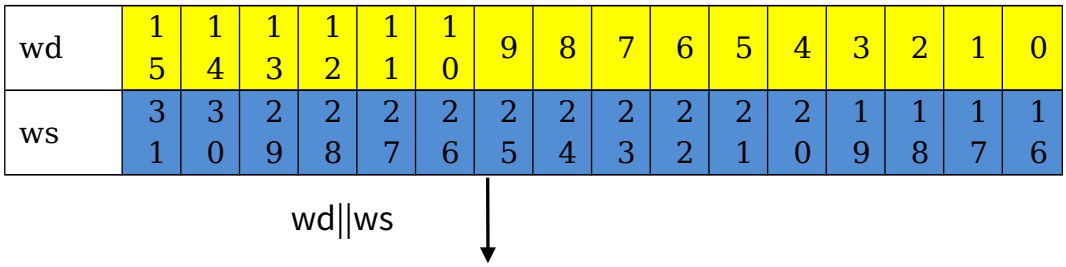
20,21,22,23,

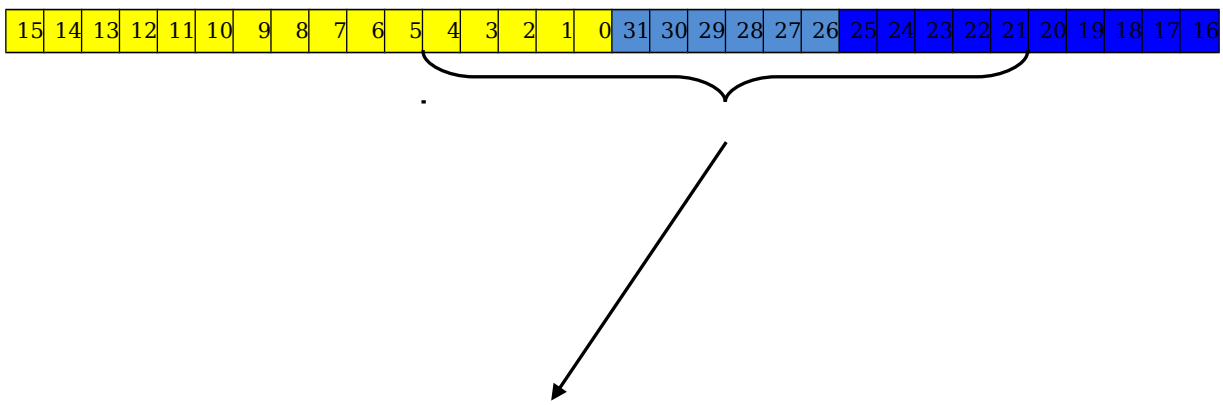
24,25,26,27,

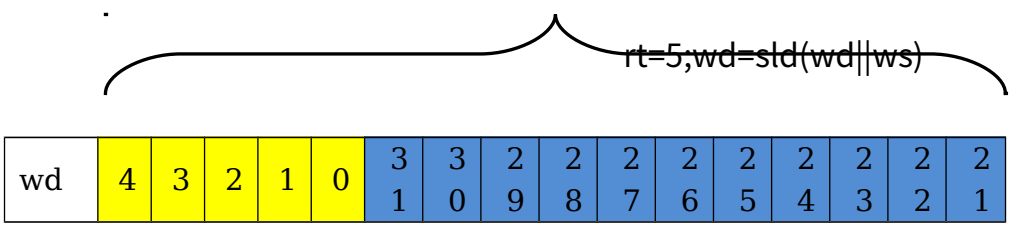
28,29,30,31};

rt=5;

step1:







Result:

wd[16]={21,22,23,24,

25,26,27,28,

29,30,31,0,

1,2,3,4};

2.10.8.2. v8i16 __builtin_msa_sld_h(v8i16,v8i16,i32)

Specific

GPR Columns Slide.

v8i16 wd = __builtin_msa_sld_h(v8i16 wd,v8i16 ws,i32 rt);

Return Value

rt=rt%8;

k=0,1

if(k==0)

{

wd[0] := (0+rt) <= 7 ? ws[0+rt] : wd[0+rt-8];

wd[1] := (1+rt) <= 7 ? ws[1+rt] : wd[1+rt-8];

.....

wd[7] := (7+rt) <= 7 ? ws[7+rt] : wd[7+rt-8];

}

else

{

wd[8] := (0+rt)<=7 ? ws[8+0+rt] : wd[8+0+rt-8];

wd[9] := (1+rt)<=7 ? ws[8+1+rt] : wd[8+1+rt-8];

.....

wd[15] := (7+rt)<=7 ? ws[8+7+rt] : wd[8+7+rt-8];

}

Requirements

Header:#include<msa.h>

Graphic example:

(v16i8)wd[16]={0,1,2,3,

4,5,6,7,

8,9,10,11,

12,13,14,15};

(v16i8)ws[16]={16,17,18,19,

20,21,22,23,

24,25,26,27,

28,29,30,31};

rt=5;

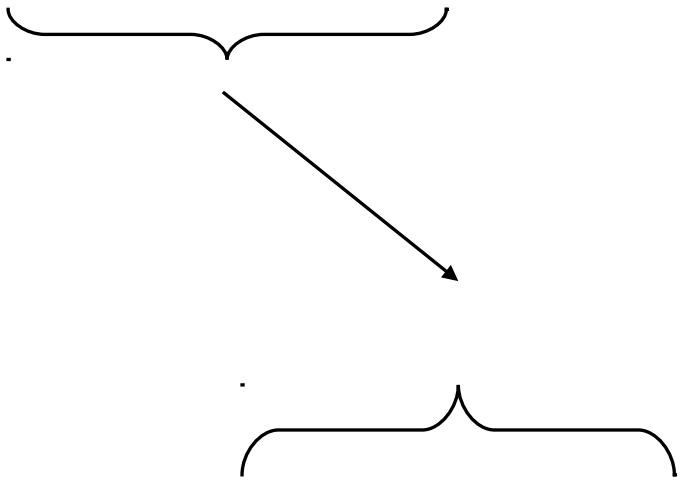
step1:

wd	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ws	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16

step2:

k=0;

Wd_{64-0}|| ws_{64-0}	7	6	5	4	3	2	1	0	23	22	21	20	19	18	17	16

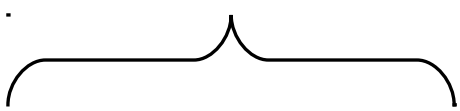


rt=5;

wd	15	14	13	12	11	10	9	8	4	3	2	1	0	23	22	21

k=1;

wd_{128-64}|| ws_{128-64}	15	14	13	12	11	10	9	8	31	30	29	28	27	26	25	24



rt=5;

wd	12	11	10	9	8	31	30	29	4	3	2	1	0	23	22	21

Result:

wd[16]={ 21,22,23,0,

1,2,3,4,

29,30,31,8,

```
9,10,11,12};
```

2.10.8.3. v4i32 __builtin_msa_sld_w(v4i32,v4i32,i32)

Specific

GPR Columns Slide.

v4i32 wd = __builtin_msa_sld_w(v4i32 wd,v4i32 ws,i32 rt);

Return Value

```
rt=rt%4;
if ( k==0 )
{
    wd[0] := (0+rt) <= 3 ? ws[0+rt] : wd[0+rt-4];
    wd[1] := (1+rt) <= 3 ? ws[1+rt] : wd[1+rt-4];
    wd[2] := (2+rt) <= 3 ? ws[2+rt] : wd[2+rt-4];
    wd[3] := (3+rt) <= 3 ? ws[3+rt] : wd[3+rt-4];
}
else if(k==1)
{
    wd[4] := (0+rt) <= 3 ? ws[4+0+rt] : wd[4+0+rt-4];
    wd[5] := (1+rt) <= 3 ? ws[4+1+rt] : wd[4+1+rt-4];
    wd[6] := (2+rt) <= 3 ? ws[4+2+rt] : wd[4+2+rt-4];
    wd[7] := (3+rt) <= 3 ? ws[4+3+rt] : wd[4+3+rt-4];
```

```
}
else if(k==2)
{
    wd[8] := (0+rt) <= 3 ? ws[8+0+rt] : wd[8+0+rt-4];
    wd[9] := (1+rt) <= 3 ? ws[8+1+rt] : wd[8+1+rt-4];
    wd[10] := (2+rt) <= 3 ? ws[8+2+rt] : wd[8+2+rt-4];
    wd[11] := (3+rt) <= 3 ? ws[8+3+rt] : wd[8+3+rt-4];

}
else if(k==3)
{
    wd[12] := (0+rt) <= 3 ? ws[12+0+rt] : wd[12+0+rt-4];
    wd[13] := (1+rt) <= 3 ? ws[12+1+rt] : wd[12+1+rt-4];
    wd[14] := (2+rt) <= 3 ? ws[12+2+rt] : wd[12+2+rt-4];
    wd[15] := (3+rt) <= 3 ? ws[12+3+rt] : wd[12+3+rt-4];
}
```

Requirements

Header:#include<msa.h>

2.10.8.4. v2i64 __builtin_msa_sld_d(v2i64,v2i64,i32)

Specific

GPR Columns Slide.

v2i64 wd = __builtin_msa_sld_d(v2i64 wd,v2i64 ws,i32 rt);

Return Value

```
rt=rt%2;
if(k==0)
{
        wd[0] := (0+rt) <= 1 ? ws[0+rt] : wd[0+rt-2];
        wd[1] := (1+rt) <= 1 ? ws[1+rt] : wd[1+rt-2];
}
else if(k==1)
{
        wd[2] := (0+rt) <= 1 ? ws[2+0+rt] : wd[2+0+rt-2];
        wd[3] := (1+rt) <= 1 ? ws[2+1+rt] : wd[2+1+rt-2];
}
else if(k==2)
{
        wd[4] := (0+rt) <= 1 ? ws[4+0+rt] : wd[4+0+rt-2];
        wd[5] := (1+rt) <= 1 ? ws[4+1+rt] : wd[4+1+rt-2];
}
else if(k==3)
{
        wd[6] := (0+rt) <= 1 ? ws[6+0+rt] : wd[6+0+rt-2];
        wd[7] := (1+rt) <= 1 ? ws[6+1+rt] : wd[6+1+rt-2];
}
```

```
else if(k==4)
{
        wd[8] := (0+rt) <= 1 ? ws[8+0+rt] : wd[8+0+rt-2];
        wd[9] := (1+rt) <= 1 ? ws[8+1+rt] : wd[8+1+rt-2];
}
else if(k==5)
{
        wd[10] := (0+rt) <= 1 ? ws[10+0+rt] : wd[10+0+rt-2];
        wd[11] := (1+rt) <= 1 ? ws[10+1+rt] : wd[10+1+rt-2];
}
else if(k==6)
{
        wd[12] := (0+rt) <= 1 ? ws[12+0+rt] : wd[12+0+rt-2];
        wd[13] := (1+rt) <= 1 ? ws[12+1+rt] : wd[12+1+rt-2];
}
else if(k==7)
{
        wd[14] := (0+rt) <= 1 ? ws[14+0+rt] : wd[14+0+rt-2];
        wd[15] := (1+rt) <= 1 ? ws[14+1+rt] : wd[14+1+rt-2];
}
```

Requirements

Header:#include<msa.h>

2.10.9. SLDI.df

2.10.9.1. v16i8 __builtin_msa_sldi_b(v16i8,v16i8,imm0_15)

Specific

Immediate Columns Slide.

v16i8 wd = __builtin_msa_sldi_b(v16i8 wd,v16i8 ws,imm0_15 n);

Return Value

n=n%16;

wd[0] := (0+n) <=15 ? ws[0+n] : wd[0+n-16];

wd[1] := (1+n) <=15 ? ws[1+n] : wd[1+n-16];

.....

wd[15] := (15+n) <=15 ? ws[15+n] : wd[15+n-16]

Requirements

Header:#include<msa.h>

2.10.9.2. v8i16 __builtin_msa_sldi_h(v8i16,v8i16,imm0_7)

Specific

Immediate Columns Slide.

v8i16 wd = __builtin_msa_sldi_h(v8i16 wd,v8i16 ws,imm0_7 n);

Return Value

n=n%8;

k=0,1

if(k==0)

```
{
    wd[0] := (0+n) <= 7 ? ws[0+n] : wd[0+n-8];
    wd[1] := (1+n) <= 7 ? ws[1+n] : wd[1+n-8];
    ......
    wd[7] := (7+n) <= 7 ? ws[7+n] : wd[7+n-8];
}
else
{
    wd[8] := (0+n)<=7 ? ws[8+0+n] : wd[8+0+n-8];
    wd[9] := (1+n)<=7 ? ws[8+1+n] : wd[8+1+n-8];
    ......
    wd[15] := (7+n)<=7 ? ws[8+7+n] : wd[8+7+n-8];
}
```

Requirements

Header:#include<msa.h>

2.10.9.3. v4i32 __builtin_msa_sldi_w(v4i32,v4i32,imm0_3)

Specific

Immediate Columns Slide.

v4i32 wd = __builtin_msa_sldi_w(v4i32 wd,v4i32 ws,imm0_3 n);

Return Value

n=n%4;

```
if ( k==0 )
{
    wd[0] := (0+n) <= 3 ? ws[0+n] : wd[0+n-4];
    wd[1] := (1+n) <= 3 ? ws[1+n] : wd[1+n-4];
    wd[2] := (2+n) <= 3 ? ws[2+n] : wd[2+n-4];
    wd[3] := (3+n) <= 3 ? ws[3+n] : wd[3+n-4];
}
else if(k==1)
{
    wd[4] := (0+n) <= 3 ? ws[4+0+n] : wd[4+0+n-4];
    wd[5] := (1+n) <= 3 ? ws[4+1+n] : wd[4+1+n-4];
    wd[6] := (2+n) <= 3 ? ws[4+2+n] : wd[4+2+n-4];
    wd[7] := (3+n) <= 3 ? ws[4+3+n] : wd[4+3+n-4];
}
else if(k==2)
{
    wd[8] := (0+n) <= 3 ? ws[8+0+n] : wd[8+0+n-4];
    wd[9] := (1+n) <= 3 ? ws[8+1+n] : wd[8+1+n-4];
    wd[10] := (2+n) <= 3 ? ws[8+2+n] : wd[8+2+n-4];
    wd[11] := (3+n) <= 3 ? ws[8+3+n] : wd[8+3+n-4];

}
```

```
else if(k==3)
{
    wd[12] := (0+n) <= 3 ? ws[12+0+n] : wd[12+0+n-4];
    wd[13] := (1+n) <= 3 ? ws[12+1+n] : wd[12+1+n-4];
    wd[14] := (2+n) <= 3 ? ws[12+2+n] : wd[12+2+n-4];
    wd[15] := (3+n) <= 3 ? ws[12+3+n] : wd[12+3+n-4];
}
```

Requirements

Header:#include<msa.h>

2.10.9.4. v2i64 __builtin_msa_sldi_d(v2i64,v2i64,imm0_1)

Specific

Immediate Columns Slide.

v2i64 wd = __builtin_msa_sldi_d(v2i64 wd,v2i64 ws,imm0_1 n);

Return Value

```
n=n%2;
if(k==0)
{
        wd[0] := (0+n) <= 1 ? ws[0+n] : wd[0+n-2];
        wd[1] := (1+n) <= 1 ? ws[1+n] : wd[1+n-2];
}
else if(k==1)
```

```
{
        wd[2] := (0+n) <= 1 ? ws[2+0+n] : wd[2+0+n-2];
        wd[3] := (1+n) <= 1 ? ws[2+1+n] : wd[2+1+n-2];
}
else if(k==2)
{
        wd[4] := (0+n) <= 1 ? ws[4+0+n] : wd[4+0+n-2];
        wd[5] := (1+n) <= 1 ? ws[4+1+n] : wd[4+1+n-2];
}
else if(k==3)
{
        wd[6] := (0+n) <= 1 ? ws[6+0+n] : wd[6+0+n-2];
        wd[7] := (1+n) <= 1 ? ws[6+1+n] : wd[6+1+n-2];
}
else if(k==4)
{
        wd[8] := (0+n) <= 1 ? ws[8+0+n] : wd[8+0+n-2];
        wd[9] := (1+n) <= 1 ? ws[8+1+n] : wd[8+1+n-2];
}
else if(k==5)
{
        wd[10] := (0+n) <= 1 ? ws[10+0+n] : wd[10+0+n-2];
```

```
        wd[11] := (1+n) <= 1 ? ws[10+1+n] : wd[10+1+n-2];
}
else if(k==6)
{
        wd[12] := (0+n) <= 1 ? ws[12+0+n] : wd[12+0+n-2];
        wd[13] := (1+n) <= 1 ? ws[12+1+n] : wd[12+1+n-2];
}
else if(k==7)
{
        wd[14] := (0+n) <= 1 ? ws[14+0+n] : wd[14+0+n-2];
        wd[15] := (1+n) <= 1 ? ws[14+1+n] : wd[14+1+n-2];
}
```

Requirements

Header:#include<msa.h>

2.10.10. VSHF.df

2.10.10.1. v16i8 __builtin_msa_vshf_b(v16i8,v16i8, v16i8)

Specific

Vector Data Preserving Shuffle.

v16i8 wd = __builtin_msa_vshf_b(v16i8 wd,v16i8 ws, v16i8 wt);

Return Value

wd[0] := vshf(wd[0],ws,wt,8);

wd[1] := vshf(wd[1],ws,wt,8);

.....

wd[15] := vshf(wd[15],ws,wt,8);

Requirements

Header:#include<msa.h>

test_vshf_b:

```
signed char a[16] = {1,2,3,4,
                     5,6,7,8,
                     9,10,11,12,
                     13,14,15,16};
signed char b[16] = {17,18,19,20,
                       21,22,23,24,
                       25,26,27,28,
                       29,30,31,32};
signed char c[16] = {0x00,0x00,0x81,0x03,
                     0x09,0x10,0x01,0x83,
                     0x17,0x30,0x07,0x25,
                     0x83,0x49,0x15,0x17};
v16i8 va,vb,vc;
va = __builtin_msa_ld_b(a,0);
vb = __builtin_msa_ld_b(b,0);
vc = __builtin_msa_ld_b(c,0);
```

vc = __builtin_msa_vshf_b(vc,vb,va);

__builtin_msa_st_b(vc,c,0);

Result:

c[16] = {1,1,0,4,

10,17,2,0,

24,17,8,6,

0,0,22,24};

Graphic example:

wt[16] = {1,2,3,4,

5,6,7,8,

9,10,11,12,

13,14,15,16};

ws[16] = {17,18,19,20,

21,22,23,24,

25,26,27,28,

29,30,31,32};

wd[16] = {0x00,0x00,0x81,0x03,

0x09,0x10,0x01,0x83,

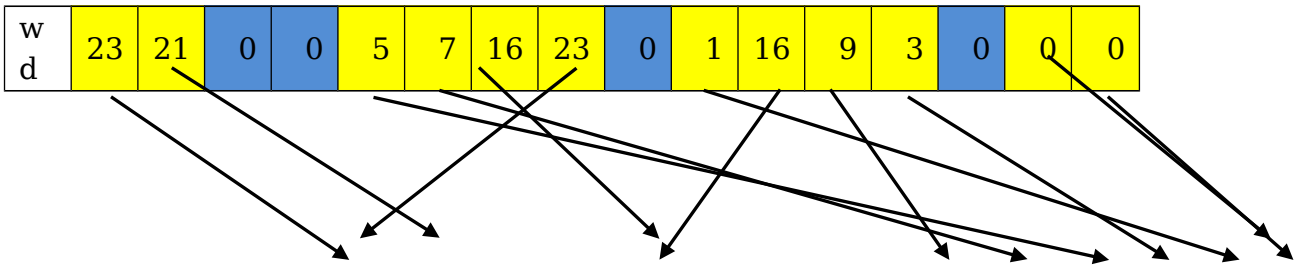
0x17,0x30,0x07,0x25,

0x83,0x49,0x15,0x17};

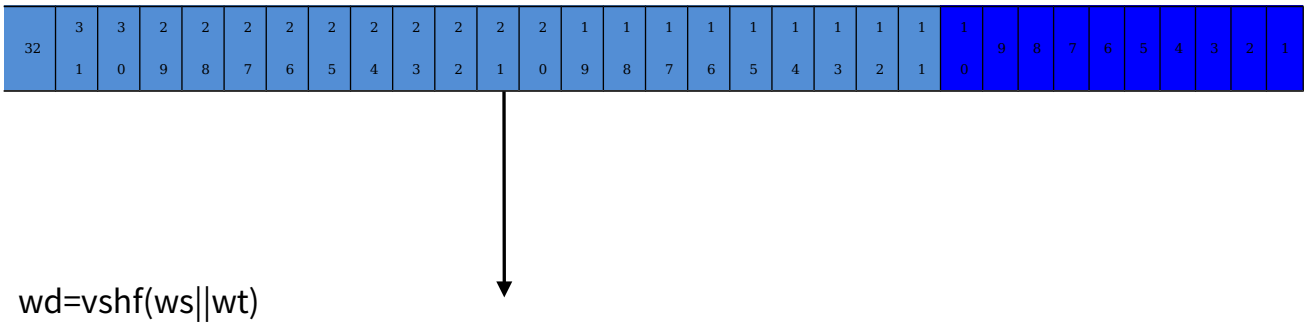
step1:

wt	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
ws	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
wd	0x17	0x15	0x49	0x83	0x25	0x07	0x30	0x17	0x83	0x01	0x10	0x09	0x03	0x81	0x00	0x00

analysis wd



ws||wt



wd=vshf(ws||wt)

wd	24	22	0	0	6	8	17	24	0	2	17	10	4	0	1	1

2.10.10.2. v8i16 __builtin_msa_vshf_h(v8i16,v8i16, v8i16)

Specific

Vector Data Preserving Shuffle.

v8i16 wd = __builtin_msa_vshf_h(v8i16 wd,v8i16 ws, v8i16 wt);

Return Value

wd[0] := vshf(wd[0],ws,wt,16);

wd[1] := vshf(wd[1],ws,wt,16);

.....

wd[8] := vshf(wd[8],ws,wt,16);

Requirements

Header:#include<msa.h>

2.10.10.3. v4i32 __builtin_msa_vshf_w(v4i32,v4i32, v4i32)

Specific

Vector Data Preserving Shuffle.

v4i32 wd = __builtin_msa_vshf_w(v4i32 wd,v4i32 ws, v4i32 wt);

Return Value

wd[0] := vshf(wd[0],ws,wt,32);

wd[1] := vshf(wd[1],ws,wt,32);

wd[2] := vshf(wd[2],ws,wt,32);

wd[3] := vshf(wd[3],ws,wt,32);

Requirements

Header:#include<msa.h>

2.10.10.4. v2i64 __builtin_msa_vshf_d(v2i64,v2i64, v2i64)

Specific

Vector Data Preserving Shuffle.

v2i64 wd = __builtin_msa_vshf_d(v2i64 wd,v2i64 ws, v2i64 wt);

Return Value

wd[0] := vshf(wd[0],ws,wt,64);

wd[1] := vshf(wd[1],ws,wt,64);

Requirements

Header:#include<msa.h>

2.10.11. VSHFR.B

2.10.11.1. v16i8 __builtin_msa_vshfr_b(v16i8,v16i8,v16i8)

Specific

Vector Data Preserving Shuffle based on the register control vector.

v16i8 wd = __builtin_msa_vshfr_b(v16i8 ws,v16i8 wt,v16i8 wr);

Return Value

wd[0] := vshf(wr[0],ws,wt,8);

wd[1] := vshf(wr[1],ws,wt,8);

.....

wd[15] := vshf(wr[15],ws,wt,8);

Requirements

Header:#include<msa.h>

2.10.12. VSLDI.B

2.10.12.1. v16i8 __builtin_msa_vsldi_b(v16i8,v16i8,imm0_15)

Specific

Immediate Vector Slide.

v16i8 wd = __builtin_msa_vsldi_b(v16i8 ws,v16i8 wt,imm0_15 n);

Return Value

n=n%16;

wd[0] := (0+n) <=15 ? ws[0+n] : wt[0+n-16];

wd[1] := (1+n) <=15 ? ws[1+n] : wt[1+n-16];

.....

wd[15] := (15+n) <=15 ? ws[15+n] : wt[15+n-16]

Requirements

Header:#include<msa.h>

Graphic example:

wt[16]={0,1,2,3,

4,5,6,7,

8,9,10,11,

12,13,14,15};

ws[16]={16,17,18,19,

20,21,22,23,

24,25,26,27,

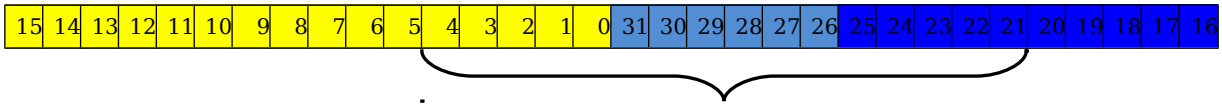
28,29,30,31};

n=5;

step1:

wt	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ws	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16

wt||ws



wd	4	3	2	1	0	31	30	29	28	27	26	25	24	23	22	21

Result:

wd[16]={21,22,23,24,

25,26,27,28,

```
29,30,31,0,
1,2,3,4};
```

3. Append

3.1. abs(x,n)

```
/*Rerurn the absolute value of integer x.*/
a_sign=0;
if(n==8)
    a_sign= x & 0x80;
if(n==16)
    a_sign= x & 0x8000
if(n==32)
    a_sign= x & 0x8000 0000
if(n==64)
    a_sign = x & 0x8000 0000 0000 0000
return a_sign ? -x : x;
```

3.2. fabs(x,n)

```
/*Rerurn the absolute value of floating-point x.*/
z=0;
sign=0;
if(n==32)
```

```
{
    if((x & 0x007f ffff)==0) // x=0
    {
        z = 0x0;
    }
    else
    {
        sign = ((x & 0x80000 0000 ) !=0);
        if(sign)
            z = x ^ 0x8000 0000;
        else
            z = x;
    }
    return z;
}
else if(n==64)
{
    if ((x & 0x000f ffff ffff ffff)==0) //x=0
    {
        z = 0x0;
    }
    else
```

```
    {
        sign = ((x & 0x8000 0000 0000 0000)!=0);
        if(sign)
            z = x ^ 0x80000 0000 0000 0000;
        else
            z = x;
    }
    return z;
}
```

3.3. sats(x,n)

```
/*Return the saturated signed value of signed x.*/
sign0 = (x >> n) & 0x1;
sign1 = x >> (n-1);
z=0;
if ((sign0==0)&&(sign1!=0))
{
    if(n==8)
        z=0x7f;
    if(n==16)
        z=0x7fff;
```

```
    if(n==32)
        z=0x7fff ffff;
    if(n==64)
        z=0x7fff ffff ffff ffff;
}
else if ((sign0==1)&&(sign1!=1))
{
    if(n==8)
        z=0x80;
    if(n==16)
        z=0x8000;
    if(n==32)
        z=0x8000 0000;
    if(n==64)
        z=0x8000 0000 0000 0000;
}
else
    z = x;
return z;
```

3.4. sats_fixed(x,n)

```
sign0=0;
```

```
sign1=0;
z=0;
if(n==8)
{
    sign0 = x & 0x8000;
    sign1 = x & 0x7f80;
    if(!sign0 && sign1)
        z = 0x7f;
    else if(sign0 && sign1)
        z = 0x80;
    else if(sign0 && !sign1)
        z = (x & 0x7f) | 0x80;
    else
        z = x & 0x7f;
}
else if(n==16)
{
    sign0 = x & 0x8000 0000;
    sign1 = x & 0x7fff 8000 ;
    if(!sign0 && sign1)
        z = 0x7fff;
    else if(sign0 && sign1)
```

```
        z = 0x8000;
    else if(sign0 && !sign1)
        z = (x & 0x7fff) | 0x8000;
    else
        z = x & 0x7fff;
}
return z;
```

3.5. sat_s(x,i,n)

```
/*Return the signed x saturated to signed values of i+1 bits without
changing the data width.*/
z = 0;
sign0=0;
sign1=0;
if(n==8)
{
    sign0 = x & 0x80;
    sign1 = x >> (i-1);
    if((!sign0)&&(sign1))
    {
        if(i==1)
            z=0;
```

```
            else
                z = 0xff >> (n-i+1)
        }
        else if(sign0 && sign1 !=(0xff>>(i-1)))
            z = 0xff << (i-1)
        else
            z = a;
    }
    else if(n==16)
    {
        sign0 = x & 0x8000;
        sign1 = x >> (i-1);
        if((!sign0)&&(sign1))
        {
            if(i==1)
                z=0;
            else
                z = 0xffff >> (n-i+1)
        }
        else if(sign0 && sign1 !=(0xffff>>(i-1)))
            z = 0xffff << (i-1)
        else
```

```
            z = a;

}
else if(n==32)
{
      sign0 = x & 0x8000 0000;
      sign1 = x >> (i-1);
      if((!sign0)&&(sign1))
      {
            if(i==1)
                  z=0;
            else
                  z = 0xffff ffff >> (n-i+1)
      }
      else if(sign0 && sign1 !=(0xffff ffff>>(i-1)))
            z = 0xffff ffff<< (i-1)
      else
            z = a;

}
else if(n==64)
{
      sign0 = x & 0x8000 0000 0000 0000;
```

```
    sign1 = x >> (i-1);
    if((!sign0)&&(sign1))
    {
        if(i==1)
            z=0;
        else
            z = 0xffff ffff ffff ffff>> (n-i+1)
    }
    else if(sign0 && sign1 !=(0xffff ffff ffff ffff>>(i-1)))
        z = 0xffff ffff ffff ffff<< (i-1)
    else
        z = a;

}
return z;
```

3.6. satu(x)

```
/*Return the saturated unsigned value of unsigned x.*/
if(x<0)
    return 0;
else
    return x;
```

3.7. satu_fixed(x,n)

```
sign=0;
z = 0;
if(n==8)
{
    sign = x & 0xff00;
    if(sign)
        z = 0xff;
    else
    z = x;
}
else if(n==16)
{
    sign = x & 0xffff 0000;
    if(sign)
        z = 0xffff;
    else
        z = x;
}
return z;
```

3.8. sat_u(x,i,n)

```
/*Return the unsigned x saturated to unsigned values of i+1 without
changing the data width.*/
z=0;
if(n==8)
{
    if (i==8)
        z=x;
    else if(x>>i)
        z=0xff >> (n-i);
    else
    z=x;

}
else if(n==16)
{
    if (i==16)
        z=x;
    else if(x>>i)
        z=0xffff >> (n-i);
    else
```

```
        z=x;

}
else if(n==32)
{
        if (i==32)
            z=x;
        else if(x>>i)
            z=0xffff ffff >> (n-i);
        else
        z=x;

}
else if(n==64)
{
        if (i==64)
            z=x;
        else if(x>>i)
            z=0xffff ffff ffff ffff >> (n-i);
        else
        z=x;
}
return z;
```

3.9. subsus_u (x,y,n)

```
sign = 0;
z = 0;
diff = x - y;
if(n==8)
{
    sign=y&0x80;
    if(x<y)
        z = 0x0;
    else if((x > (diff & 0xff))&&(sign))
        z = 0xff;
    else
        z = diff;
}
else if(n==16)
{
    sign=y&0x8000;
    if(x<y)
        z = 0x0;
    else if((x > (diff & 0xffff))&&(sign))
        z = 0xffff;
```

```
        else
            z = diff;

    }
    else if(n==32)
    {
        sign=y&0x8000 0000;
        if(x<y)
            z = 0x0;
        else if((x > (diff & 0xffff ffff))&&(sign))
            z = 0xffff ffff;
        else
            z = diff;
    }
    else if(n==64)
    {
        sign=y&0x8000 0000 0000 0000;
        if(x<y)
            z = 0x0;
        else if((x > (diff & 0xffff ffff ffff ffff))&&(sign))
            z = 0xffff ffff ffff ffff;
        else
```

```
        z = diff;
}
return z;
```

3.10. round_add(x,y,n)

```
sign=0;
z=0;
d=0;
if(n==16)
{
    sign=x & 0x8000;
    if(!sign)
    {
        d = ((uint32_t)x<<(n-1)) + y;
    }
    else
    {
        d =((uint32_t)(0x1 0000|x)<<(n-1))+y;
    }
    d = d + 0x4000;
    z = d >>(n-1);
}
```

```
else if (n==32)
{
    sign=x & 0x8000 0000;
    if(!sign)
    {
        d = ((uint64_t)x<<(n-1)) + y;
    }
    else
    {
        d =((uint64_t)(0x1 0000 0000|x)<<(n-1))+y;
    }
    d = d + 0x4000 0000;
    z = d >>(n-1);
}
return z;
```

3.11. round_sub(x,y,n)

```
sign=0;
z=0;
d=0;
if(n==16)
{
```

```
        sign=x & 0x8000;
        if(!sign)
        {
            d = ((uint32_t)x<<(n-1)) - y;
        }
        else
        {
            d =((uint32_t)(0x1 0000|x)<<(n-1))-y;
        }
        d = d + 0x4000;
        z = d >>(n-1);
    }
    else if (n==32)
    {
        sign=x & 0x8000 0000;
        if(!sign)
        {
            d = ((uint64_t)x<<(n-1)) - y;
        }
        else
        {
            d =((uint64_t)(0x1 0000 0000|x)<<(n-1)) - y;
```

```
    }
    d = d + 0x4000 0000;
    z = d >>(n-1);
}
return z;
```

3.12. round_mul(x,y,n)

```
z=0;
if(n==16)
{
    if((x==0x8000)&&(y==0x8000))
        z = 0x7fff;
    else
    {
        z = (uint32_t)((uint32_t)x * (uint32_t)y) + 0x4000;
        z = (z>>(n-1))&0xffff;
    }
}
else if(n==32)
{
    if((x==0x8000 0000)&&(y==0x8000 0000))
        z = 0x7fff ffff;
```

```
    else
    {
        z = (uint64_t)((uint64_t)x * (uint64_t)y) + 0x4000 0000;
        z = (z>>(n-1))&0xffff ffff;
    }
}
return z;
```

3.13. srar (x,y,n)

```
z=0;
shift=0;
round = 0;
sign=0;
if(n==8)
{
    shift = y & 0x7;
    if(shift==0)
        z=x;
    else
    {
        round = (x & (0x1 << (shift-1)) ) >> (shift-1);
        sign = x & 0x80;
```

```
            if(sign)
                z = (~(0xff>>shift))|((x >> shift)+round);
            else
                z = (x>>shift) + round;
        }
    }
    else if(n==16)
    {
        shift = y & 0xf;
        if(shift==0)
            z=x;
        else
        {
            round = (x & (0x1 << (shift-1)) )>>(shift-1);
            sign = x & 0x8000;
            if(sign)
                z = (~(0xffff >> shift))|((x >> shift)+round);
            else
                z = (x>>shift) + round;
        }
    }
    else if(n==32)
```

```
{
    shift = y & 0x1f;
    if(shift==0)
        z=x;
    else
    {
        round = (x & (0x1 << (shift-1)) )>>(shift-1);
        sign = x & 0x8000 0000;
        if(sign)
            z = (~(0xffff ffff>>shift))|((x >> shift)+round);
        else
            z = (x>>shift) + round;
    }
}
else if(n==64)
{
    shift = y & 0x3f;
    if(shift==0)
        z=x;
    else
    {
        round = (x & (0x1 <<(shift-1)) )>>(shift-1);
```

```
        sign = x & 0x8000 0000 0000 0000;
        if(sign)
            z = (~(0xffff ffff ffff ffff>>shift))|((x >> shift)+round);
        else
            z = (x>>shift) + round;
    }
}
return z;
```

3.14. srlr (x,y,n)

```
z=0;
shift=0;
round = 0;
if(n==8)
{
    shift = y & 0x7;
    if(shift==0)
        z=x;
    else
    {
        round = (x & (0x1 << (shift-1)) ) >> (shift-1);
        z = (x>>shift) + round;
```

```
        }
    }
    else if(n==16)
    {
        shift = y & 0xf;
        if(shift==0)
            z=x;
        else
        {
            round = (x & (0x1 << (shift-1)) )>>(shift-1);
            z = (x>>shift) + round;
        }
    }
    else if(n==32)
    {
        shift = y & 0x1f;
        if(shift==0)
            z=x;
        else
        {
            round = (x & (0x1 << (shift-1)) )>>(shift-1);
            z = (x>>shift) + round;
```

```
    }
}
else if(n==64)
{
    shift = y & 0x3f;
    if(shift==0)
        z=x;
    else
    {
        round = (x & (0x1 <<(shift-1)) )>>(shift-1);
        z = (x>>shift) + round;
    }
}
return z;
```

3.15. binsl(x,y,d,n)

```
z=0;
k=0;
if(n==8)
{
    y = 7 - y&0x7;
    k = 8 - y;
```

```
}
else if(n==16)
{
      y =15 - y&0xf;
      k = 16 - y;
}
else if(n==32)
{
      y =31 - y&0x1f;
      k = 32 - y;
}
else if(n==64)
{
      y =63 - y&0x3f;
      k = 64 - y;
}
if(y==0)
            z= x;
else
            z = ((x >> y)<<y)|((d<<k)>>k);
return z;
```

3.16. binsr(x,y,d,n)

```
z=0;
k=0;
if(n==8)
{
      y = y&0x7 + 1;
      k = 8 - y;
}
else if(n==16)
{
      y = y&0xf + 1;
      k = 16 - y;
}
else if(n==32)
{
      y = y&0x1f + 1;
      k = 32 - y;
}
else if(n==64)
{
      y =y&0x3f + 1;
```

```
    k = 64 - y;
}
if(k==0)
        z= x;
else
        z = ((d >> y)<<y)|((x<<k)>>k);
return z;
```

3.17. shf_index(a,i)

```
j=0;
k=0;
j=i % 4;
k=i - j + ((a>>(2*j)) & 0x3) ;
return k;
```

3.18. vshf(c,a[16],b[16],i)

```
temp = 128 / i;
t1 = c & 0xc0;
if (t1)
    return 0;
else
    t2 = (c & 0x3f)%temp;
```

```
    if (t2 >= (temp/2))
        return a[t2 - temp/2];
    else
        return  b[t2];
```

3.19. loc(x, n)

```
t = 0;
for(k = n-1; k >= 0; k--)
{
    t = x >> k;
    t = t & 0x1;
    if(t == 0)
        break;
}
return (n-k-1);
```

3.20. lzc(x, n)

```
t = 0;
for(k = n-1; k >= 0; k--)
{
    t = x >> k;
    t= t & 0x1;
    if(t == 1)
        break;
}
return (n-k-1);
```

3.21. pcnt(x, n)

```
count=0;
k=0;
t=0;
for(k = n-1; k >= 0; k--)
{
    t = x >> k;
    t = t & 0x1;
    if(t == 1)
        count++;
}
return count;
```

3.22. bmu_v (a, b,c)

```
i=0;
j=0;
z=b;
max = c & 0xf;
```

$b1=b_{max....0}$

$z1=z_{max....0}$

$a1=a_{max....0}$

```
for(i=0;i<=max;i++)
{
    z=z&(~(0x1<<i));
    if(b1&(0x1<<i))
    {
        z1   &= ~(0x1<<i);//clear z1 bit[i]
        z1 |= (((a1&(0x1<<j))>>j)<<i);
        j=j+1;
    }
}
z=z|z1;
return z;
```

3.23. extractFloatSign(x,n)

```
z=0;
if(n==32)
    z = x>>31;
else if(n==64)
    z = x>>63;
return z;
```

3.24. extractFloatExp(x,n)

```
z=0;

if(n==32)

    z = (x >> 23) & 0xFF;

else if(n==64)

    z = (x >> 52) & 0x7FF;

return z;
```

3.25. extractFloatFrac (x,n)

```
z=0;

if(n==32)

    z = x & 0x007f ffff;

else if(n==64)

    z = x & 0x000f ffff ffff ffff;

return z;
```

3.26. MSAroundingMode(aSign)

```
{
    increment = 0;
    roundingMode = __builtin_msa_cfcmsa(1) & 0x3;
    switch(roundingMode)
    {
        case 0:/*float round nearest*/
            increment = 0x1000;
            break;

    case 2:/* float round up*/
```

```
            increment = aSign ? 0 : 0x1fff;
            break;

        case 3:/*float round down*/
            increment = aSign ? 0x1fff : 0;
            break;
        default:
            increment = 0;
            break;
        }
        return increment;
}
```

3.27. sqrt(x)

/*Return the square value of x.*/

1. On success ,these functions return the square root of x;

2. If x is NaN,a NaN is returned.

3. If x is +0 (-0),+0 (-0) is returned.

4. If x is positive infinity ,positive infinity is returned.

5. If x is less than -0, a domain error occurs,and a NaN is returned.

3.28. fclass(x,n)

/*Return the class mask value of x.*/

aSign = extractFloatSign(x,n);

aExp = extractFloatExp (x,n);

aSig = extractFloatFrac (x,n);

mask = 0;

if(n==32)

```
{
    if(aExp==0xFF)
    {
        if(aSig)/*qNaN or sNaN*/
            mask |= (aSig>>22) + 1;
        else/*INF*/
            mask | = (aSign ? 0x004 : 0x040);
    }
    else if(aExp)/*normal*/
        mask |= (aSign ? 0x008 : 0x080);
    else
    {
        if(aSig)/*subnormal*/
            mask |=(aSign ? 0x010 : 0x100);
        else/*0*/
            mask |= (aSign ? 0x020 : 0x200);
    }
}
else if(n==64)
{
    if(aExp==0x7FF)
    {
```

```
        if(aSig)/*qNaN or sNaN*/
            mask |= (aSig>>51) + 1;
        else/*INF*/
            mask | = (aSign ? 0x004 : 0x040);
    }
    else if(aExp)/*normal*/
        mask |= (aSign ? 0x008 : 0x080);
    else
    {
        if(aSig)/*subnormal*/
            mask |=(aSign ? 0x010 : 0x100);
        else/*0*/
            mask |= (aSign ? 0x020 : 0x200);
    }
}
return mask;
```

3.29. quite_FALSE(tt,ts,n)

```
    /*Implementation defined signaling NAN test*/
    return 0;
```

3.30. signaling_FALSE(tt,ts,n)

```
    /*Implementation defined signaling NAN test*/
```

return 0;

3.31. complex_mul (ts[2],tt[2])

td[2]={0,0};

td[0] = ts[0]*tt[0] - ts[1]*tt[1];

td[1] = ts[0]*tt[1] + ts[1]*tt[0];

return td;

3.32. half_float_convert(ts,tt)

/*ts format is 16-bit XHS,f is IEEE standard 32-bit floating-point data,the tt format is 16-bit control value*/

bias=(tt&0xfff0)>>4;

ebits=tt&0xf;

f=up_covert_xhs(ts, ebits, bias);/*Implementation XHS format up-conversion*/

return f;

3.33. memory (addr)

1.load the value of the memory address addr.

2.store the value to the memory address addr.

3.34. SignalException(AddressError)

Causes AddressError to be signaled. Control does not return from this pseudocode function--the exception is signaled at the point of the call.

4. MSA Code Examples with Optimization

4.1. Converting corlor depth

4.2. Image history update

4.3. FIR filter