Xuantie 900 Series RVM-0.3 Intrinsic Manual

Version 0.3, 03/2024

Table of Contents

Copyright

Hangzhou C-SKY MicroSystems Co., LTD

Address: Room 201, 2/F, Building 5, No.699 Wangshang Road , Hangzhou, Zhejiang, China

Website: www.xrvm.cn

Contributors

This RISC-V specification has been contributed to directly or indirectly by:

Jin Ma <jinma@linux.alibaba.com>

Zixuan Wu <zixuan.wu@linux.alibaba.com>

Xianmiao Qu <cooper.qu@linux.alibaba.com>

CunJian Huang <huangcunjian.huang@alibaba-inc.com>

Chapter 1. Introduction

This document introduces the intrinsics for RISC-V matrix programming, including the general naming rules for intrinsics, the data types of matrix, and the full set of intrinsics.

Chapter 2. Naming Rules

- Data type naming rules: Prefix the basic data type with 'm'. Additionally, use the suffix "x2" to denote a pair of registers, exemplified by "mint8x2_t".
- Function interface naming rules: For simplicity, the interface is named by the instruction name and the prefix "__riscv_th_", while mm, mv, and mx are used to distinguish the source operands as matrix-matrix, matrix-vector, and matrix-scalar.

Chapter 3. Data Types

Table 1. Matrix data types

Type	Name	Description
matrix int	mint8_t	All elements of the matrix are int8_t
	mint16_t	
	mint32_t	
	mint64_t	
	muint8_t	All elements of the matrix are uint8_t
	muint16_t	
	muint32_t	
	muint64_t	
matrix float	mfloat16_t	All elements of the matrix are float16_t
	mfloat32_t	
	mfloat64_t	
matrix int x2	mint8x2_t	The type is denoted by a pair of sequential matrix registers, with the initial register being an even number.
	mint16x2_t	
	mint32x2_t	
	mint64x2_t	
	muint8x2_t	
	muint16x2_t	
	muint32x2_t	
	muint64x2_t	
matrix float x2	mfloat16x2_t	
	mfloat32x2_t	
	mfloat64x2_t	
matrix row num	mrow_t	The type is actually size_t, which represents the number of matrix rows. Only the lower 8 bits of this type are valid.
matrix column num	mcol_t	The type is actually size_t, which represents the number of matrix columns. Only the lower 16 bits of this type are valid.

Chapter 4. Intrinsic interface

4.1. Configuration instructions:

Instructions

```
mcfgi<m/n/k>  rd,uimm7
mcfg<m/n/k>    rd,rs1
```

Intrinsic functions list

```
mrow_t __riscv_th_msetmrow_m (mrow_t m);
mrow_t __riscv_th_msetmrow_n (mrow_t n);
mcol_t __riscv_th_msetmcol_e8 (mcol_t c);
mcol_t __riscv_th_msetmcol_e16 (mcol_t c);
mcol_t __riscv_th_msetmcol_e32 (mcol_t c);
mcol_t __riscv_th_msetmcol_e64 (mcol_t c);
```

Set m, n, and k in the CSR `msize` to return valid values. When the value of any parameter exceeds the maximum allowable setting, only the lower bits are considered valid: 8 bits for m, 8 bits for n, and 16 bits for k.

4.2. Read/Write Matrix CSRs

Instructions

```
csrr    rd,xmrstart
csrr    rd,xmcsr
csrr    rd,xmsize
csrr    rd,xmlenb
csrr    rd,xrlenb
csrr    rd,xmisa

csrw    xmrstart,rs1
csrw    xmcsr,rs1
csrw    xmsize,rs1
```

Intrinsic functions list

```
enum RVM_CSR {
  RVM_XMRSTART = 0,
  RVM_XMCSR,
  RVM_XMSIZE,
  RVM_XMLENB,
```

```
  RVM_XRLENB,
  RVM_XMISA
};

unsigned long __riscv_th_mread_csr(enum RVM_CSR csr);
void __riscv_th_mwrite_csr(enum RVM_CSR csr, unsigned long value)

// specialization version
unsigned long __riscv_th_xmlenb();
unsigned long __riscv_th_xrlenb();
unsigned long __riscv_th_xmsize();
```

4.3. Undefined

Intrinsic functions list

```
mint8_t __riscv_th_mundefined_i8 ();
mint16_t __riscv_th_mundefined_i16 ();
mint32_t __riscv_th_mundefined_i32 ();
mint64_t __riscv_th_mundefined_i64 ();
muint8_t __riscv_th_mundefined_u8 ();
muint16_t __riscv_th_mundefined_u16 ();
muint32_t __riscv_th_mundefined_u32 ();
muint64_t __riscv_th_mundefined_u64 ();
mfloat16_t __riscv_th_mundefined_f16 ();
mfloat32_t __riscv_th_mundefined_f32 ();
mfloat64_t __riscv_th_mundefined_f64 ();

mint8x2_t __riscv_th_mundefined_i8x2 ();
mint16x2_t __riscv_th_mundefined_i16x2 ();
mint32x2_t __riscv_th_mundefined_i32x2 ();
mint64x2_t __riscv_th_mundefined_i64x2 ();
muint8x2_t __riscv_th_mundefined_u8x2 ();
muint16x2_t __riscv_th_mundefined_u16x2 ();
muint32x2_t __riscv_th_mundefined_u32x2 ();
muint64x2_t __riscv_th_mundefined_u64x2 ();
mfloat16x2_t __riscv_th_mundefined_f16x2 ();
mfloat32x2_t __riscv_th_mundefined_f32x2 ();
mfloat64x2_t __riscv_th_mundefined_f64x2 ();
```

4.4. Reinterpret Cast Conversion Functions

Intrinsic functions list

```
mint8_t __riscv_th_mreinterpret_i8 (src);
mint16_t __riscv_th_mreinterpret_i16 (src);
mint32_t __riscv_th_mreinterpret_i32 (src);
```

```
mint64_t __riscv_th_mreinterpret_i64 (src);
muint8_t __riscv_th_mreinterpret_u8 (src);
muint16_t __riscv_th_mreinterpret_u16 (src);
muint32_t __riscv_th_mreinterpret_u32 (src);
muint64_t __riscv_th_mreinterpret_u64 (src);
mfloat16_t __riscv_th_mreinterpret_f16 (src);
mfloat32_t __riscv_th_mreinterpret_f32 (src);
mfloat64_t __riscv_th_mreinterpret_f64 (src);

mint8x2_t __riscv_th_mreinterpret_i8x2 (src);
mint16x2_t __riscv_th_mreinterpret_i16x2 (src);
mint32x2_t __riscv_th_mreinterpret_i32x2 (src);
mint64x2_t __riscv_th_mreinterpret_i64x2 (src);
muint8x2_t __riscv_th_mreinterpret_u8x2 (src);
muint16x2_t __riscv_th_mreinterpret_u16x2 (src);
muint32x2_t __riscv_th_mreinterpret_u32x2 (src);
muint64x2_t __riscv_th_mreinterpret_u64x2 (src);
mfloat16x2_t __riscv_th_mreinterpret_f16x2 (src);
mfloat32x2_t __riscv_th_mreinterpret_f32x2 (src);
mfloat64x2_t __riscv_th_mreinterpret_f64x2 (src);
```

The type of SRC can be any matrix type with the same number of registers.

4.5. Mzero

Instructions

```
mzero rd
```

Intrinsic functions list

```
mint8_t __riscv_th_mzero_i8 ();
mint16_t __riscv_th_mzero_i16 ();
mint32_t __riscv_th_mzero_i32 ();
mint64_t __riscv_th_mzero_i64 ();
muint8_t __riscv_th_mzero_u8 ();
muint16_t __riscv_th_mzero_u16 ();
muint32_t __riscv_th_mzero_u32 ();
muint64_t __riscv_th_mzero_u64 ();
mfloat16_t __riscv_th_mzero_f16 ();
mfloat32_t __riscv_th_mzero_f32 ();
mfloat64_t __riscv_th_mzero_f64 ();

mint8x2_t __riscv_th_mzero_i8x2 ();
mint16x2_t __riscv_th_mzero_i16x2 ();
mint32x2_t __riscv_th_mzero_i32x2 ();
mint64x2_t __riscv_th_mzero_i64x2 ();
```

```
muint8x2_t __riscv_th_mzero_u8x2 ();
muint16x2_t __riscv_th_mzero_u16x2 ();
muint32x2_t __riscv_th_mzero_u32x2 ();
muint64x2_t __riscv_th_mzero_u64x2 ();
mfloat16x2_t __riscv_th_mzero_f16x2 ();
mfloat32x2_t __riscv_th_mzero_f32x2 ();
mfloat64x2_t __riscv_th_mzero_f64x2 ();
```

Zero all elements of matrix register.

4.6. Load and store instructions

4.6.1. Load

Instructions

```
#matrix load
mld<b/h/w/d> md, rs2, (rs1)

#stream matrix load
msld<b/h/w/d>  md, rs2, (rs1)
```

Intrinsic functions list

```
//matrix load
mint8_t __riscv_th_mld (const int8_t *base, long stride, mrow_t row, mcol_t col);
muint8_t __riscv_th_mld (const uint8_t *base, long stride, mrow_t row, mcol_t col);
mint16_t __riscv_th_mld (const int16_t *base, long stride, mrow_t row, mcol_t col);
muint16_t __riscv_th_mld (const uint16_t *base, long stride, mrow_t row, mcol_t col);
mint32_t __riscv_th_mld (const int32_t *base, long stride, mrow_t row, mcol_t col);
muint32_t __riscv_th_mld (const uint32_t *base, long stride, mrow_t row, mcol_t col);
mint64_t __riscv_th_mld (const int64_t *base, long stride, mrow_t row, mcol_t col);
muint64_t __riscv_th_mld (const uint64_t *base, long stride, mrow_t row, mcol_t col);
mfloat16_t __riscv_th_mld (const float16_t *base, long stride, mrow_t row, mcol_t
col);
mfloat32_t __riscv_th_mld (const float32_t *base, long stride, mrow_t row, mcol_t
col);
mfloat64_t __riscv_th_mld (const float64_t *base, long stride, mrow_t row, mcol_t
col);

//stream matrix load
mint8_t __riscv_th_msld (const int8_t *base, long stride, mrow_t row, mcol_t col);
muint8_t __riscv_th_msld (const uint8_t *base, long stride, mrow_t row, mcol_t col);
mint16_t __riscv_th_msld (const int16_t *base, long stride, mrow_t row, mcol_t col);
muint16_t __riscv_th_msld (const uint16_t *base, long stride, mrow_t row, mcol_t col);
mint32_t __riscv_th_msld (const int32_t *base, long stride, mrow_t row, mcol_t col);
muint32_t __riscv_th_msld (const uint32_t *base, long stride, mrow_t row, mcol_t col);
```

```
mint64_t __riscv_th_msld (const int64_t *base, long stride, mrow_t row, mcol_t col);
muint64_t __riscv_th_msld (const uint64_t *base, long stride, mrow_t row, mcol_t col);
mfloat16_t __riscv_th_msld (const float16_t *base, long stride, mrow_t row, mcol_t
col);
mfloat32_t __riscv_th_msld (const float32_t *base, long stride, mrow_t row, mcol_t
col);
mfloat64_t __riscv_th_msld (const float64_t *base, long stride, mrow_t row, mcol_t
col);
```

Read from the memory to the matrix register: The input parameter is the memory base address, stride, and the return value is the target matrix.

4.6.2. Store

Instructions

```
#matrix store
mst<b/h/w/d>  ms3, rs2, (rs1)

#stream matrix store
msst<b/h/w/d>  ms3, rs2, (rs1)
```

Intrinsic functions list

```
//matrix store
void __riscv_th_mst (const int8_t *base, long stride, mint8_t value, mrow_t row,
mcol_t col);
void __riscv_th_mst (const uint8_t *base, long stride, muint8_t value, mrow_t row,
mcol_t col);
void __riscv_th_mst (const int16_t *base, long stride, mint16_t value, mrow_t row,
mcol_t col);
void __riscv_th_mst (const uint16_t *base, long stride, muint16_t value, mrow_t row,
mcol_t col);
void __riscv_th_mst (const int32_t *base, long stride, mint32_t value, mrow_t row,
mcol_t col);
void __riscv_th_mst (const uint32_t *base, long stride, muint32_t value, mrow_t row,
mcol_t col);
void __riscv_th_mst (const int64_t *base, long stride, mint64_t value, mrow_t row,
mcol_t col);
void __riscv_th_mst (const uint64_t *base, long stride, muint64_t value, mrow_t row,
mcol_t col);
void __riscv_th_mst (const float16_t *base, long stride, mfloat16_t value, mrow_t row,
mcol_t col);
void __riscv_th_mst (const float32_t *base, long stride, mfloat32_t value, mrow_t row,
mcol_t col);
void __riscv_th_mst (const float64_t *base, long stride, mfloat64_t value, mrow_t row,
mcol_t col);
```

```
//stream matrix store
void __riscv_th_msst (const int8_t *base, long stride, mint8_t value, mrow_t row,
mcol_t col);
void __riscv_th_msst (const uint8_t *base, long stride, muint8_t value, mrow_t row,
mcol_t col);
void __riscv_th_msst (const int16_t *base, long stride, mint16_t value, mrow_t row,
mcol_t col);
void __riscv_th_msst (const uint16_t *base, long stride, muint16_t value, mrow_t row,
mcol_t col);
void __riscv_th_msst (const int32_t *base, long stride, mint32_t value, mrow_t row,
mcol_t col);
void __riscv_th_msst (const uint32_t *base, long stride, muint32_t value, mrow_t row,
mcol_t col);
void __riscv_th_msst (const int64_t *base, long stride, mint64_t value, mrow_t row,
mcol_t col);
void __riscv_th_msst (const uint64_t *base, long stride, muint64_t value, mrow_t row,
mcol_t col);
void __riscv_th_msst (const float16_t *base, long stride, mfloat16_t value, mrow_t
row, mcol_t col);
void __riscv_th_msst (const float32_t *base, long stride, mfloat32_t value, mrow_t
row, mcol_t col);
void __riscv_th_msst (const float64_t *base, long stride, mfloat64_t value, mrow_t
row, mcol_t col);
```

Write the matrix register data into the memory, and the input parameter is the destination base address, stride, and the original operand.

4.7. Mov instructions

Instructions

```
#matrix-matrix mov
mmov.mm md, ms1

#matrix-vector add,rs1'/uimm3
mmov.mv.x md, ms1[rs1']
mmov.mv.i md, ms1[uimm3]

#matrix-scalar mov with duplicate
mdup<b/h/w/d>.m.x md, rs2

#matrix-scalar mov
mmov<b/h/w/d>.m.x md, rs2, rs1

mmov<b/h/w/d>.x.m rd, ms2, rs1
```

Intrinsic functions list

```
//matrix-vector mov,rs1/uimm3
mint8_t __riscv_th_mmov_mv (mint8_t src, size_t index);
muint8_t __riscv_th_mmov_mv (muint8_t src, size_t index);
mint16_t __riscv_th_mmov_mv (mint16_t src, size_t index);
muint16_t __riscv_th_mmov_mv (muint16_t src, size_t index);
mint32_t __riscv_th_mmov_mv (mint32_t src, size_t index);
muint32_t __riscv_th_mmov_mv (muint32_t src, size_t index);
mint64_t __riscv_th_mmov_mv (mint64_t src, size_t index);
muint64_t __riscv_th_mmov_mv (muint64_t src, size_t index);
mfloat16_t __riscv_th_mmov_mv (mfloat16_t src, size_t index);
mfloat32_t __riscv_th_mmov_mv (mfloat32_t src, size_t index);
mfloat64_t __riscv_th_mmov_mv (mfloat64_t src, size_t index);

// matrix-scalar mov with duplicate
mint8_t __riscv_th_mdup_m_x (int8_t src);
muint8_t __riscv_th_mdup_m_x (uint8_t src);
mint16_t __riscv_th_mdup_m_x (int16_t src);
muint16_t __riscv_th_mdup_m_x (uint16_t src);
mint32_t __riscv_th_mdup_m_x (int32_t src);
muint32_t __riscv_th_mdup_m_x (uint32_t src);
mint64_t __riscv_th_mdup_m_x (int64_t src);
muint64_t __riscv_th_mdup_m_x (uint64_t src);

// matrix-scalar mov
mint8_t __riscv_th_mmov_m_x (mint8_t dest, int8_t src, size_t index);
muint8_t __riscv_th_mmov_m_x (muint8_t dest, uint8_t src, size_t index);
mint16_t __riscv_th_mmov_m_x (mint16_t dest, int16_t src, size_t index);
muint16_t __riscv_th_mmov_m_x (muint16_t dest, uint16_t src, size_t index);
mint32_t __riscv_th_mmov_m_x (mint32_t dest, int32_t src, size_t index);
muint32_t __riscv_th_mmov_m_x (muint32_t dest, uint32_t src, size_t index);
mint64_t __riscv_th_mmov_m_x (mint64_t dest, int64_t src, size_t index);
muint64_t __riscv_th_mmov_m_x (muint64_t dest, uint64_t src, size_t index);

int8_t __riscv_th_mmov_x_m (mint8_t src, size_t index);
uint8_t __riscv_th_mmov_x_m (muint8_t src, size_t index);
int16_t __riscv_th_mmov_x_m (mint16_t src, size_t index);
uint16_t __riscv_th_mmov_x_m (muint16_t src, size_t index);
int32_t __riscv_th_mmov_x_m (mint32_t src, size_t index);
uint32_t __riscv_th_mmov_x_m (muint32_t src, size_t index);
int64_t __riscv_th_mmov_x_m (mint64_t src, size_t index);
uint64_t __riscv_th_mmov_x_m (muint64_t src, size_t index);
```

4.8. Tuple instructions

Intrinsic functions list

```
// matrix tuple
mint8x2_t    __riscv_th_mset (mint8x2_t    src, size_t index, mint8_t    value);
```

```
mint16x2_t   __riscv_th_mset (mint16x2_t   src, size_t index, mint16_t   value);
mint32x2_t   __riscv_th_mset (mint32x2_t   src, size_t index, mint32_t   value);
mint64x2_t   __riscv_th_mset (mint64x2_t   src, size_t index, mint64_t   value);
muint8x2_t   __riscv_th_mset (muint8x2_t   src, size_t index, muint8_t   value);
muint16x2_t  __riscv_th_mset (muint16x2_t  src, size_t index, muint16_t  value);
muint32x2_t  __riscv_th_mset (muint32x2_t  src, size_t index, muint32_t  value);
muint64x2_t  __riscv_th_mset (muint64x2_t  src, size_t index, muint64_t  value);
mfloat16x2_t __riscv_th_mset (mfloat16x2_t src, size_t index, mfloat16_t value);
mfloat32x2_t __riscv_th_mset (mfloat32x2_t src, size_t index, mfloat32_t value);
mfloat64x2_t __riscv_th_mset (mfloat64x2_t src, size_t index, mfloat64_t value);

mint8_t    __riscv_th_mget (mint8x2_t    src, size_t index);
mint16_t   __riscv_th_mget (mint16x2_t   src, size_t index);
mint32_t   __riscv_th_mget (mint32x2_t   src, size_t index);
mint64_t   __riscv_th_mget (mint64x2_t   src, size_t index);
muint8_t   __riscv_th_mget (muint8x2_t   src, size_t index);
muint16_t  __riscv_th_mget (muint16x2_t  src, size_t index);
muint32_t  __riscv_th_mget (muint32x2_t  src, size_t index);
muint64_t  __riscv_th_mget (muint64x2_t  src, size_t index);
mfloat16_t __riscv_th_mget (mfloat16x2_t src, size_t index);
mfloat32_t __riscv_th_mget (mfloat32x2_t src, size_t index);
mfloat64_t __riscv_th_mget (mfloat64x2_t src, size_t index);
```

The INDEX argument must be provided as a constant integer expression.

4.9. Matrix Multiplication Instruction

The DEST represents the previous value of the return value, which requires initialization in the absence of an old value to prevent the appearance of unknown data. Furthermore, both the SRC1 and SRC2 serve as multipliers.

4.9.1. Floating point Matrix Multiplication

Fmmacc

Instructions

```
#matrix-matrix
fmmacc.h md, ms2, ms1
```

Intrinsic functions list

```
//matrix-matrix
mfloat16_t __riscv_th_fmmacc (mfloat16_t dest, mfloat16_t src1, mfloat16x2_t src2,
mrow_t row1, mrow_t row2, mcol_t col);
```

4.9.2. Integer 4x Extension Matrix Multiplication

Mmaqa

Instructions

```
#8bit data width
#signed matrix multiply
mmaqa.b md, ms2, ms1

#unsigned matrix multiply
mmaqau.b md, ms2, ms1

#unsigned-signed matrix multiply
mmaqaus.b md, ms2, ms1

#signed-unsigned matrix multiply
mmaqasu.b md, ms2, ms1
```

Intrinsic functions list

```
//signed matrix multiply
mint32_t __riscv_th_mmaqa (mint32_t dest, mint8_t src1, mint8_t src2, mrow_t row1,
mrow_t row2, mcol_t col);

//unsigned matrix multiply
mint32_t __riscv_th_mmaqau (mint32_t dest, muint8_t src1, muint8_t src2, mrow_t row1,
mrow_t row2, mcol_t col);

//unsigned-signed matrix multiply
mint32_t __riscv_th_mmaqaus (mint32_t dest, muint8_t src1, mint8_t src2, mrow_t row1,
mrow_t row2, mcol_t col);

//signed-unsigned matrix multiply
mint32_t __riscv_th_mmaqasu (mint32_t dest, mint8_t src1, muint8_t src2, mrow_t row1,
mrow_t row2, mcol_t col);
```

Pmmaqa

Instructions

```
#4bit data width
#signed matrix multiply
pmmaqa.b md, ms2, ms1

#unsigned matrix multiply
pmmaqau.b md, ms2, ms1
```

```
#unsigned-signed matrix multiply
pmmaqaus.b md, ms2, ms1

#signed-unsigned matrix multiply
pmmaqasu.b md, ms2, ms1
```

Intrinsic functions list

```
//signed matrix multiply
mint32_t __riscv_th_pmmaqa (mint32_t dest, mint8_t src1, mint8_t src2, mrow_t row1,
mrow_t row2, mcol_t col);

//unsigned matrix multiply
mint32_t __riscv_th_pmmaqau (mint32_t dest, muint8_t src1, muint8_t src2, mrow_t row1,
mrow_t row2, mcol_t col);

//unsigned-signed matrix multiply
mint32_t __riscv_th_pmmaqaus (mint32_t dest, muint8_t src1, mint8_t src2, mrow_t row1,
mrow_t row2, mcol_t col);

//signed-unsigned matrix multiply
mint32_t __riscv_th_pmmaqasu (mint32_t dest, mint8_t src1, muint8_t src2, mrow_t row1,
mrow_t row2, mcol_t col);
```

4.10. Mrelease

Instructions

```
mrelease
```

Intrinsic functions list

```
void __riscv_th_mrelease();
```

Chapter 5. Example

Source:

```
#include <stdio.h>
#include <thead_matrix.h>
#define N 16

void
print_data(const char *fmt, mfloat16_t src1, mfloat16x2_t src2, mfloat16_t dest,
mrow_t row1, mrow_t row2, mcol_t col)
{
  unsigned int i, j;
  float16_t tmp_src1[N];
  float16_t tmp_src2[N];
  float16_t tmp_dest[N];
  mfloat16_t src2_0 = __riscv_th_mget(src2, 0);

  long stride = col * sizeof(float16_t);

  __riscv_th_mst(tmp_src1, stride, src1, row1, col);
  __riscv_th_mst(tmp_src2, stride, src2_0, row1, col);
  __riscv_th_mst(tmp_dest, stride, dest, row1, col);

  printf("%s:\n", fmt);
  printf("src1:\t\tsrc2:\t\tdest:\n");
  for (i = 0; i < 2; i++)
  {
    for (j = 0; j < 2; j++)
    {
      printf("%-.2f ", tmp_src1[i * 2 + j]);
    }
    printf("\t");
    for (j = 0; j < 2; j++)
    {
      printf("%-.2f ", tmp_src2[i * 2 + j]);
    }
    printf("\t");
    for (j = 0; j < 2; j++)
    {
      printf("%-.2f ", tmp_dest[i * 2 + j]);
    }
    printf("\n");
  }
  printf("\n");
}

int main()
{
  /* init data */
```

```
  float16_t x[N] = {16.0, 15.0, 14.0, 13.0, 12.0, 11.0, 10.0, 9.0, 8.0, 7.0, 6.0, 5.0,
4.0, 3.0, 2.0, 1};
  float16_t y[N] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0,
13.0, 14.0, 15.0, 16};

  mrow_t row1 = 2;
  mrow_t row2 = 2;
  mcol_t col = 2;
  long stride = col * sizeof(float16_t);

  /* init matrix value*/
  mfloat16_t src1 = __riscv_th_mld(x, stride, row1, col);
  mfloat16_t src2_0 = __riscv_th_mld(y, stride, row1, col);
  mfloat16_t src2_1 = __riscv_th_mundefined_f16();

  mfloat16x2_t src2 = __riscv_th_mzero_f16x2();
  src2 = __riscv_th_mset(src2, 0, src2_0);
  src2 = __riscv_th_mset(src2, 1, src2_1);

  mfloat16_t dest = __riscv_th_mzero_f16();
  print_data("Initial value of matrix:", src1, src2, dest, row1, row2, col);

  dest = __riscv_th_fmmacc(dest, src1, src2, row1, row2, col);
  print_data("Results of multiplication:", src1, src2, dest, row1, row2, col);

  return 0;
}
```

Compile:

```
riscv64-unknown-linux-gnu-gcc -static -O2 -mcpu c907fdvm example.c -o example.exe
```

In order to enable the matrix intrinsics, we need to specify a CPU that supports the matrix extension, or add the option "xtheadmatrix" to the architecture option, such as "-march=rv64gc_xtheadmatrix". In this example, we use the "c907fdvm" core to demonstrate.

Result:

```
$ qemu-riscv64 -cpu c907fdvm ./example.exe
Initial value of matrix::
src1:           src2:           dest:
16.00 15.00     1.00 2.00       0.00 0.00
14.00 13.00     3.00 4.00       0.00 0.00

Results of multiplication::
src1:           src2:           dest:
16.00 15.00     1.00 2.00       46.00 108.00
```

```
14.00 13.00     3.00 4.00       40.00 94.00
```