RISC-V Matrix Multiplication Extension Specification

Alibaba

Version 0.4, 12/2023: This document is in development. Assume everything can change.

Table of Contents

Preamble

This document is in the Development state

Assume everything can change. This draft specification will change before being accepted as standard, so implementations made to this draft specification will likely not conform to the future standard.

Copyright and license information

Contributors

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

ZhaoJiang, WenmengZhang, WenganShao, ZhiweiLiu, XianmiaoQu, JingQiu, MingxinZhao, YunhaiShang, XiaoyanXiang, ChenChen

Others are welcome to help improve the specification.

Chapter 1. Introduction

This document is a matrix extension proposal for RISC-V.

The extension chooses a decoupled architecture from the vector extension for the flexibility and implementation considerations. Each harts supporting the matrix extension defines a RLEN constant parameters. RLEN is the number of bits in a row of a matrix register as described in Section Matrix Registers. The programmer’s model is designed to achieve binary portability on harts with different RLEN values.

The extension is strongly inspired by the RISC-V Vector extension.

Chapter 2. Programmer's Model

Matrix extension adds eight two-dimensional matrix registers and six CSRs.

Address	Privilege	Name	Description
xxxx	URW	xmrstart	matrix start row position
xxxx	URW	xmcsr	matrix control and status register
xxxx	URW	xmsize	matrix size configure, matrix size
xxxx	URO	xmlenb	matrix register size in byte, mrows*xrlenb
xxxx	URO	xrlenb	RLEN (in bytes)
xxxx	URO	xmisa	matrix instruction subset

2.1. Matrix Registers

If matrix extension is implemented, there are eight two-dimensional matrix registers named M0, M1, M2, M3, M4, M5, M6 and M7 added to architectural state.

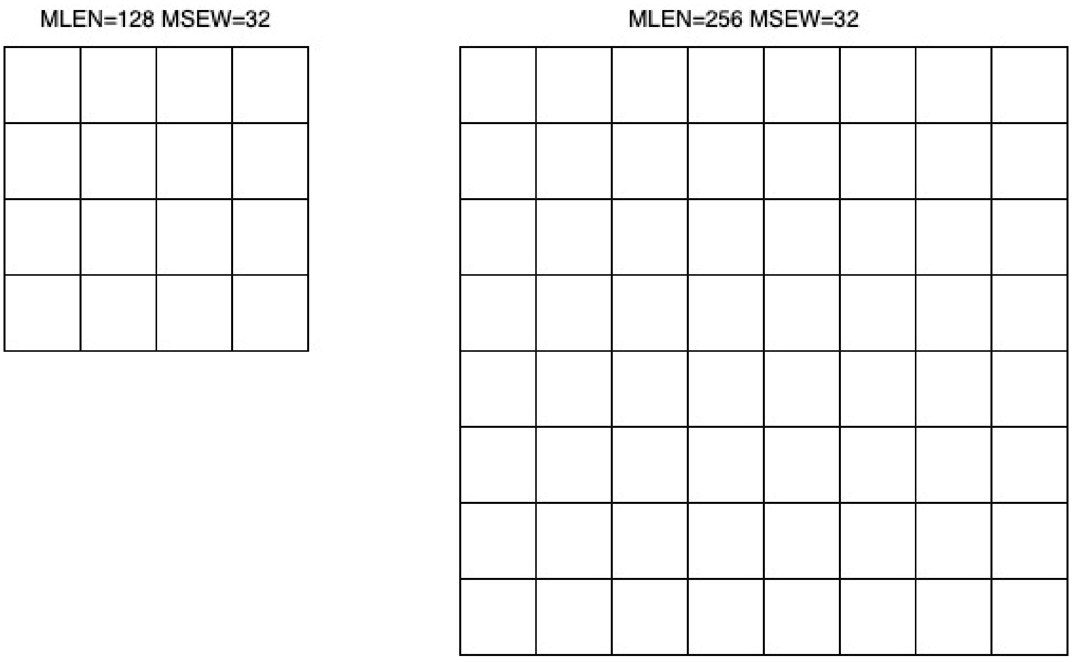
RLEN represents the length of each register row in bits. It is a constant value in any implementation, and the available RLEN options are 128, 256, or 512 or more. The matrix register row number is RLEN divided by 32, resulting in 4, 8, or 16 or more. Consequently, each matrix register comprises [M x K] MSEW elements, where M is calculated as RLEN divided by 32, K is calculated as RLEN divided by MSEW, and MSEW represents the element size.

The example size of the matrix registers varies as following.

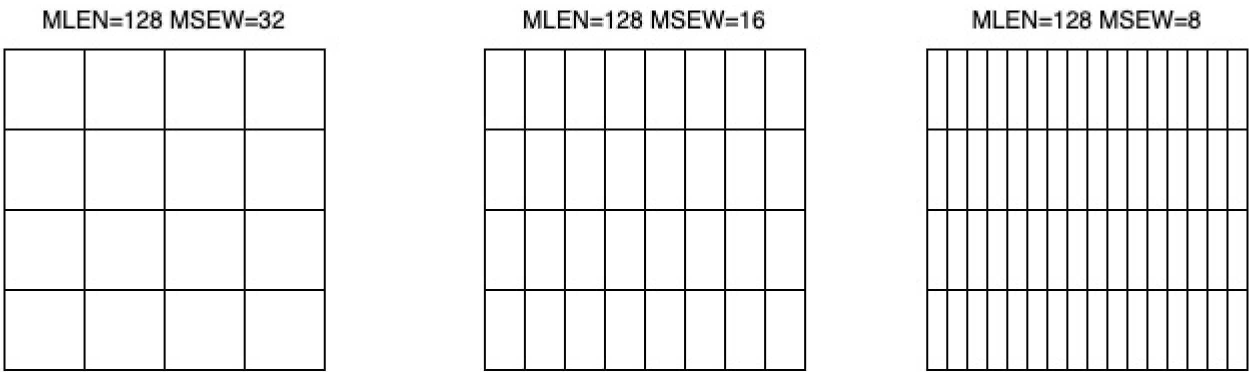
size[2:0]	operand datawidth	RLEN(bit)	M	K	Matrix size in bits
100	4bit	128	4	32	512
		256	8	64	2048
		512	16	128	8192
000	8bit	128	4	16	512
		256	8	32	2048
		512	16	64	8192
001	16bit	128	4	8	512
		256	8	16	2048
		512	16	32	8192
010	32bit	128	4	4	512
		256	8	8	2048
		512	16	16	8192

size[2:0]	operand datawidth	RLEN(bit)	M	K	Matrix size in bits
011	64bit	128	4	2	512
		256	8	4	2048
		512	16	8	8192

If MSEW remains constant, when RLEN doubles, the rows and the columns both double, resulting in four times as many bits.



If RLEN remains constant, when MSEW doubles, the rows remain the same while the columns halve.



2.2. Matrix Size Configure

Matrix size configure is a XLEN-bit WARL read-write register. It also can be updated by matrix configure instructions. The matrix size register has three fields, sizeK, sizeN and sizeM. Bits[XLEN-1:32] are reserved.

bits	Name	Description
XLEN-1:XLEN-32	0	reserved if non-zero
31:16	sizeK[15:0]	column of Matrix A or Matrix B, in bytes
15:8	sizeN[7:0]	row of Matrix B
7:0	sizeM[7:0]	row of Matrix A

The sizeM/sizeN/sizeK field hold an unsigned integer specifying the source elements needed and the destination elements updated by a matrix instructions. The sizeK which is not the multiples of element size in byte will raise an illegal instruction exception.

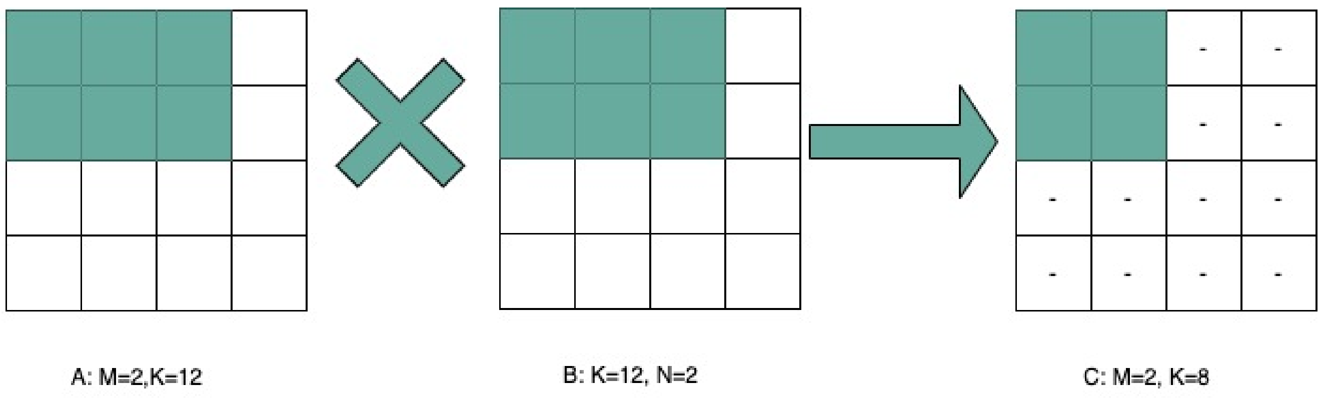
For matrix-multiplication instructions, which computing C[M][N] += A[M][K]*B^T[N][K], there are 3 source operands and 1 destination operand. Only sizeM x sizeN elements will be updated, the other elements are set by zeros. The source operands dimensions are defined as follows:

- Matrix A: sizeM x (sizeK/element size)
- Matrix B: sizeN x (sizeK/element size)
- Matrix C: sizeM x sizeN

Thus, there are the limitations of Matrix shape due to the matrix register.

- sizeK ⇐ xrlenb
- sizeM ⇐ RLEN/32
- sizeN ⇐ RLEN/32, for fmmacc.h sizeN ⇐ 2*(RLEN/32)

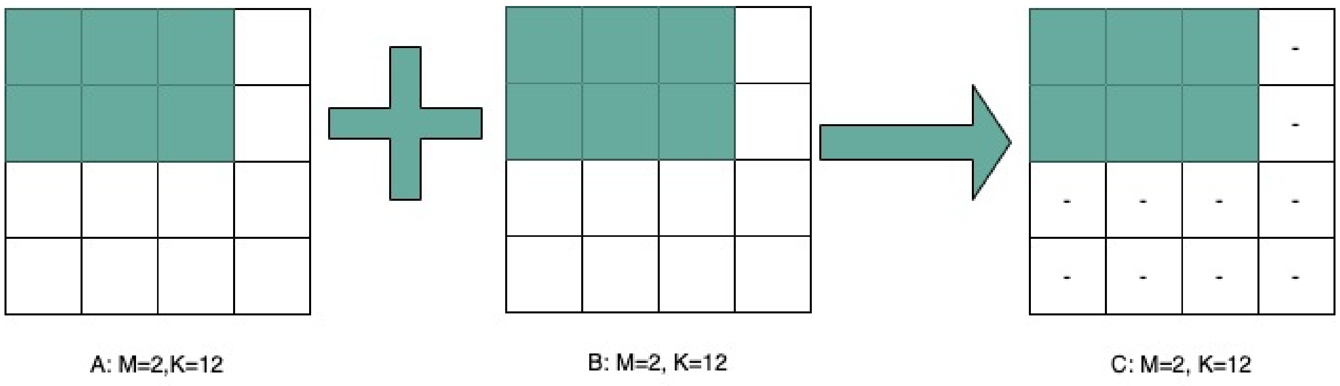
Taking 32-bit matrix-multiplication with RLEN=128 as an example, the configuration of sizeM=2 / sizeK=12 / sizeN=2 indicates MatrixA(2x3) x MatrixB^T(2x3)+MatrixC(2x2), only the green block elements are used or updated by the instruction.



For pointwise and load/store instructions, the matrix shapes keep during the execution, which are specified by sizeM and sizeK. Only sizeM x sizeN elements will be updated, the other elements are set by zeros. The size limitations are:

- sizeM ⇐ RLEN/32
- sizeK ⇐ max_colb

Int32 matrix add as example , the configuration of sizeM=2/sizeK=12 indicates MatrixA(2x3) x+MatrixB(2x3)=MatrixC(2x3), only the green block elements are used or updated by the instruction.



2.3. Matrix Control and Status

The xmcsr CSR is a WARL read-write register. Bits[XLEN-1:3] are reserved and should be written with zero. The layout of matrix control and status register is:

bits	name	description
XLEN-1:3	0	reserved if non-zero
2	xmsat	Fixed-point accrued saturation flag
1:0	xmxrm	Fixed-point rounding mode

2.3.1. Matrix fixed-point rounding mode

Matrix fixed-point rounding mode(xmxrm) filed is defined in bit[3:2] of matrix control and status register. The xmxrm uses the same encoding and rounding algorithm with vxrm[1:0] as follows. Suppose the pre-rounding result is v, and d bits of that result are to be rounded off. Then the rounded result is (v >> d) + r, where r depends on the rounding mode as specified in the following table.

vxrm[1:0]		rounding mode	rounding increment r
0	0	rnu round-to-nearest-up (add +0.5 LSB)	v[d-1]
0	1	rne round-to-nearest-even	v[d-1] & (v[d-2:0]≠0 | v[d])
1	0	rdn round-down (truncate)	0
1	1	rod round-to-odd (OR bits into LSB, aka "jam")	!v[d] & v[d-1:0]≠0

The rounding functions are used to represent this operation in the instruction descriptions below:

```
roundoff_unsigned(v, d) = (unsigned(v) >> d) + r
roundoff_signed(v, d) = (signed(v) >> d) + r
```

2.3.2. Matrix fixed-point saturation flag

The xmxsat filed indicates if a fixed-point instruction has had to saturate an output value to fit into a destination format.

2.4. Matrix Register Information

Matrix register information includes two read-only XLEN-bit registers, which are constant in any implementation.

- xrlenb: RLEN in byte indicating RLEN-bits state of each matrix register row
- xmlenb: matrix register size in byte, mrows*xrlenb, mrows=RLEN/32

2.5. Matrix Start Row

The xmrstart read-write register indicates the first matrix row index to be executed by a matrix load/store instruction. Normally xmrstart is only written by hardware on a trap of matrix load/store instructions, the unsigned value of register specifies the row at which the execution should resume after a resumable trap is handled.

All matrix instructions, including mcfg/mcfgi, reset the xmrstart CSR to zero.

The xmrstart CSR is defined to have only enough writable bits to hold the largest row index(one less than the max row) or log2(RLEN/32). The upper bits of the xmrstart CSR are hardwired to zero(reads zero, writes ignored)

For example, xmrstart would have 2 bits to represent row indices from 0 through 3

2.6. Matrix ISA

Xmisa is an XLEN-bit read-only CSR register, specifying the supported matrix instruction subset of the current hardware implementation.

bits	FEATURE	
31	MATRIX_PW_FLOAT	optional
29	MATRIX_PW_FLOAT	optional
28	MATRIX_PW_INT	optional
8	MATRIX_MULT_F16F32	optional
7	MATRIX_MULT_F32F64	optional
6	MATRIX_MULT_F16F32	optional
5	MATRIX_MULT_F64F64	optional
4	MATRIX_MULT_F32F32	optional
3	MATRIX_MULT_F16F16	optional

bits	FEATURE	
2	MATRIX_MULT_I16I64	optional
1	MATRIX_MULT_I8I32	compulsory
0	MATRIX_MULT_I4I32	optional

bit[i] =1 indicates the optional feature is supported.

- MATRIX_MULT_I4I32: for matrix-multiplication instruction, element in source registers is int4 and in destination registers is int 32;
- MATRIX_MULT_I8I32: for matrix-multiplication instruction, element in source registers is int8 and in destination registers is int 32;
- MATRIX_MULT_I16I64: for matrix-multiplication instruction, element in source registers is int16 and in destination registers is int 64;
- MATRIX_MULT_F16F16: for matrix-multiplication instruction, element in source and destination registers are fp16/bf16;
- MATRIX_MULT_F32F32: for matrix-multiplication instruction, element in source and destination registers are fp32;
- MATRIX_MULT_F64F64: for matrix-multiplication instruction, element in source and destination registers are fp64;
- MATRIX_MULT_F16F32: for widen matrix-multiplication instruction, element in source registers is fp16/bf16, element in destination registers is fp16/bf16;
- MATRIX_MULT_F32F64: for widen matrix-multiplication instruction, element in source registers is fp32 element in destination registers is fp64;
- MATRIX_PW_INT: for integer pointwise instruction, cannot be set alone, only works if it is enabled with the corresponding integer multiply instruction (refer to section 2.4 for details);
- MATRIX_PW_FP: for float pointwise instruction, cannot be set alone, only works if it is enabled with the corresponding multiply instruction (refer to section 2.4 for details).
- MATRIX_FLOAT_INT_CVT: for float integer conversion instruction, cannot be set alone, only works if it is enabled with the corresponding multiply instruction (refer to section 2.4 for details).

2.7. State of Matrix Extension at Reset

The matrix extension must have a consistent state at reset. It is recommended that at reset, CSRs are set to zero.

2.8. Matrix Context Status

A matrix context status field, MS, is defined to mstatus and shadowed in sstatus, which can be used to reduce the cost of context save and restore. The MS fields uses the same status encoding as FS/VS/XS, shown in the table.

status	ms[1:0]	MS Meaning
0	2’b00	All off
1	2’b01	Initial
2	2’b10	Clean
3	2’b11	Dirty

Attempts to execute any matrix instructions, or to access the matrix CSRs raise an illegal instruction exception when MS is set to off. If MS is set to initial or clean, executing any instructions that change the matrix state will change the ms to dirty.

An implementation can use the activity of the Initial state to influence the choice of power-saving states.

Chapter 3. Instructions

3.1. Matrix Multiplication Instructions

Matrix multiplication instructions take matrixA(sizeMxsizeK) and matrixB(sizeNxsizeK) from matrix registers specified by ms1 and ms2, and accumulate the multiplication result of A[M][K] * (B[N][K])T to matrixC(sizeM x sizeN) from md register, the output will overwrite the accumulation register.

- shape of matrixA: M rows(sizeM), K columns(sizeK/element size in byte)
- shape of matrixB: N rows(sizeN), K columns(sizeK/element size in byte)
- shape of matrixC: M rows(sizeM), N columns(sizeN)

The function description:

```
for(int i=0; i<sizeM; i++) {
  for(int j=0; j<sizeN; j++) {
      for(int k=0; k<(sizeK/element size); k++)
         C[i,j] += A[i,k]*B[j,k];
}}}
```

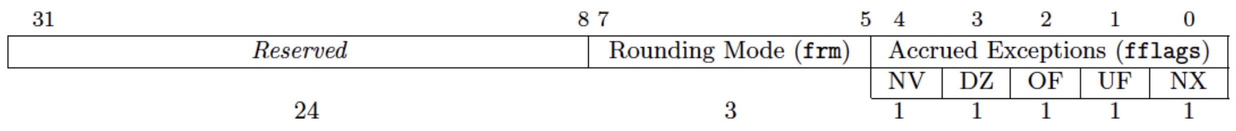
The ISA specification provides different instructions to support float and integer matrix multiplication and and operation. Hardware design has the flexibility of supported data types.

category	instructions	Operand Type A,B	Accumulator Type C	Optional Feature
Float	fmmacc.h	fp16/bf16	fp16	MATRIX_MULT_F16F16
	fmmacc.s	fp32	fp32	MATRIX_MULT_F32F32
	fmmacc.d	fp64	fp64	MATRIX_MULT_F64F64
	fwmmacc.h	fp16/bf16	fp32	MATRIX_MULT_F16F32
	fwmmacc.s	fp32	fp64	MATRIX_MULT_F32F64

category	instructions	Operand Type A,B	Accumulator Type C	Optional Feature
Int	mmaqa.b mmaqu.b mmaqasu.b mmaqaus.b	int8	int32	MATRIX_MULT_I8I32
	mmaqa.h mmaqu.h mmaqasu.h mmaqaus.h	int16	int64	MATRIX_MULT_I16I64
	pmmaqa.b pmmaqu.b pmmaqasu.b pmmaqaus.b	int4(mx8)	int32(mxm)	MATRIX_MULT_I4I32

The hardware implementation can choose one or more subsets .

The float matrix multiplication reuses the floating-point control and status register, fcsr, to select the dynamic rounding mode for floating-point arithmetic operations and hold the accrued exception flags.



The float matrix multiplication uses the dynamic rounding mode in frm. If frm is set to an invalid value (101-111), any subsequent attempt to execute a floating-point operation with a dynamic rounding mode will cause an illegal instruction exception.

rounding mode	Mnemonic	Meaning
000	RNE	Round to Nearest, ties to Even
001	RTZ	Round towards Zero
010	RDN	Round Down (towards -∞)
011	RUP	Round Up (towards +∞)
100	RMM	Round to Nearest, ties to Max Magnitude
101		Invalid. Reserved for future use
110		Invalid. Reserved for future use
111		Invalid in rounding mode register

If the floating-point unit status field mstatus.FS is off then any attempt to execute a matrix floating-point instruction will raise an illegal instruction exception. Any matrix floating-point instruction that modifies any floating-point extension state (i.e., floating-point CSRs or f registers) must set mstatus.FS to Dirty. The basic operation of float matrix multiplication is float dot , the float dot operations follow the IEEE-754/2008 standard.

For float dot, if any operand element is NaN or a product of ∞ x 0 or a sum of infinities of different signs, the result is NaN. Except when otherwise stated, if the result is NaN, it is the canonical NaN. A product of ∞ x 0 or a sum of infinities of different signs signals the invalid operation exception. Otherwise, sums are computed with no avoidable intermediate exception conditions in the calculation and the final result is determined from that intermediate result. If the final result overflows, signal overflows. If the final result underflows, signal underflows. If the final result is inexact, signal is inexact.

The standard matrix floating-point instructions treat elements as IEEE-754/2008-compatible values. If the EEW of a matrix floating-point operand does not correspond to a supported IEEE floating-point type, the instruction encoding is reserved. For bf16-extension, 16-bit floating-point element can be seen as bf16 or fp16.

3.1.1. Float Matrix Multiplication(non-widen)

Non-widen float matrix multiplication indicates the source and destination operands data width keep the same which are encoded in the instruction.

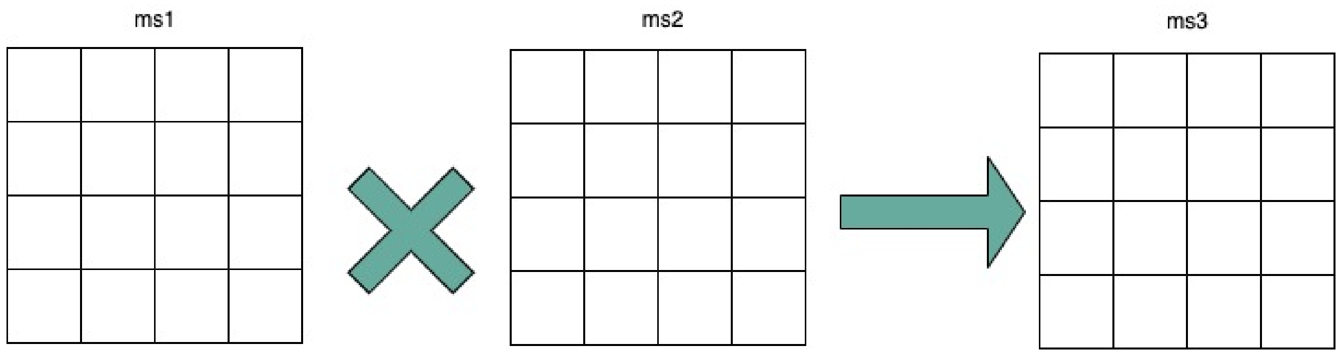
- fmmacc.h: fp16/bf16 floating-point ,illegal if bit[3] of xmisa register is 0
- fmmacc.s: fp32 floating-point, illegal if bit[4] of xmisa register is 0
- fmmacc.d: fp64 floating-point, illegal if bit[5] of xmisa register is 0

```
#float matrix multiplication, md = md + ms1*ms2
fmmacc.h md, ms2, ms1
fmmacc.s md, ms2, ms1
fmmacc.d md, ms2, ms1
```

For fmmacc.s, the max matrix shape is:

- matrixA: M ⇐ RLEN/32, K ⇐ RLEN/32
- matrixB: N ⇐ RLEN/32, K ⇐ RLEN/32
- matrixC: M ⇐ RLEN/32, N ⇐ RLEN/32

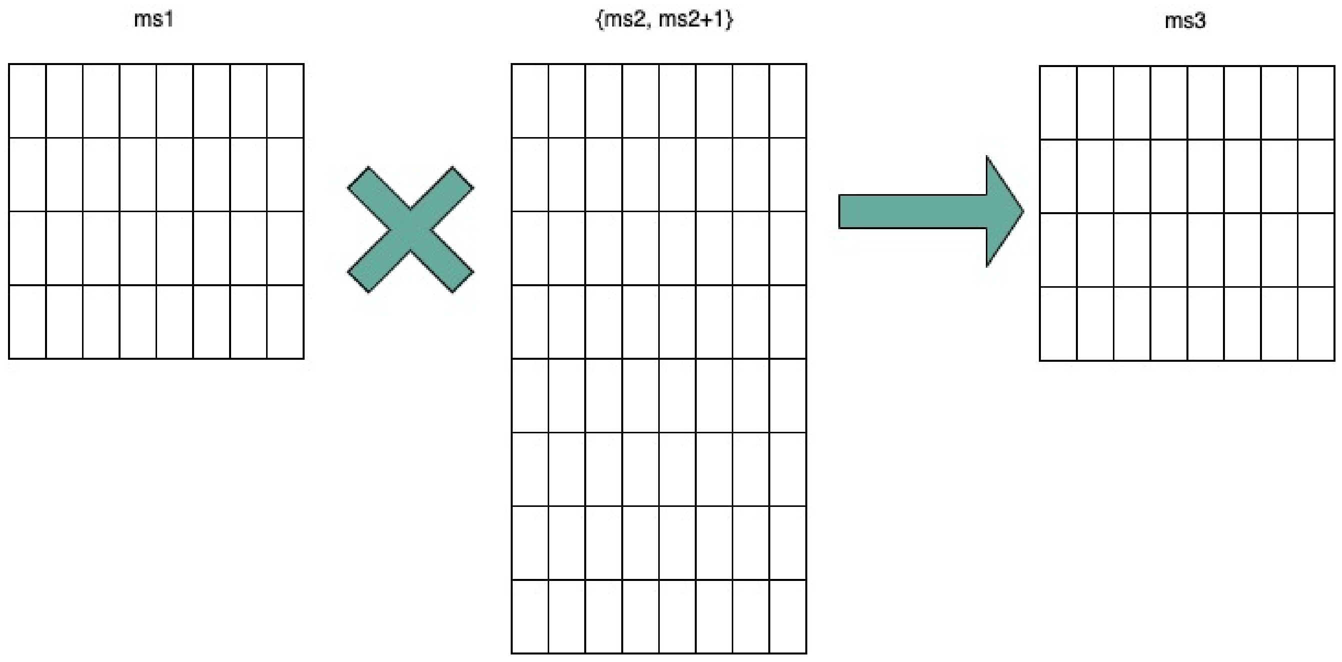
The operation of fmmacc.s is shown below for RLEN=128.



For fmmacc.h, 16-bit float matrix multiplication and add instruction, the element can be seen as fp16 or bf16 if bf16 data type is supported. The max matrix shape is:

- matrixA: M ⇐ RLEN/32, K ⇐ RLEN/16
- matrixB: N ⇐ RLEN/16, K ⇐ RLEN/16
- matrixC: M ⇐ RLEN/32, N ⇐ RLEN/16

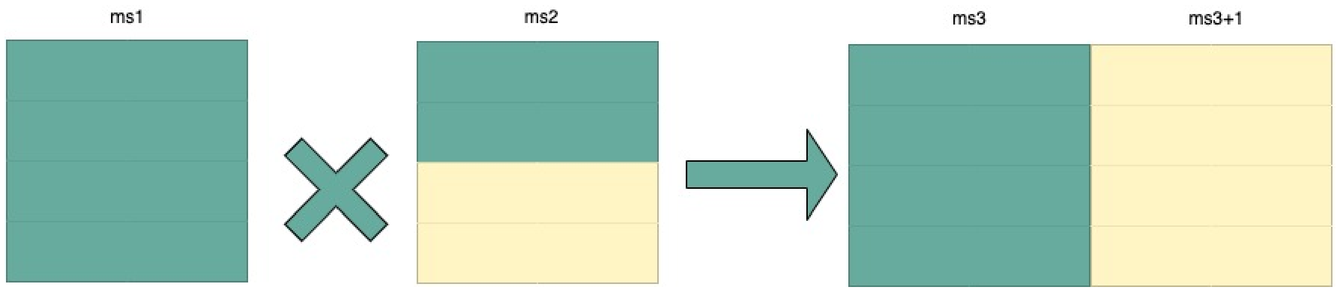
As data width for matrix B is twice that of matrix A and C, two matrix register(register-pair) are used by Matrix B specified by ms_2 and ms_2+1. Instructions specifying an odd-numbered ms_2 is reserved. The operation is shown below for RLEN=128.



For fmmacc.d, 64-bit float matrix multiplication and add instruction, The maximum matrix shape is:

- matrixA: M ⇐ RLEN/32, K ⇐ RLEN/64
- matrixB: N ⇐ RLEN/32, K ⇐ RLEN/64
- matrixC: M ⇐ RLEN/32, N ⇐ RLEN/32

As data width for matrix C is twice that of matrix A and B, two matrix register(register-pair) are used by MatrixC specified by md and md+1. Instructions specifying an odd-numbered md is reserved. the operation is shown below for RLEN=128.



Summary for max Matrix size of fmmacc instructions for typical RLEN:

		matrix A			**matrix B**			**matrix C**		
	RLEN	M	K	data width	N	K	data width	M	N	data width
fmacc.s	128	4	4	512 bits	4	4	512 bits	4	4	512 bits
	256	8	8	2048 bits	8	8	2048 bits	8	8	2048 bits
	512	16	16	8192 bits	16	16	8192 bits	16	16	8192 bits
fmacc.h	128	4	8	512 bits	8	8	1024 bits	4	8	512 bits
	256	8	16	2048 bits	16	16	4096 bits	8	16	2048 bits
	512	16	32	8192 bits	32	32	16384 bits	16	32	8192 bits
fmacc.d	128	4	2	512 bits	4	2	512 bits	4	4	1024 bits
	256	8	4	2048 bits	8	4	2048 bits	8	8	4096 bits
	512	16	8	8192 bits	16	8	8192 bits	16	16	16384 bits

3.1.2. Float Matrix Multiplication(widen)

Widen float matrix multiplication indicates destination operand data width is twice of the source operand. The data width of source operand is in instruction encoding.

- fwmmacc.h: fp16/bf16 floating-point source and fp32 result ,illegal if bit[8] of xmisa register is 0
- fwmmacc.s: fp32 floating-point source and fp64 result , illegal if bit[9] of xmisa register is 0

```
#float matrix multiplication, output widen, md = md + ms1*ms2
fwmmacc.h md, ms2, ms1
fwmmacc.s md, ms2, ms1
```

For fwmmacc.h, 16-bit float widen matrix multiplication and add instruction, the element can be seen as fp16 or bf16 if bf16 data type is supported. The maximum matrix shape is:

- matrixA: M ⇐ RLEN/32, K ⇐ RLEN/16
- matrixB: N ⇐ RLEN/32, K ⇐ RLEN/16
- matrixC: M ⇐ RLEN/32, N ⇐ RLEN/32

For fwmmacc.s, 32-bit float widen matrix multiplication and add instruction, The maximum matrix shape is:

- matrixA: M ⇐ RLEN/32, K ⇐ RLEN/32
- matrixB: N ⇐ RLEN/32, K ⇐ RLEN/32
- matrixC: M ⇐ RLEN/32, N ⇐ RLEN/32

As data width for matrix C is twice that of matrix A and B, two matrix register(register-pair) are used by MatrixC specified by md and md+1. Instructions specifying an odd-numbered md is reserved. Summary for max Matrix size of fwmmacc instructions for typical RLEN:

		matrix A			**matrix B**			**matrix C**		
	RLEN	M	K	data width	N	K	data width	M	N	data width
fwmacc.h	128	4	8	512 bits	4	8	512 bits	4	4	512 bits
	256	8	16	2048 bits	8	16	2048 bits	8	8	2048 bits
	512	16	32	8192 bits	16	32	8192 bits	16	16	8192 bits
fwmacc.s	128	4	4	512 bits	4	4	512 bits	4	4	1024 bits
	256	8	8	2048 bits	8	8	2048 bits	8	8	4096 bits
	512	16	16	8192 bits	16	16	8192 bits	16	16	16384 bits

3.1.3. Integer Matrix Multiplication (4x widen)

The integer matrix multiplication with destination data width is four-times that of the source data width. The source operand data width in instruction encoding supported are int8 and int16, other data widths are reserved. Both signed/unsigned versions are provided . Thus, the source operand can be both signed/both unsigned/signed-unsigned/unsigned-signed, the result of multiplication is sign-extended before addition and accumulation. Overflow is ignored and the result wraps around.

- mmaqa.b/mmaqau.b/mmaqaus.b/mmaqasu.b: int8 four-times widen matrix multiplication, illegal if bit[1] of xmisa register is 0
- mmaqa.h/mmaqau.h/mmaqaus.h/mmaqasu.h: int16 four-times widen matrix multiplication, illegal if bit[2] of xmisa register is 0

```
#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

#16bit data width
#signed matrix multiply
mmaqa.h md, ms2, ms1
```

```
#unsigned matrix multiply
mmaqau.h md, ms2, ms1
#unsigned-signed matrix multiply
mmaqaus.h md, ms2, ms1
#signed-unsigned matrix multiply
mmaqasu.h md, ms2, ms1
```

For int8 four-times matrix-multiplication, the maximum matrix shape is:

- matrixA: M ⇐ RLEN/32, K ⇐ RLEN/8
- matrixB: N ⇐ RLEN/32, K ⇐ RLEN/8
- matrixC: M ⇐ RLEN/32, N ⇐ RLEN/32

For int16 four-times matrix-multiplication, as data width for matrix C is four-times of matrix A and B, two matrix register(register-pair) are used by matrix C specified by md and md+1. Instructions specifying an odd-numbered md is reserved. the maximum matrix shape is:

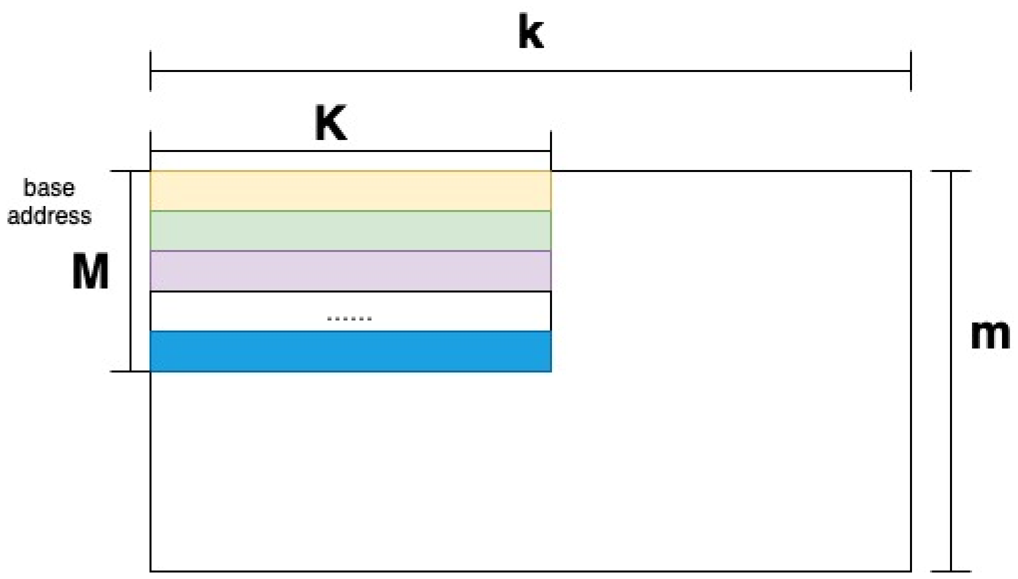
- matrixA: M ⇐ RLEN/32, K ⇐ RLEN/16
- matrixB: N ⇐ RLEN/32, K ⇐ RLEN/16
- matrixC: M ⇐ RLEN/32, N ⇐ RLEN/32

Summary for max Matrix size of integer matrix multiply and add instructions for typical RLEN:

	matrix A				**matrix B**			**C**		
	RLEN	M	K	data width	N	K	data width	M	N	data width
int8 4x	128	4	16	512 bits	4	16	512 bits	4	4	512 bits
	256	8	32	2048 bits	8	32	2048 bits	8	8	2048 bits
	512	16	64	8192 bits	16	64	8192 bits	16	16	8192 bits
int16 4x	128	4	8	512 bits	4	8	512 bits	4	4	1024 bits
	256	8	16	2048 bits	8	16	2048 bits	8	8	4096 bits
	512	16	32	8192 bits	16	32	8192 bits	16	16	16384 bits

3.2. Matrix Load/Store Instructions

Matrix load instructions load a matrix from memory to matrix register. and matrix store instructions store a matrix from matrix register to memory.



The element data width is in instruction encoding, including byte/halfword/word/doubleword, other data widths are reserved. The base address is in rs1 and row stride in byte is in rs2, md/ms3 is the register index for destination of matrix load and source for matrix store.

```
#matrix load
mld.<b/h/w/d> md, (rs1), rs2
#stream matrix load
mld.<b/h/w/d>.s  md, (rs1), rs2
#matrix store
mst.<b/h/w/d>  ms3, (rs1), rs2
#stream matrix store
mst.<b/h/w/d>.s  ms3, (rs1), rs2
#whole matrix load
mld<1/2/4/8>m.<b/h/w/d> md, (rs1)
#whole matrix store
mst<1/2/4/8>m.<b/h/w/d> ms3, (rs1)
```

Matrix shape (MxK) is in matrix size configure register, M given by sizeM and K given by sizeK(in byte). M=sizeM ⇐ RLEN/32, K=sizeK/element size in byte, sizeK ⇐ RLEN/8. If sizeM < RLEN/32 or sizeK < RLEN/8, the matrix register data with row index > sizeM or column index > (sizeK/ element size in byte) set zero for load, and don’t write to memory for store.

There are 2 versions provided: (1)normal (2) whole register load/store.

Whole register load/store data with maximum matrix size from/to memory with sizeM = RLEN/32 and sizeK = RLEN/8. The matrix size configurations are ignored.

These instructions are intended to be used to save and restore matrix registers when the length of the current contents of the matrix register is not known, or where modifying matrix size would be costly. Examples include compiler register spills, function calls where values are passed in matrix registers, interrupt handlers, and OS context switches. Software can determine the number of bytes transferred by reading the xmlenb register.

---

rs2 field is reused to specify the register number. rs2[4:3] is set to 0, otherwise reserved. rs2[2:0] is nf field, encoding how many matrix registers to load and store using the NFIELDS encoding. md/ms2 register index should be aligned with the register number.

nf[2:0]	register number
000	1
001	2
011	4
111	8
others	reserved

All matrix load/store instructions may generate and accept a non-zero row-start value. The row-start register is reset to zero at the end of the matrix instruction execution.

With the ZIHINTNTL extension, matrix memory access instruction can behave as stream memory access operations to fit different memory hierarchy. Stream memory access instructions have the same function as normal matrix load/store instructions, except that the data may not be reused in the near future which can be potentially optimized by hardware implementation.

3.3. Configuration Instructions

Matrix configuration instructions configure a field or the whole matrix size configuration register. The field retains the value if not changed by a configuration instruction. The index field of the instruction indicates which field is updated, sizeM/sizeK/sizeN or the entire configure register as following table shows. The new matrix size are returned to rd.

index	instruction	effect on matrix size
000	mcfgk(i)	msize.half1 = x[rs1]
001	mcfgm(i)	msize.byte0 = x[rs1]
010	mcfgn(i)	msize.byte1 = x[rs1]
111	mcfg	msize.byte0 = x[rs1] .byte0 msize.byte1 = x[rs1] .byte1 msize.half1 = x[rs1] .half1
others	reserved	

```
#imm type
mcfg<m/n/k>i  uimm7
#register type
mcfg<m/n/k>    rs1
#entire register
mcfg rs1
```

3.4. Pointwise Instructions

3.4.1. Integer Pointwise Arithmetic Instructions

For integer matrix pointwise instructions, matrix-matrix/matrix-vector instruction format are provided. 32-bit and 64-bit integer instructions are optionally supported.

- 32bit instructions: legal if bit[28] and bit[1] of xmisa register are both set to 1, also legal if bit[28] and bit[0] of xmisa register are both set to 1, illegal In all other cases.
- 64bit instructions: legal if bit[15] and bit[2] of xmisa register are both set to 1, illegal In all other cases.

The matrix operands shape is M/K, provided by sizeM x (sizeK/element size in byte).

- sizeM ⇐ RLEN/32
- sizeK ⇐ RLEN/8

operand datawidth	RLEN (bit)	M	K
32bit	128	4	4
	256	8	8
	512	16	16
64bit	128	4	2
	256	8	4
	512	16	8

For matrix-matrix instructions, both sources are matrix. For matrix-vector instructions, src2 is matrix and src1 is one row of matrix. Use row0 as an example, the vector operand operates on each row of matrix operand as md[i, j] = ms2[i, j] op ms1[0, j].

madd performs the addition of src1 and src2. msub performs the subtraction of src2 from src1. mmul performs the multiplication of src1 and src2. Overflows are ignored and the low XLEN bits of results are written to the destination rd. The mmul versions write the low bits of the product to the destination register, while the mmulh versions write the high bits of the product to the destination register. mmax and mmin perform signed and unsigned compares respectively.

```
#matrix-matrix add
madd.<s/d>.mm md, ms2, ms1
#matrix-vector add
madd.<s/d>.mv.i md, ms2, ms1[uimm3]

#matrix-matrix sub
msub.<s/d>.mm md, ms2, ms1
#matrix-vector sub
msub.<s/d>.mv.i md, ms2, ms1[uimm3]

#matrix-matrix mul
```

```
mmul.<s/d>.mm md, ms2, ms1
#matrix-vector mul
mmul.<s/d>.mv.i md, ms2, ms1[uimm3]

#matrix-matrix mulh
mmulh.<s/d>.mm md, ms2, ms1
#matrix-vector mulh
mmulh.<s/d>.mv.i md, ms2, ms1[uimm3]

#matrix-matrix max
mmax.<s/d>.mm md, ms2, ms1
#matrix-vector max
mmax.<s/d>.mv.i md, ms2, ms1[uimm3]

#matrix-matrix unsigned max
mumax.<s/d>.mm md, ms2, ms1
#matrix-vector unsigned max
mumax.<s/d>.mv.i md, ms2, ms1[uimm3]

#matrix-matrix min
mmin.<s/d>.mm md, ms2, ms1
#matrix-vector min
mmin.<s/d>.mv.i md, ms2, ms1[uimm3]

#matrix-matrix unsigned min
mumin.<s/d>.mm md, ms2, ms1
#matrix-vector unsigned min
mumin.<s/d>.mv.i md, ms2, ms1[uimm3]
```

Matrix shift instructions including mn4clip/msll/msrl/msra.

mn4clip/mn4clipu instructions are used to pack a fixed-point value into a 4x narrower destination. Rounding, scaling and saturation are supported. The scaling shift amount comes from a matrix (specified by ms1), a vector(ms1[uimm3]) . The low 6-bits for 64-bit and 5-bits for 32-bit source data width are used, the higher bits are ignored. Saturation sets xmsat if the destination overflows.

msll msrl and msra perform logical left logic right and arithmetic right shift, the source data is in ms2, and the shift amount is provided by a matrix/vector data specified by ms1/ms1[uimm3].

Matrix shift instructions support rounding with rounding mode specified in the xmxrm CSR. For clip instructions, rounding occurs before saturation.

```
#matrix-matrix logical left shift
msll.<s/d>.mm md, ms2, ms1
#matrix-vector logical left shift
msll.<s/d>.mv.i md, ms2, ms1[uimm3]

#matrix-matrix logic right shift
msrl.<s/d>.mm md, ms2, ms1
#matrix-vector logic right shift
```

```
msrl.<s/d>.mv.i md, ms2, ms1[uimm3]


#matrix-matrix arithmetic right shift
msra.<s/d>.mm md, ms2, ms1
#matrix-vector arithmetic right shift
msra.<s/d>.mv.i md, ms2, ms1[uimm3]

#matrix-matrix signed clip
mn4clip.<s/d>.mm md, ms2, ms1
#matrix-vector clip,uimm3
mn4clip.<s/d>.mv.i md, ms2, ms1[uimm3]

#matrix-matrix unsigned clip
mn4clipu.<s/d>.mm md, ms2, ms1
#matrix-vector clip,uimm3
mn4clipu.<s/d>.mv.i md, ms2, ms1[uimm3]
```

3.5. Float Pointwise Arithmetic Instructions

16bit 32-bit and 64-bit floating point pointwise operations are optionally supported.

- for fp16/bf16 pointwise arithmetic instructions, legal if bit[31] and bit[3] of xmisa register are both set to 1, illegal In all other cases.
- for fp32 pointwise arithmetic instructions, legal if bit[31] and bit[6] of xmisa register are both set to 1, also legal if bit[31] and bit[4] of xmisa register are both set to 1, illegal In all other cases.
- for fp64 pointwise arithmetic instructions, legal if bit[31] and bit[7] of xmisa register are both set to 1, also legal if bit[31] and bit[5] of xmisa register are both set to 1, illegal In all other cases.
- for conversion between fp64 and fp32 , legal if bit[31] and bit[7] of xmisa register are both set to 1, illegal In all other cases.
- for conversion between fp32 and fp16/bf16 , legal if bit[31] and bit[6] of xmisa register are both set to 1, illegal In all other cases

The matrix operands shape is M/K, provided by sizeM x (sizeK/element size in byte).

operand datawidth	**RLEN (bit)**	**M**	**K**
16bit	128	4	8
	256	8	16
	512	16	32
32bit	128	4	4
	256	8	8
	512	16	16

operand datawidth	RLEN (bit)	M	K
64 bit	128	4	2
	256	8	4
	512	16	8

For matrix-matrix instructions, both sources are matrixs. For matrix-vector instructions, src2 is matrix and src1 is one row of matrix. Use row0 as an example, the vector operand operates on each row of matrix operand as md[i, j] = ms2[i, j] operations with ms1[0, j].

All floating-point pointwise instructions that perform rounding can select the rounding mode using the frm field.

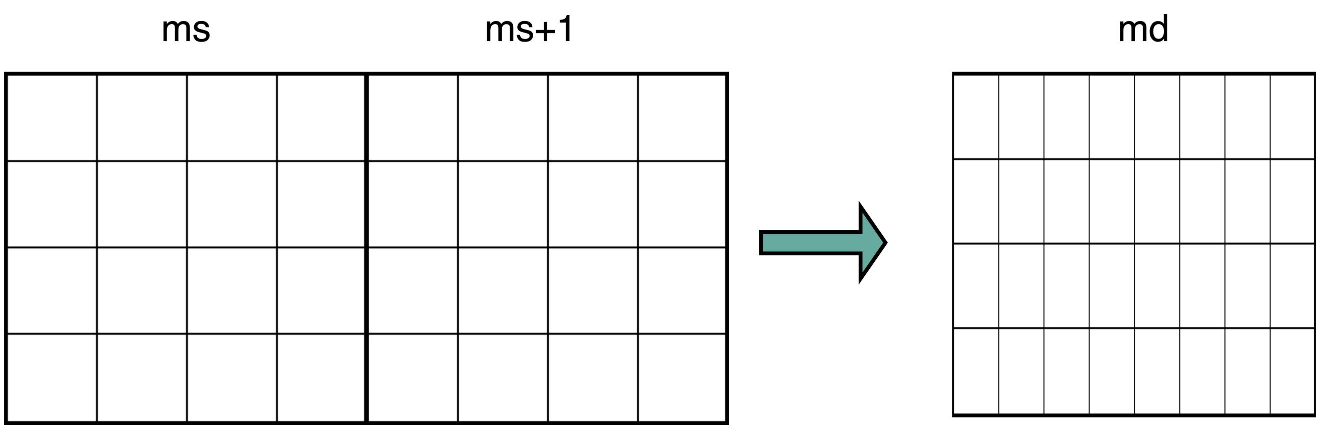
Non-widen float pointwise instruction indicates the source and destination operands data width keep the same which are encoded in the instruction. Non-widen float pointwise instruction includes mfadd/mfsub/mfmul/mfmax/mfmin.

```
#matrix-matrix fadd
mfadd.<h/s/d>.mm md, ms2, ms1
#matrix-vector fadd
mfadd.<h/s/d>.mv.i md, ms2, ms1[uimm3]

#matrix-matrix fsub
mfsub.<h/s/d>.mm md, ms2, ms1
#matrix-vector sub
mfsub.<h/s/d>.mv.i md, ms2, ms1[uimm3]

#matrix-matrix fmul
mfmul.<h/s/d>.mm md, ms2, ms1
#matrix-vector fmul
mfmul.<h/s/d>.mv.i md, ms2, ms1[uimm3]

#matrix-matrix fmax
mfmax.<h/s/d>.mm md, ms2, ms1
#matrix-vector fmax
mfmax.<h/s/d>.mv.i md, ms2, ms1[uimm3]

#matrix-matrix fmin
mfmin.<h/s/d>.mm md, ms2, ms1
#matrix-vector fmin
mfmin.<h/s/d>.mv.i md, ms2, ms1[uimm3]
```

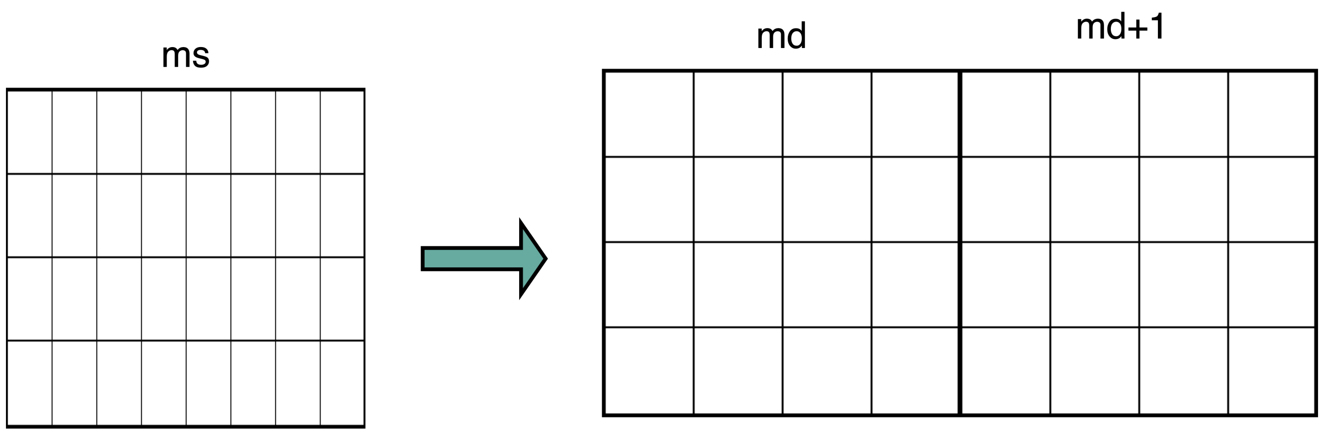
Widen float pointwise instruction indicates destination operand data width is twice that of the source operand. The data width of source operand is in instruction encoding. If the source element is fp16/bf16 floating-point data, the destination element is fp32 floating-point data, the instruction is classified as fp16/bf16 pointwise arithmetic instructions. If the source element is fp32 floating-point data, the destination element is fp64 floating-point data, the instruction is classified as fp32 pointwise arithmetic instructions. As data width for destination matrix is twice that of source matrix, two matrix register(register-pair) are used by destination matrix specified by md and md+1. Instructions specifying an odd-numbered md is reserved.

```
#matrix-matrix widen fadd
mfwadd.<h/s>.mm md, ms2, ms1
#matrix-vector widen fadd
mfwadd.<h/s>.mv.i md, ms2, ms1[uimm3]

#matrix-matrix widen fsub
mfwsub.<h/s>.mm md, ms2, ms1
#matrix-vector widen fsub
mfwsub.<h/s>.mv.i md, ms2, ms1[uimm3]

#matrix-matrix widen fmul
mfwmul.<h/s>.mm md, ms2, ms1
#matrix-vector widen fmul
mfwmul.<h/s>.mv.i md, ms2, ms1[uimm3]
```

The convert instruction converts each element from ms1 to destination register md1. Narrow conversion is supported by instruction mfncvt, source elements are twice as wide as destination elements, two matrix register(register-pair) are used by source matrix specified by ms and ms+1. Instructions specifying an odd-numbered ms is reserved.



Widen conversion is supported by instruction mfwcvt, destination elements are twice as wide as source elements, two matrix register(register-pair) are used by destination matrix specified by md and md+1. Instructions specifying an odd-numbered md is reserved.



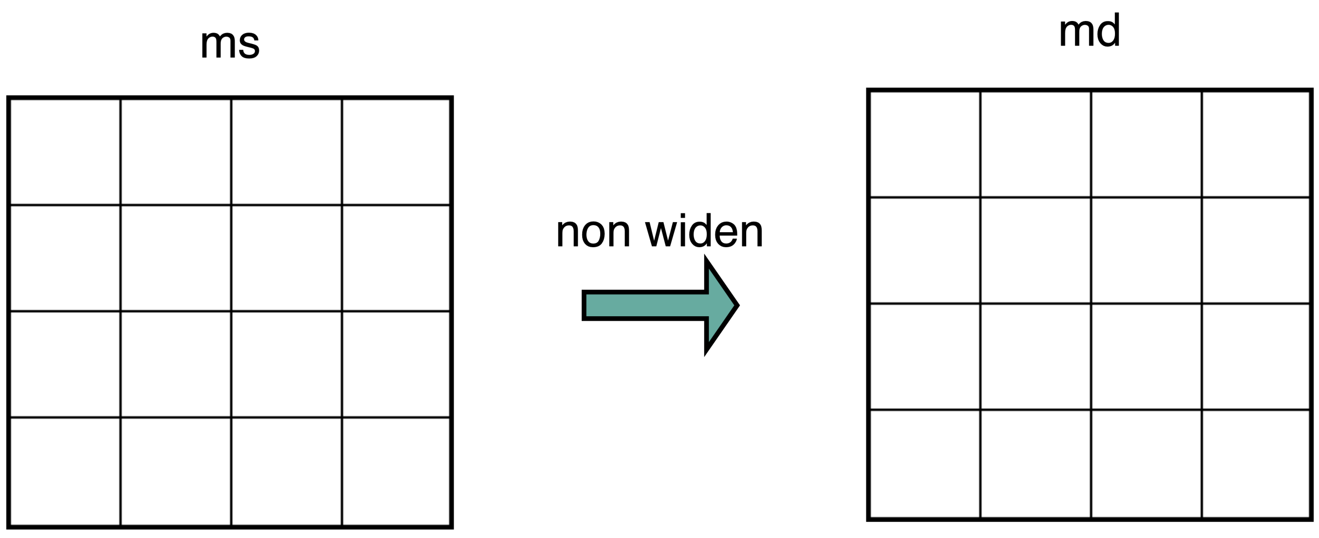
```
#matrix-matrix floating point narrow convert(fp32tofp16   fp64tofp32)
mfncvt.<s/d>.mm md, ms1

#matrix-matrix floating point widen convert(fp16tofp32 fp32tofp64)
mfwcvt.<h/s>.mm md, ms1
```

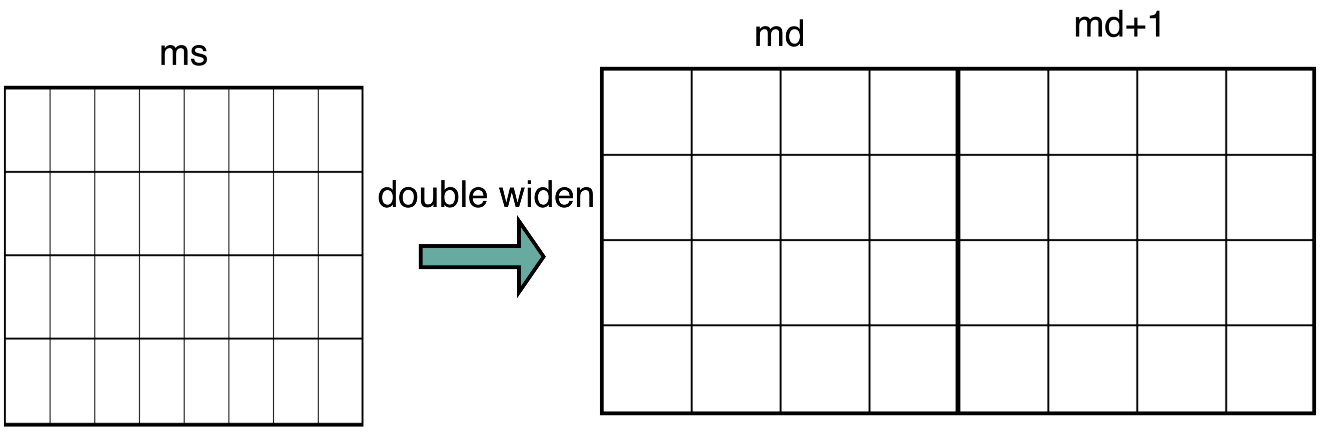
3.5.1. Float integer Conversion Instructions

Float integer conversion instructions are optionally supported.

- for conversion between fp32 and int32, legal if bit[29] and bit[4] of xmisa register are both set to 1, also legal if bit[29] and bit[7] of xmisa register are both set to 1, illegal In all other cases
- for conversion between fp16 and int8, legal if bit[29] and bit[3] of xmisa register are all set to 1, also legal if bit[29] and bit[6] of xmisa register are all set to 1, illegal In all other cases

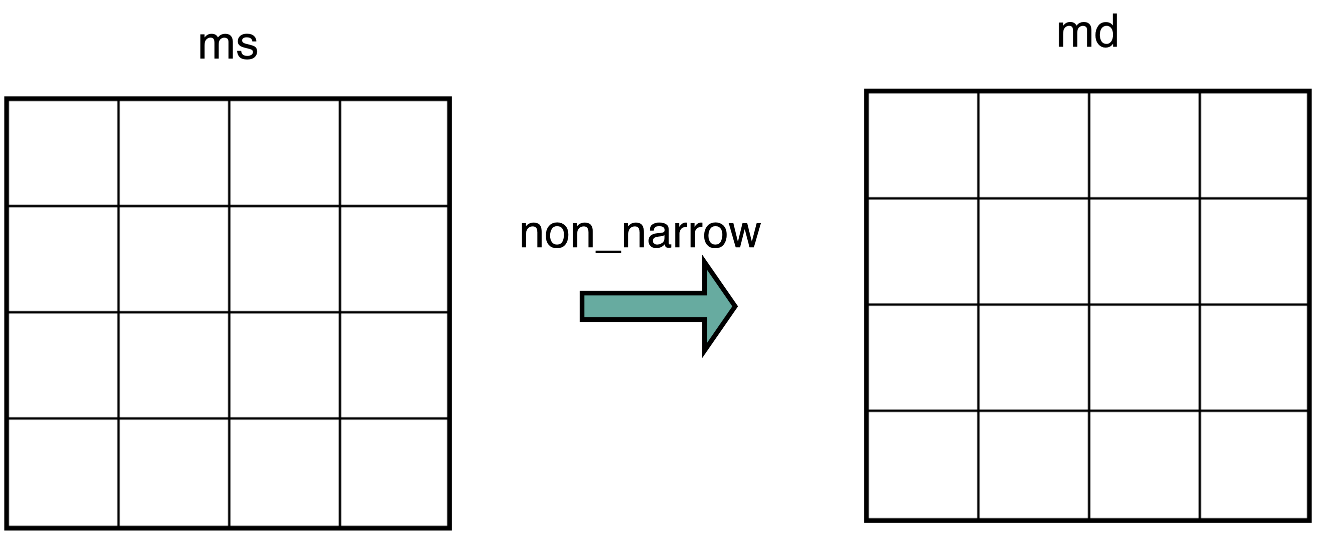
The conversion from integer to floating point supports non-widen conversion/double widen conversion. Integers support signed and unsigned integers. For double widen conversion, two matrix registers are used by destination matrix specified by md and md+1. Instructions specifying an odd-numbered md is reserved.

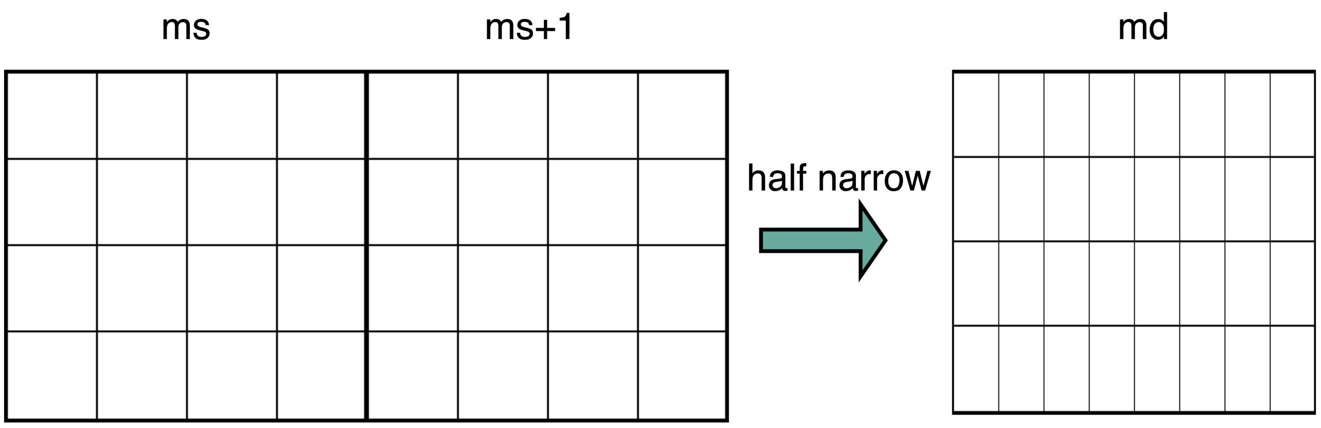




```
#matrix-matrix unsigned integer floating point convert(uint32 to fp32)
mufcvt.<w>.mm md, ms2
#matrix-matrix signed integer floating point widen convert(uint8 to fp16)
mufwcvt.<b>.mm md, ms2

#matrix-matrix signed integer floating point convert(sint32 to fp32)
msfcvt.<w>.mm md, ms2
#matrix-matrix signed integer floating point widen convert(sint8 to fp16)
msfwcvt.<b>.mm md, ms2
```

The conversion from floating point to integer supports non-narrow conversion/half narrow conversion. Integers support signed and unsigned integers. For half narrow conversion, two matrix registers are used by source matrix specified by ms and ms+1. Instructions specifying an odd-numbered ms is reserved.





```
#matrix-matrix floating point unsigned integer convert(fp32 to uint32)
mfucvt.<s>.mm md, ms2
#matrix-matrix floating point unsigned integer narrow convert(fp16 to uint8)
mfuncvt.<h>.mm md, ms2

#matrix-matrix floating point signed integer convert(fp32 to sint32)
mfscvt.<s>.mm md, ms2
#matrix-matrix floating point signed integer narrow convert(fp16 to sint8)
mfsncvt.<h>.mm md, ms2
```

3.6. Other Instructions

3.6.1. Mzero Instruction

Mzero instruction sets the destination register to zero.

```
#matrix-matrix
mzero md
```

3.6.2. Mrelease Instruction

Mrelease Instruction sets MS to Initial state.

```
mrelease
```

mrelease shares the encoding with mcfgi, with index filed is 3'b111.

3.6.3. Matrix Move Instructions

Matrix move instructions ignore matrix size configuration.

move between matrix registers

The mmov.mm instruction moves a whole matrix register to another matrix register.

The mmov.mv.i instruction moves and duplicates a vector to every row of the destination matrix register. The vector data is a row of matrix register, indexed by uimm3.The log2 (RLEN/32) bits are used.

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

move from GPR to matrix registers

The mdup<b/h/w/d>.m.x instruction moves and duplicates a scalar data to every element of the destination matrix register.

The mmov<b/h/w/d>.m.x instruction moves a scalar data to an element of the destination matrix register. The elements number within a matrix row is selected by rs1, modulo the number of such elements in a row. The row number is selected by rs1 , divided by the number of such elements in a row. The low $\log_2$(xmlenb/ element size) bits are used.

The scalar data is taken from the scalar x register specified by rs2 with XLEN data width. If data width < XLEN, the least-significant bits are copied and the upper bits are ignored. If data width > XLEN, the value is sign-extended.

```
#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
```

move from matrix registers to GPR

mmov<b/h/w/d>.x.m instruction moves a scalar data from a matrix register to a general purpose register specified by rd.

The scalar data is indexed by rs1. The elements number within a matrix row is selected by rs1, modulo the number of such elements in a row. The row number is selected by rs1, divided by the number of such elements in a row. The low $\log_2$(xmlenb/ element size) bits of rs1 are used.

If data width > XLEN, the least-significant XLEN bits are transferred and the upper bits are ignored. If data width < XLEN, the value is sign-extended to XLEN bits.

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

3.7. Matrix Register Overlap

Instructions support matrix source and destination registers overlap except matrix multiplication instructions.

Chapter 4. Instruction Format

Matrix instructions use custom-1 (0101011) as major opcode and the func3 is 3’b000. bit[26:25] is uop filed, indicating the operation type.

uop[1:0]	**type**	**meaning**
00	Matrix-Matrix(mm)	matrix computation, source and destination operands are matrix
01	Matrix-Vector(mv.i)	matrix computation, one source operand is vector, row index provided by uimm3
10	Matrix memory access	normal, streaming and whole regsiter loads/stores
11	Matrix MISC	special instructions and configuration instructions

The instruction formats are:

	31 27	**26 25**	**24**	**23 21**	**20**	**19**	**18**	**17 15**	**14 12**	**11 10**	**9 7**	**6 0**
Calc	func	01/00	size	ms2	ms1			md/ms3	func3	size	uimm3	major opcode
Load/Store	func	10	rs2			rs1			func3	size	md/ms3	major opcode
MISC	func	11	uimm7					0	func3	rd		major opcode
	func	11	0			rs1			func3	rd		major opcode
	func	11	0					md	func3	0		major opcode
	func	11	rs2			rs1			func3	size	md/ms3	major opcode

4.1. Arithmetic Instructions

The arithmetic instructions format:

31 28	**27 25**	**24**	**23 21**	**20 18**	**17 15**	**14 12**	**11 10**	**9 7**	**6 0**
func	uop	size	ms2	ms1	md/ms3	func3	size	uimm3	major opcode

Size field indicates the element, set to 0 if not needed.

size[1:0]	**element data width**
00	8-bit
01	16-bit
10	32-bit
11	64-bit

The instruction encoding list is in following tables.

4.1.1. Matrix Move

Move between matrix instructions and mzero(section 4.3.2) reuse arithmetic instruction format.

31 27	26 25	24	23 21	20 18	17 15	14 12	11 10	9 7	6 0	
00000	00	0	000	ms1	md	func3	00	001	major opcode	mmov.mm
00000	01	0	000	ms1	md	func3	00	uimm3	major opcode	mmov.mv.i

4.1.2. Matrix Multiplication

Matrix multiplication instructions use arithmetic instruction format where the func domain are 00001/00010.

31 27	26 25	24	23 21	20 18	17 15	14 12	11 10	9 7	6 0	
00001	00	0	ms2	ms1	md/ms3	func3	01	000	major opcode	fmmacc.h
00001	00	0	ms2	ms1	md/ms3	func3	10	000	major opcode	fmmacc.s
00001	00	0	ms2	ms1	md/ms3	func3	11	000	major opcode	fmmacc.d
00001	00	1	ms2	ms1	md/ms3	func3	01	000	major opcode	fwmmacc.h
00001	00	1	ms2	ms1	md/ms3	func3	10	000	major opcode	fwmmacc.s
00010	00	0	ms2	ms1	md/ms3	func3	00	000	major opcode	mmaqa.b
00010	00	0	ms2	ms1	md/ms3	func3	00	001	major opcode	mmaqau.b
00010	00	0	ms2	ms1	md/ms3	func3	00	010	major opcode	mmaqaus.b
00010	00	0	ms2	ms1	md/ms3	func3	00	011	major opcode	mmaqasu.b
00010	00	0	ms2	ms1	md/ms3	func3	01	000	major opcode	mmaqa.h
00010	00	0	ms2	ms1	md/ms3	func3	01	001	major opcode	mmaqau.h
00010	00	0	ms2	ms1	md/ms3	func3	01	010	major opcode	mmaqaus.h

00010	00	0	ms2	ms1	md/ms3	func3	01	011	major opcode	mmaqasu.h

4.1.3. Integer Pointwise

Integer pointwise instructions use arithmetic instruction format where the func domains are 00011/00100/00101/00110/00111/01000/01001/01010/01011/01100/01101/01110/01111.

31 27	26 25	24	23 21	20 18	17 15	14 12	11 10	9 7	6 0	
00011	00	0	ms2	ms1	md	func3	10	000	major opcode	madd.s.mm
00011	01	0	ms2	ms1	md	func3	10	uimm3	major opcode	madd.s.mv.i
00100	00	0	ms2	ms1	md	func3	10	000	major opcode	msub.s.mm
00100	01	0	ms2	ms1	md	func3	10	uimm3	major opcode	msub.s.mv.i
00101	00	0	ms2	ms1	md	func3	10	000	major opcode	msra.s.mm
00101	01	0	ms2	ms1	md	func3	10	uimm3	major opcode	msra.s.mv.i
00110	00	0	ms2	ms1	md	func3	10	000	major opcode	msrl.s.mm
00110	01	0	ms2	ms1	md	func3	10	uimm3	major opcode	msrl.s.mv.i
00111	00	0	ms2	ms1	md	func3	10	000	major opcode	msll.s.mm
00111	01	0	ms2	ms1	md	func3	10	uimm3	major opcode	msll.s.mv.i
01000	00	0	ms2	ms1	md	func3	10	000	major opcode	mn4clip.s.mm
01000	01	0	ms2	ms1	md	func3	10	uimm3	major opcode	mn4clip.s.mv.i
01001	00	0	ms2	ms1	md	func3	10	000	major opcode	mn4clipu.s.mm
01001	01	0	ms2	ms1	md	func3	10	uimm3	major opcode	mn4clipu.s.mv.i
01010	00	0	ms2	ms1	md	func3	10	000	major opcode	mmul.s.mm

01010	10	0	ms2	ms1	md	func3	10	uimm3	major opcode	mmul.s.mv.i
01011	00	0	ms2	ms1	md	func3	10	000	major opcode	mmulh.s.mm
01011	01	0	ms2	ms1	md	func3	10	uimm3	major opcode	mmulh.s.mv.i
01100	00	0	ms2	ms1	md	func3	10	000	major opcode	mmax.s.mm
01100	01	0	ms2	ms1	md	func3	10	uimm3	major opcode	mmax.s.mv.i
01101	00	0	ms2	ms1	md	func3	10	000	major opcode	mumax.s.mm
01101	01	0	ms2	ms1	md	func3	10	uimm3	major opcode	mumax.s.mv.i
01110	00	0	ms2	ms1	md	func3	10	000	major opcode	mmin.s.mm
01110	01	0	ms2	ms1	md	func3	10	uimm3	major opcode	mmin.s.mv.i
01111	00	0	ms2	ms1	md	func3	10	000	major opcode	mumin.s.mm
01111	01	0	ms2	ms1	md	func3	10	uimm3	major opcode	mumin.s.mv.i
00011	00	0	ms2	ms1	md	func3	11	000	major opcode	madd.d.mm
00011	01	0	ms2	ms1	md	func3	11	uimm3	major opcode	madd.d.mv.i
00100	00	0	ms2	ms1	md	func3	11	000	major opcode	msub.d.mm
00100	01	0	ms2	ms1	md	func3	11	uimm3	major opcode	msub.d.mv.i
00101	00	0	ms2	ms1	md	func3	11	000	major opcode	msra.d.mm
00101	01	0	ms2	ms1	md	func3	11	uimm3	major opcode	msra.d.mv.i
00110	00	0	ms2	ms1	md	func3	11	000	major opcode	msrl
00110	01	0	ms2	ms1	md	func3	11	uimm3	major opcode	msrl
00111	00	0	ms2	ms1	md	func3	11	000	major opcode	msll

00111	01	0	ms2	ms1	md	func3	11	uimm3	major opcode	msll
01000	00	0	ms2	ms1	md	func3	11	000	major opcode	mn4clip.d.mm
01000	01	0	ms2	ms1	md	func3	11	uimm3	major opcode	mn4clip.d.mv.i
01001	00	0	ms2	ms1	md	func3	11	000	major opcode	mn4clipu.d.mm
01001	01	0	ms2	ms1	md	func3	11	uimm3	major opcode	mn4clipu.d.mv.i
01010	00	0	ms2	ms1	md	func3	11	000	major opcode	mmul.d.mm
01010	01	0	ms2	ms1	md	func3	11	uimm3	major opcode	mmul.d.mv.i
01011	00	0	ms2	ms1	md	func3	11	000	major opcode	mmulh.d.mm
01011	01	0	ms2	ms1	md	func3	11	uimm3	major opcode	mmulh.d.mv.i
01100	00	0	ms2	ms1	md	func3	11	000	major opcode	mmax.d.mm
01100	01	0	ms2	ms1	md	func3	11	uimm3	major opcode	mmax.d.mv.i
01101	00	0	ms2	ms1	md	func3	11	000	major opcode	mumax.d.mm
01101	01	0	ms2	ms1	md	func3	11	uimm3	major opcode	mumax.d.mv.i
01110	00	0	ms2	ms1	md	func3	11	000	major opcode	mmin.d.mm
01110	01	0	ms2	ms1	md	func3	11	uimm3	major opcode	mmin.d.mv.i
01111	00	0	ms2	ms1	md	func3	11	000	major opcode	mumin.d.mm
01111	01	0	ms2	ms1	md	func3	11	uimm3	major opcode	mumin.d.mv.i

4.1.4. Float Pointwise

Float pointwise instructions use arithmetic instruction format where the func domains are 10000/10001/10010/10011/10100/10101.

31 27	26 25	24	23 21	20 18	17 15	14 12	11 10	9 7	6 0	
10000	00	0	ms2	ms1	md	func3	01	000	major opcode	mfadd.h.mm
10000	01	0	ms2	ms1	md	func3	01	uimm3	major opcode	mfadd.h.mv.i
10000	00	1	ms2	ms1	md	func3	01	000	major opcode	mfwadd.h.mm
10000	01	1	ms2	ms1	md	func3	01	uimm3	major opcode	mfwadd.h.mv.i
10001	00	0	ms2	ms1	md	func3	01	000	major opcode	mfsub.h.mm
10001	01	0	ms2	ms1	md	func3	01	uimm3	major opcode	mfsub.h.mv.i
10001	00	1	ms2	ms1	md	func3	01	000	major opcode	mfwsub.h.mm
10001	01	1	ms2	ms1	md	func3	01	uimm3	major opcode	mfwsub.h.mv.i
10010	00	0	ms2	ms1	md	func3	01	000	major opcode	mfmul.h.mm
10010	01	0	ms2	ms1	md	func3	01	uimm3	major opcode	mfmul.h.mv.i
10010	00	1	ms2	ms1	md	func3	01	000	major opcode	mfwmul.h.mm
10010	01	1	ms2	ms1	md	func3	01	uimm3	major opcode	mfwmul.h.mv.i
10011	00	0	ms2	ms1	md	func3	01	000	major opcode	mfmax.h.mm
10011	01	0	ms2	ms1	md	func3	01	uimm3	major opcode	mfmax.h.mv.i
10100	00	0	ms2	ms1	md	func3	01	000	major opcode	mfmin.h.mm
10100	01	0	ms2	ms1	md	func3	01	uimm3	major opcode	mfmin.h.mv.i
10000	00	0	ms2	ms1	md	func3	10	000	major opcode	mfadd.s.mm
10000	01	0	ms2	ms1	md	func3	10	uimm3	major opcode	mfadd.s.mv.i
10000	00	1	ms2	ms1	md	func3	10	000	major opcode	mfwadd.s.mm

10000	01	1	ms2	ms1	md	func3	10	uimm3	major opcode	mfwadd.s.mv.i
10001	00	0	ms2	ms1	md	func3	10	000	major opcode	mfsub.s.mm
10001	01	0	ms2	ms1	md	func3	10	uimm3	major opcode	mfsub.s.mv.i
10001	00	1	ms2	ms1	md	func3	10	000	major opcode	mfwsub.s.mm
10001	01	1	ms2	ms1	md	func3	10	uimm3	major opcode	mfwsub.s.mv.i
10010	00	0	ms2	ms1	md	func3	10	000	major opcode	mfmul.s.mm
10010	01	0	ms2	ms1	md	func3	10	uimm3	major opcode	mfmul.s.mv.i
10010	00	1	ms2	ms1	md	func3	10	000	major opcode	mfwmul.s.mm
10010	01	1	ms2	ms1	md	func3	10	uimm3	major opcode	mfwmul.s.mv.i
10011	00	0	ms2	ms1	md	func3	10	000	major opcode	mfmax.s.mm
10011	01	0	ms2	ms1	md	func3	10	uimm3	major opcode	mfmax.s.mv.i
10100	00	0	ms2	ms1	md	func3	10	000	major opcode	mfmin.s.mm
10100	01	0	ms2	ms1	md	func3	10	uimm3	major opcode	mfmin.s.mv.i
10000	00	0	ms2	ms1	md	func3	11	000	major opcode	mfadd.d.mm
10000	01	0	ms2	ms1	md	func3	11	uimm3	major opcode	mfadd.d.mv.i
10001	00	0	ms2	ms1	md	func3	11	000	major opcode	mfsub.d.mm
10001	01	0	ms2	ms1	md	func3	11	uimm3	major opcode	mfsub.d.mv.i
10010	00	0	ms2	ms1	md	func3	11	000	major opcode	mfmul.d.mm
10010	01	0	ms2	ms1	md	func3	11	uimm3	major opcode	mfmul.d.mv.i
10011	00	0	ms2	ms1	md	func3	11	000	major opcode	mfmax.d.mm

10011	01	0	ms2	ms1	md	func3	11	uimm3	major opcode	mfmax.d.mv.i
10100	00	0	ms2	ms1	md	func3	11	000	major opcode	mfmin.d.mm
10100	01	0	ms2	ms1	md	func3	11	uimm3	major opcode	mfmin.d.mv.i
10101	00	0	000	ms1	md	func3	10	000	major opcode	mfncvt.s.mm
10101	00	0	000	ms1	md	func3	11	000	major opcode	mfncvt.d.mm
10101	00	0	000	ms1	md	func3	01	001	major opcode	mfwcvt.h.mm
10101	00	0	000	ms1	md	func3	10	001	major opcode	mfwcvt.s.mm

4.1.5. Float Integer Conversion

Float integer conversion instructions use arithmetic instruction format where the func domains are 10110.

31 27	26 25	24	23 21	20 18	17 15	14 12	11 10	9 7	6 0	
10110	00	0	000	ms1	md	func3	10	000	major opcode	mufcvt.w.mm
10110	00	0	000	ms1	md	func3	00	001	major opcode	mufwcvt.b.mm
10110	00	0	000	ms1	md	func3	10	100	major opcode	msfcvt.w.mm
10110	00	0	000	ms1	md	func3	00	101	major opcode	msfwcvt.b.mm
10110	00	0	001	ms1	md	func3	10	000	major opcode	mfucvt.s.mm
10110	00	0	001	ms1	md	func3	01	001	major opcode	mfuncvt.h.mm
10110	00	0	001	ms1	md	func3	10	100	major opcode	mfscvt.s.mm
10110	00	0	001	ms1	md	func3	01	101	major opcode	mfsncvt.h.mm

4.2. Matrix Load/Store Instructions

The matrix load/store instruction format:

31 27	26 25	24 20	19 15	14 12	11 10	9 7	6 0
func	10	rs2	rs1	func3	size	md/ms3	major opcode

bit[27] = 1 indicates store operations, while bit[27] = 0 indicates load operations. bit[28] = 1 indicates streaming memory access and bit[29]=1 indicates whole register memory access.

31 27	26 25	24 20	19 15	14 12	11 10	9 7	6 0	
00000	10	rs2	rs1	func3	size	md	major opcode	mld<b/h/w/d>
00001	10	rs2	rs1	func3	size	ms3	major opcode	mst<b/h/w/d>
00010	10	rs2	rs1	func3	size	md	major opcode	mld.<b/h/w/d>.s
00011	10	rs2	rs1	func3	size	ms3	major opcode	mst.<b/h/w/d>.s
00100	10	{00,nf}	rs1	func3	size	md	major opcode	mld<1/2/4/8>m<b/h/w/d>
00101	10	{00,nf}	rs1	func3	size	md	major opcode	mst<1/2/4/8>m <b/h/w/d>

4.3. Other Instructions

4.3.1. configuration

The uop of configuration instructions is 2’b11.

31	30 27	26 25	24 20	19 15	14 12	11 7	6 0	
0	0001	11	{uimm7,000}		func3	rd	major opcode	mcfgki
0	0011	11	{uimm7,000}		func3	rd	major opcode	mcfgmi
0	0101	11	{uimm7,000}		func3	rd	major opcode	mcfgni
1	0001	1	00000	rs1	func3	rd	major opcode	mcfgk
1	0011	11	00000	rs1	func3	rd	major opcode	mcfgm
1	0101	11	00000	rs1	func3	rd	major opcode	mcfgn
1	1111	11	00000	rs1	func3	rd	major opcode	mcfg

4.3.2. mzero

The mzero instruction shares the 2’b00 uop with the arithmetic instructions.

31 27	26 25	24	23 21	20 18	17 15	14 12	11 10	9 7	6 0	
11111	00	0	000	000	md	func3	00	000	major code	mzero

4.3.3. mrelease

The mrelease instruction uses the configuration 2’b11 uop.

31	30 27	26 25	24 20	19 15	14 12	11 7	6 0	
0	1111	11	00000	00000	func3	00000	major opcode	mrelease

4.3.4. move from matrix

31 27	26 25	24	23 21	20	19 15	14 12	11 7	6 0	
00000	11	0	ms2	0	rs1	func3	rd	major opcode	mmovb.x.m
00000	11	0	ms2	1	rs1	func3	rd	major opcode	mmovh.x.m
00000	11	1	ms2	0	rs1	func3	rd	major opcode	mmovw.x.m
00000	11	1	ms2	1	rs1	func3	rd	major opcode	mmovd.x.m

4.3.5. move GPR to matrix

31 28	27 25	24 20	19 15	14 12	11 10	9 7	6 0	
00010	11	rs2	0000	func3	00	md	major opcode	mdupb.m.x
00010	11	rs2	0000	func3	01	md	major opcode	mduph.m.x
00010	11	rs2	0000	func3	10	md	major opcode	mdupw.m.x
00010	11	rs2	0000	func3	11	md	major opcode	mdupd.m.x
00100	11	rs2	rs1	func3	00	md	major opcode	mmovb.m.x
00100	11	rs2	rs1	func3	01	md	major opcode	mmovh.m.x
00100	11	rs2	rs1	func3	10	md	major opcode	mmovw.m.x
00100	11	rs2	rs1	func3	11	md	major opcode	mmovd.m.x

Chapter 5. Standard Matrix Extensions

5.1. Bf16 Extension

The 16-bit float operand can be seen as Bfloat-16 format. The bf16 extension adds a bit in FCSR, the 16-bit float data is bf16 if the bit is set to 1.

```
#float matrix multiplication, md = md + ms1*ms2
fmmacc.h md, ms2, ms1
#float matrix multiplication, output widen, md = md + ms1*ms2
fwmmacc.h md, ms2, ms1
```

16 bit float pointwise operations and 16 bit float integer conversion operations are optionally supported.

5.2. Int4 Extension

For int4 matrix multiplication, the source operand is 4-bit width and the destination is 32-bit width. Two int4 data pair are considered as an 8-bit element, the sizeK is set as int8 data width, so the K should be an even value, otherwise reserved.

- pmmaqa.b/pmmaqau.b/pmmaqaus.b/pmmaqasu.b: int4 8x widen matrix multiplication and add , illegal if bit[0] of xmisa register is 0
- for conversion between int8 and int4, legal if bit[28] and bit[0] of xmisa register are both set to 1, also legal if bit[28] and bit[0] of xmisa register are both set to 1, illegal In all other cases

The maximum matrix shape is:

- matrixA: M ⇐ RLEN/32, K ⇐ RLEN/4
- matrixB: N ⇐ RLEN/32, K ⇐ RLEN/4
- matrixC: M ⇐ RLEN/32, N ⇐ RLEN/32

```
#4bit data width
#signed matrix multiply
pmmaqa.b ms3, ms2, ms1
#unsigned matrix multiply
pmmaqau.b ms3, ms2, ms1
#unsigned-signed matrix multiply
pmmaqaus.b ms3, ms2, ms1
#signed-unsigned matrix multiply
pmmaqasu.b ms3, ms2, ms1
```

	matrix A			**matrix B**			**matrix C**		
RLEN	M	K	data width	N	K	data width	M	N	data width

	matrix A			**matrix B**			**matrix C**		
128	4	32	512 bits	4	32	512 bits	4	4	512 bits
256	8	64	2048 bits	8	64	2048 bits	8	8	2048 bits
512	16	128	8192 bits	16	128	8192 bits	16	16	8192 bits

The int4 matrix multiplication instruction uses arithmetic instructions format, the code is as follows:

31 27	**26 25**	**24**	**23 21**	**20 18**	**17 15**	**14 12**	**11 10**	**9 7**	**6 0**	
00010	00	1	ms2	ms1	md/ms3	func3	00	000	major opcode	pmmaqa.b
00010	00	1	ms2	ms1	md/ms3	func3	00	001	major opcode	pmmaqau.b
00010	00	1	ms2	ms1	md/ms3	func3	00	010	major opcode	pmmaqaus.b
00010	00	1	ms2	ms1	md/ms3	func3	00	011	major opcode	pmmaqasu.b

Integer conversion instruction is optionally supported in int4 extension. As data width for destination matrix is twice that of source matrix, two matrix register(register-pair) are used by destination matrix specified by md and md+1. Instructions specifying an odd-numbered md is reserved.

```
#4bit data width
#signed matrix multiply
pmmaqa.b ms3, ms2, ms1
#unsigned matrix multiply
pmmaqau.b ms3, ms2, ms1
#unsigned-signed matrix multiply
pmmaqaus.b ms3, ms2, ms1
#signed-unsigned matrix multiply
pmmaqasu.b ms3, ms2, ms1
```

The integer conversion instruction uses arithmetic instructions format, the code is as follows:

31 27	26 25	24	23 21	20 18	17 15	14 12	11 10	9 7	6 0	
10111	00	1	000	ms1	md	func3	00	000	major opcode	pmswcvt
10111	00	1	000	ms1	md	func3	00	001	major opcode	pmuwcvt

Chapter 6. Instruction List

There are 39 instructions extended for matrix, some are optional for hardware implementations.

catagory	instructions	
matrix multiplication(10)	fmmacc.	float matrix multiplication
	fwmmacc.	float widen matrix multiplication
	mmaqa.	signed integer 4x matrix multiplication
	mmaqau.	unsigned integer 4x matrix multiplication
	mmaqasu.	signed-unsigned integer 4x matrix multiplication
	mmaqaus.	unsigned-signed integer 4x matrix multiplication
	pmmaqa.	int4 signed integer matrix multiplication
	pmmaqau.	int4 unsigned integer matrix multiplication
	pmmaqasu.	int4 signed -unsigned integer matrix multiplication
	pmmaqaus.	int4 unsigned -signed integer matrix multiplication
matrix load/store(6)	mld.<b/h/w/d>	matrix load to matrix registers
	mst.<b/h/w/d>	matrix store from matrix registers
	mld.<b/h/w/d>.s	stream load to matrix registers
	mst.<b/h/w/d>.s	stream matrix store from matrix registers
	mld<1/2/4/8>m.<b/h/w/d>	load to whole matrix register
	mst<1/2/4/8>m.<b/h/w/d>	store to whole matrix register
matrix movement(1)	mmov.mm/mmov.mv.x mmov.mv.i/ mmov.<b/h/w/d>.m.x/ mdup.<b/h/w/d>.m.x/ mmov.<b/h/w/d>.x.m	move from/to matrix registers
matrix integer pointwise arithmetic (7)	madd/msub.<s/d>.<mm/mv>.<i>	
	mshift.<s/d>.<mm/mv>.<i>	
	mn4clip.<s/d>.<mm/mv>.<i>	
	mn4clipu.<s/d>.<mm/mv>.<i>	
	mmul.<s/d>.<mm/mv>.<i>	
	mmulh.<s/d>.<mm/mv>.<i>	
	mmax/mmin/mumax/mumin/<s/d>.<mm/mv>.<i>	

catagory	instructions	
matrix float pointwise arithmetic (7)	mfadd/mfsub.<h/s/d>.<mm/mv>.<i>	
	mfwadd/mfwsub.<h/s>.<mm/mv>.<i>	
	mfmul.<h/s/d>.<mm/mv>.<i>	
	mfwmul.<h/s>.<mm/mv>.<i>	
	mfmax/mfmin.<h/s/d>.<mm/mv>.<i>	
	mfncvt.<s/d>.mm	
	mfwcvt.<h/s>.mm	
matrix float integer conversion (4)	mufcvt/msfcvt.s.mm	
	mufwcvt/msfwcvt.b.mm	
	mfucvt/mfscvt.s.mm	
	mfuncvt/mfsncvt.b.mm	
configuration(2)	mcfgi	
	mcfg	
others(2)	mrelease	clear ms to CLEAN
	mzero	

Chapter 7. Matrix Memory Model

Matrix memory instructions appear to execute in program order on the local hart.

Memory operations for each element are unordered within the instruction.

Matrix memory instructions follow RVWMO at the instruction level. Except for when a pair of read access the same locations, and at least one of the reads is generated by matrix instructions, they will not necessarily appear to execute in the same order by other processors.