



RISC-V Matrix Specification Proposal

Version v0.6.0, 2025-02-11: Draft

Table of Contents

Preamble	1
Copyright and license information	2
Contributors.....	3
1. Introduction	4
2. Implementation-defined Constant Parameters.....	5
3. Programmer's Model.....	7
3.1. Matrix Registers	7
3.2. Matrix ISA Register, misa.....	9
3.3. Matrix Tile Size Registers, mtilem/mtilek/mtilen.....	11
3.4. Matrix Control and Status Register, mcsr.....	11
3.5. Matrix Fixed-Point Rounding Mode, xmfrm	11
3.6. Matrix Fixed-Point Saturation Flag, xmsat	12
3.7. Matrix Float-Point Exception Flags, xmfflags	12
3.8. Matrix Float-Point Rounding Mode, xmfrm	12
3.9. Matrix Saturation Mode Enable, xmsaten.....	13
3.10. Matrix Context Status in mstatus and sstatus	14
4. Instruction Format	15
4.1. 32-bit instruction	15
4.1.1. Configuration Instructions	16
4.1.2. Load/Store Instructions	16
4.1.3. Matrix Multiplication Instructions	17
4.1.4. MISC Instructions.....	18
4.1.5. Element-wise Instructions.....	19
4.2. 64-bit instruction	20
4.2.1. Configuration Instructions	20
4.2.2. Load/Store Instructions	20
4.2.3. Matrix Multiplication Instructions	20
4.2.4. MISC Instructions.....	21
4.2.5. Element-wise Instructions.....	21
5. Instructions	23
5.1. Configuration Instructions.....	23
5.1.1. Mtilem/n/k Configuration Instructions	23
5.1.2. Mrelease Instruction	23
5.2. Matrix Multiplication Instructions	23
5.2.1. Float Matrix Multiplication(non-widen).....	26
5.2.2. Float Matrix Multiplication(double-widen)	29
5.2.3. Float Matrix Multiplication(quad-widen).....	32

5.2.4. Integer Matrix Multiplication	34
5.2.5. Examples Summary	35
5.3. Load and Store Instructions	35
5.3.1. Load Instructions	35
5.3.2. Store Instructions	36
5.3.3. Load Transposed Instructions	37
5.3.4. Store Transposed Instructions	37
5.3.5. Whole Matrix Load & Store Instructions	38
5.3.6. Examples for Load and Store Instructions	38
5.4. MISC Instructions	40
5.4.1. Mzero Instructions	40
5.4.2. Data Move Instructions	40
Data Move Instructions between Matrix Registers	40
Data Move Instructions between Integer and Matrix	41
5.4.3. Data Broadcast Instructions	42
5.4.4. Matrix Pack Instructions	42
5.4.5. Matrix Slide Instructions	43
5.5. Element-Wise Instructions	44
5.5.1. Integer Arithmetic Instructions	44
5.5.2. Float-point Arithmetic Instructions	46
5.5.3. Type-Convert Instructions	48
Float-point Conversion Instructions	48
Float-point Interger Instructions	50
Fixed-point Conversion Instructions	52
6. Matrix Extension	54
6.1. Zmint4: INT4 Extension	54
7. Matrix Memory Model	55
8. Instruction List	56

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.

We will be very grateful to the huge number of other people who will have helped to improve this specification through their comments, reviews, feedback and questions.

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. This extension defines extra matrix registers, including matrix multiplication instructions($C += A \times BT$), matrix load/store instructions, matrix element-wise operation instructions, and matrix misc instructions, which supports matrix multiplication and other basic matrix operation.

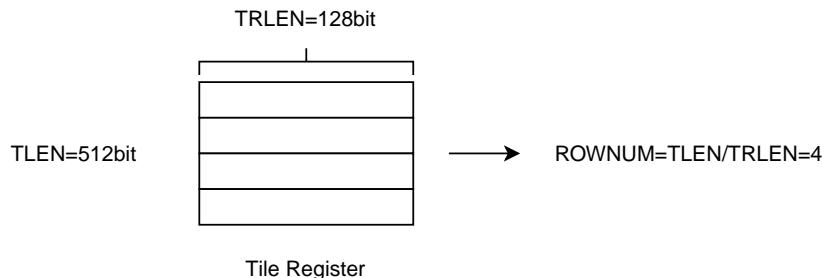
Chapter 2. Implementation-defined Constant Parameters

Each hart supporting a matrix extension defines three parameters:

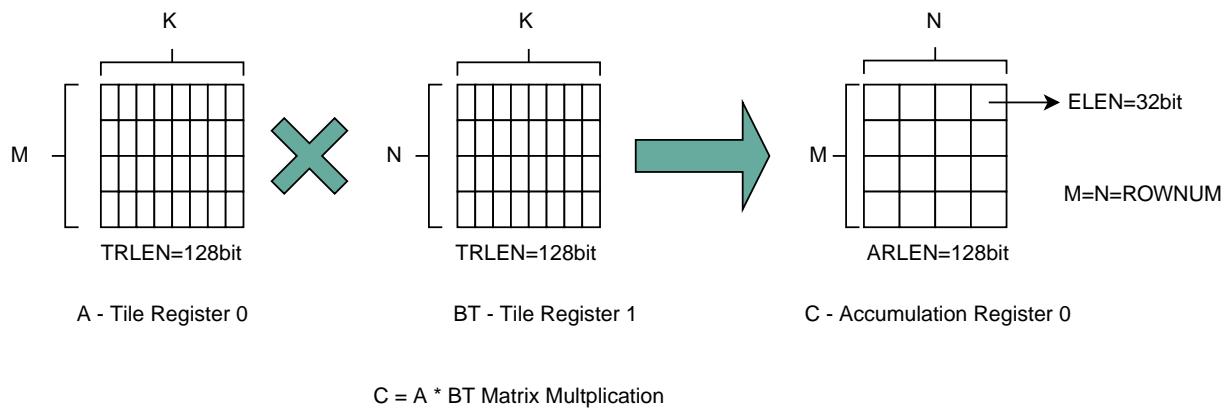
1. ELEN: The maximum size in bits of a matrix element that any operation can produce or consume, $ELEN \geq 8$, which must be a power of 2.
2. TLEN: The number of bits in a single matrix tile register, TLEN, which must be a power of 2, and must be no greater than 2^{32} .
3. TRLEN: The number of bits in a row of a single matrix tile register, TRLEN, which must be a power of 2, and must be no greater than 2^{16} .

The following parameter can be derived from the above fundamental parameters:

1. ROWNUM: The number of rows of tile register and accumulation register, equals to $TLEN/TRLEN$, which must be a power of 2.
2. ARLEN: The number of bits in a row of a single matrix accumulation register, ARLEN, which must be a power of 2, and must be no greater than 2^{16} . $ARLEN = TLEN/TRLEN * ELEN$.
3. ALEN: The number of bits in a single matrix accumulation register, ALEN, which must be a power of 2, and must be no greater than 2^{32} , $ALEN = ARLEN * ROWNUM$.



Tile Register Shape



The picture show examples for the tile register shape and the $C += A \times BT$ matrix multiplication operation. For a tile register, when TLEN is 512bit and TRLEN is 128bit, ROWNUM is 4.

For $C += A \times BT$, A matrix is stored in tile register 0 in normal format($M*K$), and B matrix are stored in tile register 1 in transposed format($N*K$). The shape of C matrix is $M*N$. When the ELEN is 32bit, ARLEN = TLEN/TRLEN * ELEN = $512/128 * 32 = 128$, and ALEN = ARLEN * ROWNUM = $128*4 = 512$.

Chapter 3. Programmer's Model

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

Address	Privilege	Name	Description
0x802	URW	xmcsr	matrix control and status register
0x803	URW	mtilem	Tile length in m direction.
0x804	URW	mtilen	Tile length in n direction.
0x805	URW	mtilek	Tile length in k direction.
0xcc0	URO	xmisa	matrix instruction subset
0xcc1	URO	xtlenb	tile register size in byte, ROWNUM*TRLEN/8
0xcc2	URO	xtrlenb	tile register row size in byte, TRLEN/8
0xcc3	URO	xalenb	accumulation register size in byte, ROWNUM ² * ELEN/8

Matrix extention gives seperate CSR address to allow independent access of xmfrm/xmsat/xmfflags/xmfrm/xmsat which is also reflected as a field in xmcsr.

Address	Privilege	Name	Description
0x806	URW	xmxrm	Fixed-point rounding mode
0x807	URW	xmsat	Fixed-point accrued saturation flag
0x808	URW	xmfflags	float-point accrued exception flags
0x809	URW	xmfrm	float-point rounding mode
0x80a	URW	xmsaten	saturation mode enable for fp8 or integer matrix multiplication instructions

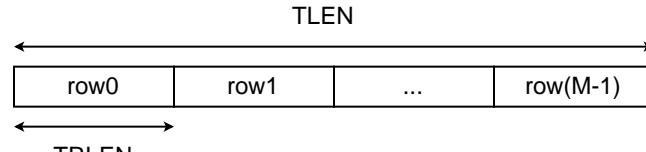
3.1. Matrix Registers

The matrix extension adds 4 architectural **Tile Registers** (tr0-tr3) for input tile matrices and 4 architectural **Accumulation Registers** (acc0-acc3) for output accumulation matrices.

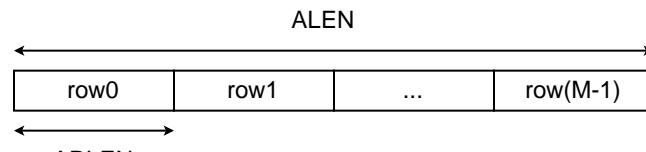
A **Tile Register** has a fixed TLEN bits of state, where each row has TRLEN bits. As a result, there are TLEN / TRLEN rows for each tile register in logic. Each tile register comprises [M x K] elements, K is TRLEN / element width.

An **Accumulation Register** has a fixed ALEN bits of state, where each row has ARLEN bits. As a result, there are ALEN / ARLEN, which is equal to TLEN/TRLEN rows for each acculation register in logic.

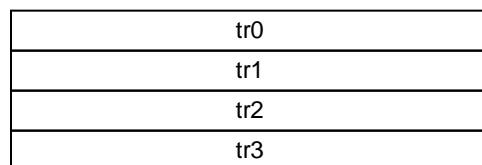
The ARLEN is derived by TLEN/TRLEN * ELEN, which means the element size of accumulation registers should not be bigger than ELEN.



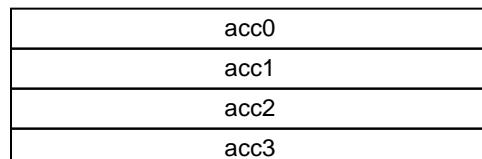
Tile Register Structure



Accumulation Register Structure



Tile Registers



Accumulation Registers

The size of the matrix registers varies as following when ELEN is 32bit and TLEN is 512bit.

SOURCE	TLEN	TRLEN	ALEN	M/NxK	MxN	Notes
32bit	512	32	8192	16x1	16x16	Outer Product
32bit	512	64	2048	8x2	8x8	
32bit	512	128	512	4x4	4x4	
32bit	512	256	128	2x8	2x2	
32bit	512	512	32	1x16	1x1	Inner Product
16bit	512	32	8192	16x2	16x16	
16bit	512	64	2048	8x4	8x8	

SOURCE	TLEN	TRLEN	ALEN	M/NxK	MxN	Notes
16bit	512	128	512	4x8	4x4	
16bit	512	256	128	2x16	2x2	
16bit	512	512	32	1x32	1x1	
8bit	512	32	8192	16x4	16x16	
8bit	512	64	2048	8x8	8x8	
8bit	512	128	512	4x16	4x4	
8bit	512	256	128	2x32	2x2	
8bit	512	512	32	1x64	1x1	

3.2. Matrix ISA Register, misa

The read-only XLEN-wide CSR, `misa`, specifying the supported matrix arithmetic feature of the current hardware implementation.

Bits	Feature	Support Type
XLEN-1	miew	optional
XLEN-2	mfew	optional
XLEN-3	mfic	optional
XLEN-4:10	0	reserved
9	mmf8f32	compulsory
8	mmf32f64	optional
7	mmbf16f32	compulsory
6	mmf16f32	compulsory
5	mmf8bf16	compulsory
5	mmf8f16	compulsory
4	mmf64f64	optional
3	mmf32f32	optional
2	mmf16f16	compulsory
1	mmi8i32	compulsory
0	mmi4i32	optional

For each field, a value 0 means the corresponding feature is not supported, a value 1 means the corresponding feature is supported.

For **mmi4i32** field, a value 1 means the corresponding int4/uint4 oct-widen matrix multiplication instructions are supported. The source operands type of this field are (su)int4 and the destination operand is int32. This field is optional for hardware implementation.

For **mmi8i32** field, a value 1 means the corresponding int8/uint8 quad-widen matrix multiplication instructions are supported. The source operands type of this field are (su)int8 and the destination operand is int32. This field is compulsory for hardware implementation.

For **mmf16f16** field, a value 1 means the corresponding fp16 no-widen matrix multiplication instructions are supported. The source operands type of this field are fp16 and the destination operand is fp16. This field is compulsory for hardware implementation.

For **mmf32f32** field, a value 1 means the corresponding fp32 no-widen matrix multiplication instructions are supported. The source operands type of this field are fp32 and the destination operand is fp32. This field is optional for hardware implementation.

For **mmf64f64** field, a value 1 means the corresponding fp64 no-widen matrix multiplication instructions are supported. The source operands type of this field are fp64 and the destination operand is fp64. This field is optional for hardware implementation.

For **mmf8f16** field, a value 1 means the corresponding fp8 double-widen matrix multiplication instructions are supported. The source operands type of this field are fp8(E4M3 and E5M2) and the destination operand is fp16. This field is compulsory for hardware implementation.

For **mmf8bf16** field, a value 1 means the corresponding fp8 double-widen matrix multiplication instructions are supported. The source operands type of this field are fp8(E4M3 and E5M2) and the destination operand is bf16. This field is compulsory for hardware implementation.

For **mmf16f32** field, a value 1 means the corresponding fp16 double-widen matrix multiplication instructions are supported. The source operands type of this field are fp16 and the destination operand is fp32. This field is compulsory for hardware implementation.

For **mmbf16f32** field, a value 1 means the corresponding bf16 double-widen matrix multiplication instructions are supported. The source operands type of this field are bf16 and the destination operand is fp32. This field is compulsory for hardware implementation.

For **mmf32f64** field, a value 1 means the corresponding fp32 double-widen matrix multiplication instructions are supported. The source operands type of this field are fp32 and the destination operand is fp64. This field is optional for hardware implementation.

For **mmf8f32** field, a value 1 means the corresponding fp32 quad-widen matrix multiplication instructions are supported. The source operands type of this field are fp8 and the destination operand is fp32. This field is compulsory for hardware implementation.

For **mfic** field, a value 1 means the conversion between integer and float point extension is supported. This field is optional for hardware implementation.

For **mfew** field, a value 1 means float point element wise extension(including arithmetic instruction and conversion instructions) is supported, including the . This field is optional for hardware implementation.

For **miew** field, a value 1 means integer element wise extension(including arithmetic instruction and conversion instructions) is supported. This field is optional for hardware implementation.

3.3. Matrix Tile Size Registers, **mtilem/mtilek/mtilen**

The XLEN-bit-wide read-only **mtilem/mtilek/mtilen** CSRs can only be updated by the **msettile{m|k|n}{i}** instructions. The registers holds 3 unsigned integers specifying the tile shapes for tiled matrix.

3.4. Matrix Control and Status Register, **mcsr**

The **xmcsr** CSR is a WARL read-write register. Bits[XLEN-1:12] are reserved and should be written with zero. The **mcsr** register has 5 fields. The xmxrm, xmsat, xmfflags, xmfrm, xmsaten separate CSRs can also be accessed via fields in the matrix control and status CSR, xmcsr.

Table 1. xmcsr register layout

Bits	Name	Description
XLEN-1:12	0	Reserved if non-zero.
11	xmsaten	integer and fp8 saturation mode.
10:8	xmfrm	Float-point arithmetic instruction rounding mode.
7:3	xmfflags	Float-point arithmetic instruction accrued exception flags
2	xmsat	Fixed-point arithmetic instruction accrued saturation flag.
1:0	xmxrm	Fixed-point arithmetic instruction rounding mode.

3.5. Matrix Fixed-Point Rounding Mode, **xmxrm**

xmrn field indicates the rounding mode of fixed-point instructions. Suppose the pre-rounding result is v, and d bits of that result are to be rounded off. Then the rounded result is $(v \gg d) + r$, where r depends on the rounding mode as specified in the following table.

mrm	Mnemonic	meaning	Rounding Increment r
00	RNU	round-to-nearest-up (add +0.5 LSB)	v[d-1]

mrm	Mnemonic	meaning	Rounding Increment r
01	RNE	round-to-nearest-even	$v[d-1] \& (v[d-2:0] != 0 \mid v[d])$
10	RDN	round-down (truncate)	0
11	ROD	round-to-odd (OR bits into LSB)	$\neg v[d] \& v[d-1:0] != 0$

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
```

3.6. Matrix Fixed-Point Saturation Flag, xmxsat

The matrix fixed-point saturation flag **xmxsat** holds a single read-write least-significant bit(xmxsat[0]) that indicates if a fixed-point instruction has had to saturate an output value to fit into a destination format. The upper bits, xmxsat[XLEN-1:1], should be written as zeros. The matrix fixed-point saturation flag(xmxsat) is given a separate CSR address to allow independent access, but is also reflected as a field in xmcsr.

3.7. Matrix Float-Point Exception Flags, xmfflags

The matrix float-point exception flags(xmfflags) holds a five-bit read-write fields in the least-significant bits(xmfflags[4:0]). The upper bits, xmfflags[XLEN-1:5], should be written as zeros. The matrix float-point exception flags(xmfflags) is given a separate CSR address to allow independent access, but is also reflected as a field in xmcsr. The float matrix multiplication and float pointwise instructions update accrued exception fflags in xmfflags.

bits	Meaning
0	NX
1	UF
2	OF
3	reserved for future use
4	NV

3.8. Matrix Float-Point Rounding Mode, xmfrm

The matrix float-point rounding mode`xmfrm` holds a three-bit read-write fields in the least-significant bits(xmfrm[2:0]). The upper bits, xmfrm[XLEN-1:3], should be written as zeros. The matrix float-point rounding mode **xmfrm** is given a separate CSR address to allow independent access, but is also reflected as a field in xmcsr.

mfrm	Mnemonic	Meaning
000	RNE	Round to Nearest, ties to Even
001	RTZ	Round towards Zero
010	RDN	Round Down (towards $-\infty$)
011	RUP	Round Up (towards $+\infty$)
100	RMM	Round to Nearest, ties to Max Magnitude
101		Invalid
110		Invalid
111		Invalid

If **xmfrm** 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.

3.9. Matrix Saturation Mode Enable, xmsaten

The matrix saturation mode **xmsaten** holds a single read-write least-significant bit(xmsaten[0]) that defines the saturation mode when the operation result is fp8 or integer. The upper bits, xmfp8sat[XLEN-1:1] , should be written as zeros. The matrix saturation mode(xmsaten) is given a separate CSR address to allow independent access, but is also reflected as a field in xmcsr.

msaten	Meaning
0	non-saturation mode
1	saturation mode

Conversion from all other values first applies rounding to reduce the mantissa bit count to that of the destination FP8 format. After that, if the rounded magnitude is above the maximum destination magnitude:

```

if 'msaten' = 1:
    fp8 conversions: generate the max normal FP8 magnitude
    mmacc operations: the result should be saturated
    integer arithmetic operations: the result should be saturated
if 'msaten' = 0:
    fp8 conversion:E4M3 destination: generate a NaN
                E5M2 destination: generate an infinity
    mmacc operations: ignore the overflow and wrap around the result
    integer arithmetic operations: ignore the overflow and wrap around the result

```

3.10. Matrix Context Status in mstatus and sstatus

A 2-bit matrix context status field, MS, should be added to mstatus and shadowed in sstatus. It is defined analogously to the vector context status field, VS.

ms[1:0]	Meaning
00	All Off
01	Initial
10	Clean
11	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 4. Instruction Format

4.1. 32-bit instruction

The 32-bit instruction format fit under the custom-1(0101011) major opcode. There are three main types of instruction formats, as illustrated below.

31	28	27	26	25	23	22	20	19	18	17	15	14	12	11	10	9	7	6	0
func4	uop	imm3/ctrl		ms2	s_size	ms1		func3	d_size	md		major opcode							

Format for Matrix Instruction only with Matrix Registers

31	28	27	26	25	23	22	20	19	15	14	12	11	7	6	0			
func4	uop	ctrl		ms2		rs1		func3		rd		major opcode						

Format for Matrix Instruction with GPR

31	28	27	26	25	24	15	14	12	6	0
func4	uop	ctrl		imm10[9:0]		func3		0	major opcode	

Format for Matrix Instruction with IMM10

For the matrix instructions only with matrix register, md/ms1/ms2 fields indicate the matrix register indexes for the dest/src3/src1/src2 matrix operands, d_size field indicates the destination operand size, s_size field indicates the source operand size, imm3/ctrl field indicates the immediate index of matrix row/column or more operation information, and uop field indicates the different micro-operation types.

The md/ms1/ms2 fields are 3-bit width to index the four tile registers and four accumulation registers.

md/ms1/ms2[2:0]	Matrix Register Index
000	tr0
001	tr1
010	tr2
011	tr3
100	acc0
101	acc1
110	acc2

111	acc3
-----	------

For matrix multiplication instructions, ms1/ms2 should be 000-011 for tile registers, and md should be 100-111 for accumulation registers. For matrix element-wise instruction, the md/ms1/ms2 are all accumulation registers, which means md/ms1/ms2 should be 100-111. For matrix misc instructions, md/ms1/ms2 can be 000-111, according to the specific misc instruction.

The element sizes of d_size and s_size are as follows:

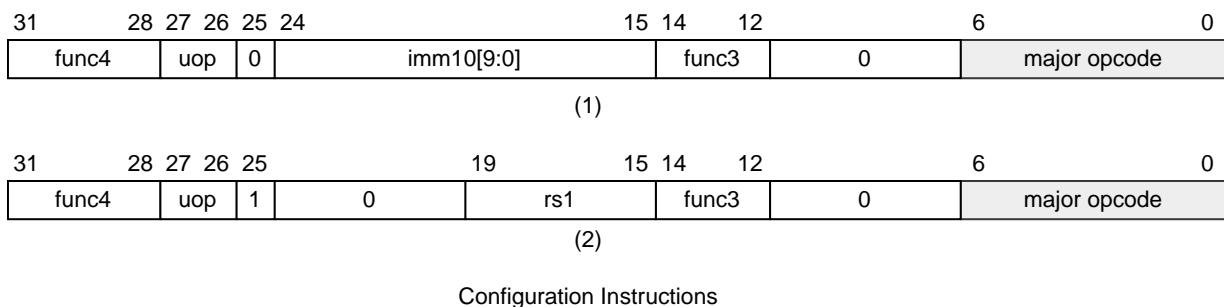
d/s_size[1:0]	Element Size
00	8-bit
01	16-bit
10	32-bit
11	64-bit

For the matrix instructions with GPR operand, rd/rs1/rs2 fields indicate the scalar register indexes for the dest/src1/src2 scalar operands.

For the matrix instructions with IMM10, imm10 fields indicates the 10-bit width immediate, usually used in configuration instruction.

4.1.1. Configuration Instructions

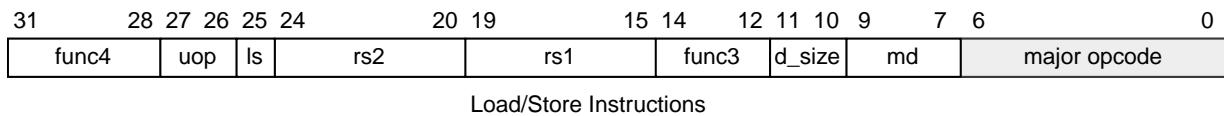
The following figure shows the encoding format for the Configuration instructions.



The func3 field of Configuration instructions is 000, and the uop field is 00. Except for the major opcode, func3, and uop fields, the rest fields of mrelease instruction are all zero. For mcfg{m/n/k}[i] instructions, the func4 field distinguishes m/n/k, and inst[25] indicates the configuration parameters is from imm10 fields or rs1 register.

4.1.2. Load/Store Instructions

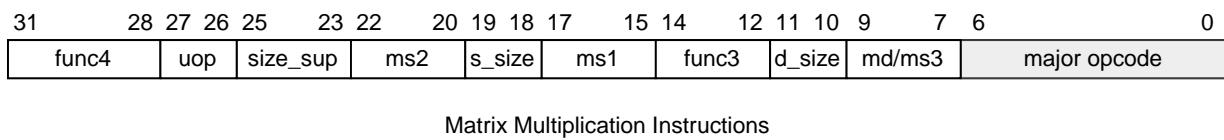
The following figure shows the encoding format for the Load/Store instructions.



The func3 field of Load/Store instructions is 000, and the uop field is 01. Inst[25] is the ls field, which means the load/store direction. The d_size field indicates the element size of Load/Store instructions. The func4 field defines load/store types, like A/B/C matrix type, transpose type and whole register load/store.

4.1.3. Matrix Multiplication Instructions

The following figure shows the encoding format for the Matrix Multiplication instructions.



The func3 field of Matrix Multiplication instructions is 000, and the uop field is 10. The A/B/C matrix come from ms1/ms2/ms3 fields. The d_size field indicates the element size of C matrix. Usually, the s_size indicates the element size of A/B matrix. When the func4 is 0000, the instruction type is float point matrix multiplication, including float point hybrid-precision instruction. When the func4 is 0001, the instruction type is integer matrix multiplication. When the func4 is 0010, the instruction type is integer hybrid-precision matrix multiplication.

The size_sup field indicates more operand type information. When there exist fp8 source type, inst[23]=0 means the fp8 type is E5M2, and inst[23]=1 means the fp8 type is E4M3. When there exist half-precision float point operand, inst[25]=0 means the operand type is fp16, and inst[25]=1 means the operand type is bf16. For float point operation, inst[24]=1 means the inst type is hybrid-precision matrix multiplication, inst[23] and inst[25] indicates different types of hybrid-precision matrix multiplication.

For integer matrix multiplication, inst[24:23] indicates source operands are signed/unsigned. When the s_size field is 00 and inst[25] is 1, the source type is 4-bit integer.

For integer hybrid-precision matrix multiplication, inst[24:23] also indicates source operands are signed/unsigned.

The specific meanings of inst[25] for different instructions are as follows:

func4	src_size	dst_size	inst[24]	inst[25]=0	inst[25]=1
0000	01	!01	0	half is fp16	half is bf16
0000	!01	01	0	half is fp16	half is bf16
0000	-	-	1	ms2 is int8	ms2 is int4

func4	src_size	dst_size	inst[24]	inst[25]=0	inst[25]=1
0001	00	-	-	src is int8	src is int4

The specific meanings of inst[24] for different instructions are as follows:

func4	inst[24]=0	inst[24]=1
0000	same-precision	hybrid-precision
0001	ms1 unsigned	ms1 signed
0010	ms1 unsigned	ms1 signed

The specific meanings of inst[23] for different instructions are as follows:

func4	src_size	dst_size	inst[24]	inst[23]=0	inst[23]=1
0000	00	-	0	fp8 is E5M2	fp8 is E4M3
0000	-	-	1	ms2 unsigned	ms2 signed
0001	-	-	-	ms2 unsigned	ms2 signed
0010	-	-	-	ms2 unsigne	ms2 signed

4.1.4. MISC Instructions

The following figure shows the encoding format for the MISC instructions.

31	28	27	26	25	23	22	20	19	18	17	15	14	12	11	10	9	7	6	0
func4	uop	imm3/func	ms2	s_size	ms1	func3	d_size	md											major opcode

(1)

31	28	27	26	25	24	23	22	20	19	15	14	12	11	7	6	0		
func4	uop	ctrl	e_size	ms2		rs1		func3		rd								major opcode

(2)

31	28	27	26	25	23	22	20	19	15	14	12	11	9	7	6	0	
func4	uop	ctrl	rs2		rs1		func3		md								major opcode

(3)

MISC Instructions

The func3 field of MISC instructions is 000, and the uop field is 11. The func4 field indicates different MISC instruction type.

Format type(1) shows the encoding of MISC instruction only using matrix registers. When the ms1 or ms2 are not used, these fields are set to zero. The imm3 field indicates the row index for row slide/broadcast instructions, the column index for column slide/broadcast, the matrix register index

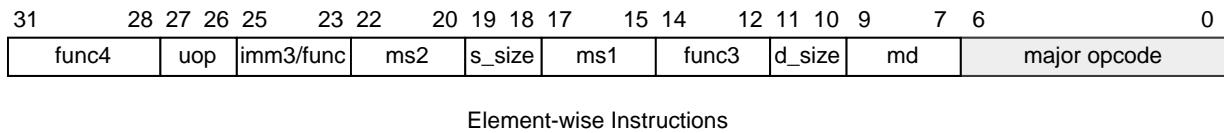
for the mzero instruction.

Format type(2) shows the encoding of matrix-to-scalar data-move instructions. The func4 field is 0010. The e_size field indicates the element size.

Format type(3) shows the encoding of scalar-to-matrix data-move instructions. The func4 field is 0011. When inst[25]=1, the instruction is mmov<b/h/w/d>.m.x. When inst[25]=0, the instruction is mdup<b/h/w/d>, and the rs1 fields ties to zero.

4.1.5. Element-wise Instructions

The following figure shows the encoding format for the Element-wise instructions.



The func3 field of Element-wise instructions is 001. The uop field of convert instructions is 00, the uop field of integer arithmetic instructions is 01, and the uop field of float point arithmetic instructions is 10. The func4 field indicates more different Element-wise instruction types.

For convert instructions between float point operands, the func4 field is 0000. Inst[25] indicates the operand is bf16 or fp16, when the float point is half precision. Inst[23] indicates the operand is E5M2 or E4M3, when the float point is fp8. Inst[24] indexes the low/high half columns for the double-widen or half-narrow convert instructions. Inst[24] indexes the lowest 1/4 columns or the second lowest 1/4 columns for fp32-to-fp8 convert instructions.

For convert instructions between float point operand and integer operand, the func4 field is 0001. Inst[24] indexes the low/high half columns for the double-widen or half-narrow convert instructions. When inst[25]=0, the convert type is integer to float point, and when inst[25]=0, the convert type is float point to integer. Inst[23] indicates the integer is signed or unsigned.

For mn4clip convert instructions, the func4 field indicates the part selection of columns, the signed or unsigned integer. When the inst[25:23] is 111, the two operands all come from matrix registers. When the inst[25:23] is not 111, the ms1 operand comes from a matrix register, and the ms2 comes from a row of a matrix register indexed by inst[25:23].

For int4-to-int8 convert instructions, the func4 field is 0110. Inst[24] indexes the low/high half columns for ms1. Inst[23] indicates the integer is signed or unsigned.

For integer or float point arithmetic instructions, the func4 fields indicates different operation types. When the inst[25:23] is 111, the two operands all come from matrix registers. When the inst[25:23] is not 111, the ms1 operand comes from a matrix register, and the ms2 comes from a row of a matrix register indexed by inst[25:23].

4.2. 64-bit instruction

The 64-bit instruction format fit under 0111111 major code, and the inst[38:32] is xxyyy11, where yyy not equal 111.

Other than the major code, inst[31:7] in 64-bit format is the same with the 32-bit instruction format.

4.2.1. Configuration Instructions

The following figure shows the encoding format for the Configuration instructions.

63 61 60				39 38		32
0			imm[31:10]			xxyyy11
31 28 27 26 25 24				15 14 12		6 0
func4	uop	0	imm[9:0]	func3		0111111
(1)						
63				39 38		32
			0			xxyyy11
31 28 27 26 25			19	15 14 12		6 0
func4	uop	1		rs1	func3	
(2)						
Configuration Instructions						

The fields in inst[31:7] is the same with the 32-bit instruction format. For the type(1) configuration instructions, the immediate width is 32bit, the imm[9:0] is in inst[24:15], and the imm[31:10] is in inst[60:39].

4.2.2. Load/Store Instructions

The following figure shows the encoding format for the Load/Store instructions.

63				39 38		32
			0			xxyyy11
31 28 27 26 25 24			20 19	15 14 12 11 10 9 7 6		0
func4	uop	ls	rs2	rs1	func3 d_size	md
Load/Store Instructions						

The fields in inst[31:7] is the same with the 32-bit instruction format. The inst[63:39] are not used, and is reserved for future extension.

4.2.3. Matrix Multiplication Instructions

The following figure shows the encoding format for the Matrix Multiplication instructions.

63		43 42 41	39 38	32					
	0		se	frm					
31	28 27 26 25	23 22	20 19 18 17	15 14	12 11 10	9	7 6	0	
func4	uop	size_sup	ms2	s_size	ms1	func3	d_size	md/ms3	0111111

Matrix Multiplication Instructions

The fields in inst[31:7] is the same with the 32-bit instruction format. The field frm in inst[41:39], indicates the float-point rounding mode. The field se in inst[42], indicates the integer saturation mode. The inst[63:43] are not used, and is reserved for future extension.

4.2.4. MISC Instructions

The following figure shows the encoding format for the MISC instructions.

63		39 38	32						
	0		xxyy11						
31	28 27 26 25	23 22	20 19 18 17	15 14	12 11 10	9	7 6	0	
func4	uop	imm3/func	ms2	s_size	ms1	func3	d_size	md	0111111

(1)

63		39 38	32					
	0		xxyy11					
31	28 27 26 25	24 23 22	20 19	15 14	12 11	7	6	0
func4	uop	ctrl	e_size	ms2	rs1	func3	rd	0111111

(2)

63		39 38	32					
	0		xxyy11					
31	28 27 26 25	23 22	20 19	15 14	12 11 10	9	7 6	0
func4	uop	ctrl	rs2	rs1	func3	d_size	md	0111111

(3)

MISC Instructions

The fields in inst[31:7] is the same with the 32-bit instruction format. The inst[63:39] are not used, and is reserved for future extension.

4.2.5. Element-wise Instructions

The following figure shows the encoding format for the Element-wise instructions.

63		46 45 44 43 42	39 38	32					
	0		xrm	se	frm	xxyy11			
31	28 27 26 25	23 22	20 19 18 17	15 14	12 11 10	9	7 6	0	
func4	uop	imm3/func	ms2	s_size	ms1	func3	d_size	md	0111111

Element-wise Instructions

The fields in inst[31:7] is the same with the 32-bit instruction format. The field frm in inst[41:39],

indicates the float-point rounding mode. The field se in inst[42], indicates the integer saturation mode for integer arithmetic instruction, and fp8 saturation mode for fp8 conversion instructions. The field xrm in inst[45:44], indicates the fixed-point rounding mode. The inst[63:46] are not used, and is reserved for future extension.

Chapter 5. Instructions

5.1. Configuration Instructions

5.1.1. Mtilem/n/k Configuration Instructions

Due to hardware resource constraints, one of the common ways to handle large-sized matrix multiplication is "tiling", where each iteration of the loop processes a subset of elements, and then continues to iterate until all elements are processed. The Matrix extension provides direct, portable support for this approach.

The block processing of matrix multiplication requires three levels of loops to iterate in the direction of m/k/n of the input application tiles.

For each iteration, most of the matrix multiplication can fully use the input tile registers. But when dealing with the tail of each input application tile's dimension, the length of each multiplication dimension like mtilem/mtiln/mtilek need to be updated to proper tail size.

A set of instructions is provided to allow rapid configuration of the values in mtile* to match application needs.

The msettile{m|k|n}[i] instructions set the mtilem/mtilek/mtilen CSRs based on their arguments.

```
msettilemi imm # imm = new mtilem
msettilem rs1 # rs1 = new mtilem
msettileki imm # imm = new mtilek
msettilek rs1 # rs1 = new mtilek
msettileni imm # imm = new mtilen
msettilen rs1 # rs1 = new mtilen
```

5.1.2. Mrelease Instruction

Mrelease instruction sets **MS** field in **mstatus** register to 01 (initial). This instruction is often used after the executing of a set of matrix instructions, indicating that the data in matrix registers can be discarded. Mrelease instruction can reduce context switch costs.

```
mrelease
```

5.2. Matrix Multiplication Instructions

Matrix multiplication operations take two matrix tiles, matrixA and matrixB, from matrix **tile registers** specified by **ms1** and **ms2**, and accumulate the multiplication result to matrixC specified by **md**, the output will overwrite the **md accumulation register**.

If the accumulation register is larger than the output matrix tile(matrixC) in bits, only the lower columns are updated and the higher parts are set by zeros.

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

category	instructions	Operand Type	Accumulator Type
Float	mfmacc.h	fp16	fp16
Float	mfmacc.s	fp32	fp32
Float	mfmacc.d	fp64	fp64
Float	mfmacc.h.<e4/e5>	fp8(e4m3/e5m2)	fp16
Float	mfmacc.bf16.<e4/e5>	fp8(e4m3/e5m2)	bf16
Float	mfmacc.s.h	fp16	fp32
Float	mfmacc.s.bf16	bf16	fp32
Float	mfmacc.d.s	fp32	fp64
Float	mfmacc.s.<e4/e5>	fp8(e4m3/e5m2)	fp32
Int	mmacc.w.b	int8	int32
Int	mmaccu.w.b	uint8	int32
Int	mmaccsu.w.b	(su)int8	int32
Int	mmaccus.w.b	(us)int8	int32

The mfmacc.x.y is float matrix multiplication operation, where x is destination operand width and y is source operand width. If x=y, y can be eliminated. The basic operation of float matrix multiplication is float dot-product, the float dot-product operations follow the IEEE-754/2008 standard. IEEE754/2008 standard's definition gives the implementation-specific flexibility of matrix-multiplication operation. The definitions are as followings:

1. Subnormal input/output: subnormal inputs and output are correctly computed and not treated as zero, tininess is detected after rounding.
2. The floating-point matrix-multiplication instructions which multiply single elements from each source operand and accumulate their product to each accumulator element. The reduction is done as an unordered-sum. Rounding are done after addition to accumulator.

The mmacc.x.y is int matrix multiplication operation, where x is destination operation width and y is source operation. If x=y, y can be eliminated. If **xmsaten** equals 1, the output should be saturated. Otherwise, the mmacc operations ignore the overflow and wrap around the result .



As integer matrix multiplication operation with non-widen or widen output is uncommon in AI scenarios, the matrix instruction set does not include such

instructions by default. The hardware can extend the mmacc.<b/h/w/d> or mmacc.<h/w/d>.<b/h/w> instructions as needed.

```
# integer matrix multiplication and add, md = md + ms1 * ms2.
mmacc.w.b    md, ms2, ms1    #signed 8bit, output quad-widen
mmaccu.w.b   md, ms2, ms1    #unsigned 8bit, output quad-widen
mmaccus.w.b  md, ms2, ms1    #unsigned-signed 8bit, output quad-widen
mmaccsu.w.b  md, ms2, ms1    #signed-unsigned 8bit, output quad-widen

# Float point matrix multiplication and add, md = md + ms1 * ms2.
mfmacc.h     md, ms2, ms1    # 16-bit float point
mfmacc.s     md, ms2, ms1    # 32-bit float point
mfmacc.d     md, ms2, ms1    # 64-bit float point
mfmacc.h.e4  md, ms2, ms1    # 8-bit float point, output double-widen
mfmacc.h.e5  md, ms2, ms1    # 8-bit float point, output double-widen
mfmacc.bf16.e4 md, ms2, ms1    # 8-bit float point, output double-widen
mfmacc.bf16.e5 md, ms2, ms1    # 8-bit float point, output double-widen
mfmacc.s.h   md, ms2, ms1    # 16-bit float point, output double-widen
mfmacc.s.bf16 md, ms2, ms1    # 16-bit float point, output double-widen
mfmacc.d.s   md, ms2, ms1    # 32-bit float point, output double-widen
mfmacc.s.e4  md, ms2, ms1    # 8-bit float point, output quad-widen
mfmacc.s.e5  md, ms2, ms1    # 8-bit float point, output quad-widen
```

Hardware can support a subset of these instructions above. Executing of unsupported instructions will raise illegal instruction exception. The supported instructions are shown in **misa** registers.

The field **mfrm** in **mcsr** register indicates the rounding mode of float-point matrix instructions.

For both integer and float point instructions, the output matrix register may not be filled. If the accumulation register is larger than the matrixC in bits, only the lower columns are updated. The elements out of bound of **mtilem** x **mtilen** are set to 0 (agnostic).

For executing matrix multiplication instructions, illegal instruction exception will occur under unsupported corresponding mtilem/mtilen/mtilek setting. The illegal condition for upper instructions is mtilem/mtilk/mtilen is bigger than TMMAX/TKMAX/TNMAX/.

The function description of **C = A x BT** is as follows:

```
#A[i,k] means the i row and j column element of ms1
#B[j,k] means the j row and k columns element of ms2

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

5.2.1. 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.

- mfmacc.h: fp16 floating-point, illegal if 'mmf16f16' of xmisa register is 0
- mfmacc.s: fp32 floating-point, illegal if 'mmf32f32' of xmisa register is 0
- **mfmacc.d: fp64 floating-point, illegal if 'mmf64f64' of xmisa register is 0**

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

For mfmacc.h, the legal matrix shape is:

- matrixA: $M \leq TLEN/TRLEN, K \leq TRLEN/16$
- matrixB: $N \leq TLEN/TRLEN, K \leq TRLEN/16$
- matrixC: $M \leq TLEN/TRLEN, N \leq TLEN/TRLEN$

For mfmacc.s, the legal matrix shape is:

- matrixA: $M \leq TLEN/TRLEN, K \leq TRLEN/32$
- matrixB: $N \leq TLEN/TRLEN, K \leq TRLEN/32$
- matrixC: $M \leq TLEN/TRLEN, N \leq TLEN/TRLEN$

For mfmacc.d, the legal matrix shape is:

- matrixA: $M \leq TLEN/TRLEN, K \leq TRLEN/64$
- matrixB: $N \leq TLEN/TRLEN, K \leq TRLEN/64$
- matrixC: $M \leq TLEN/TRLEN, N \leq TLEN/TRLEN$

Taking TLEN=512, TRLEN=128 as an example, the max matrix shapes of mfmacc.h, mfmacc.s and mfmacc.d are:

- mfmacc.h
 - matrixA: $M = 4, K = 8$
 - matrixB: $N = 4, K = 8$
 - matrixC: $M = 4, N = 4$
- fmmacc.s
 - matrixA: $M = 4, K = 4$

- matrixB: $N = 4, K = 4$
- matrixC: $M = 4, N = 4$
- mfmacc.d
 - matrixA: $M = 4, K = 2$
 - matrixB: $N = 4, K = 2$
 - matrixC: $M = 4, N = 4$

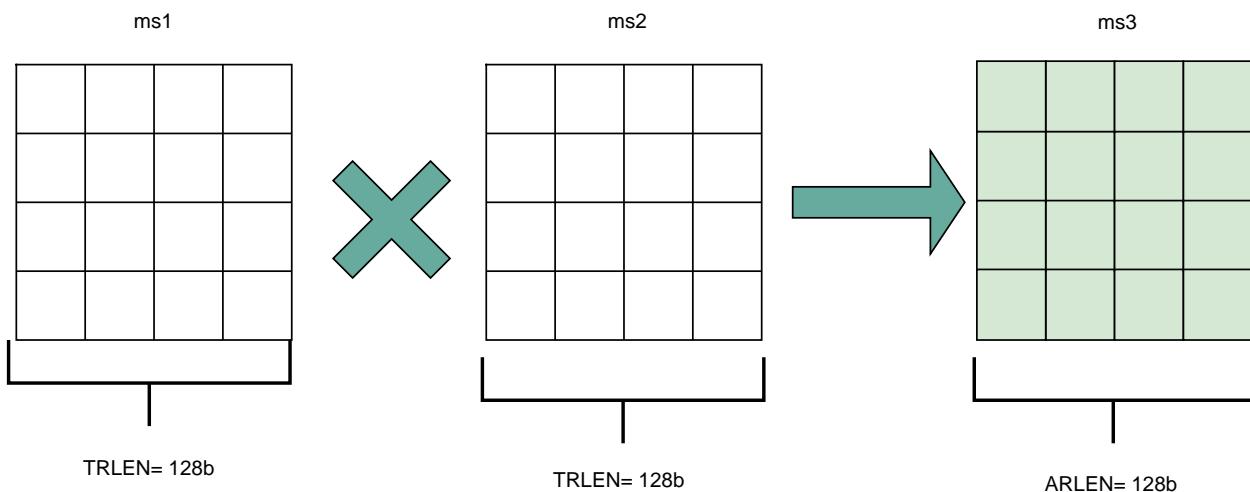
If ELEN is 32bits, the ALEN is 512 bits. The operation is shown below, where each cell in the figure represents a 64-bit, 32-bit or 16-bit element:

- mfmacc.d

When ELEN=32, mfmacc.d is reserved.

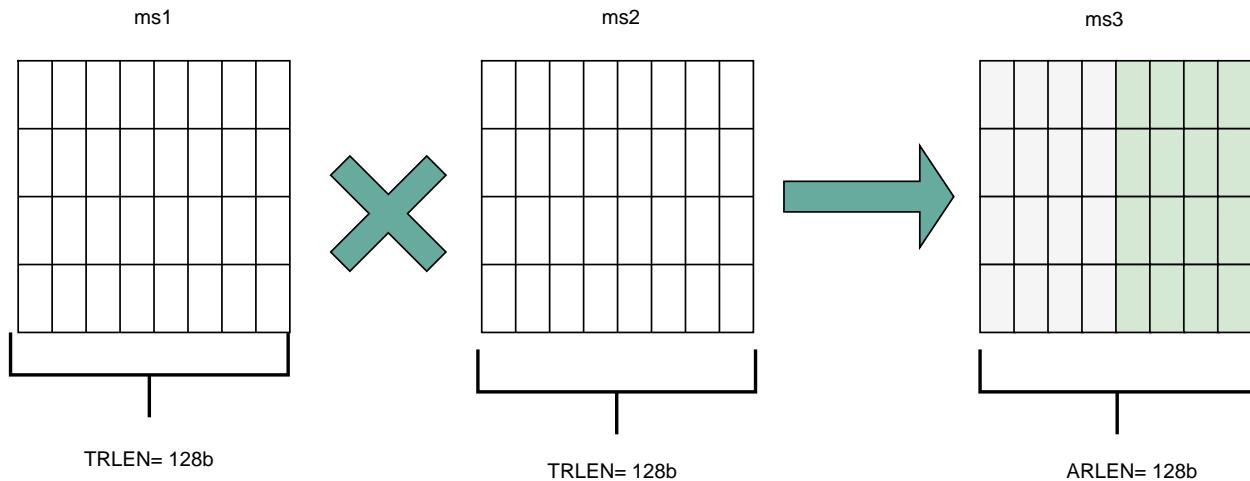
- mfmacc.s

When ELEN=32, mfmacc.s is as follow:



- mfmacc.h

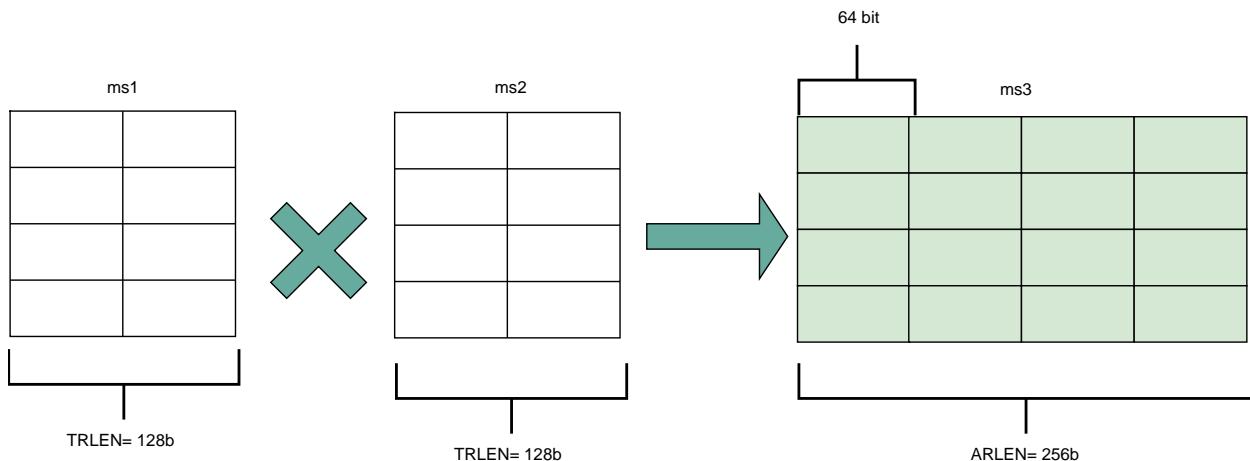
When ELEN=32, mfmacc.h is as follow:



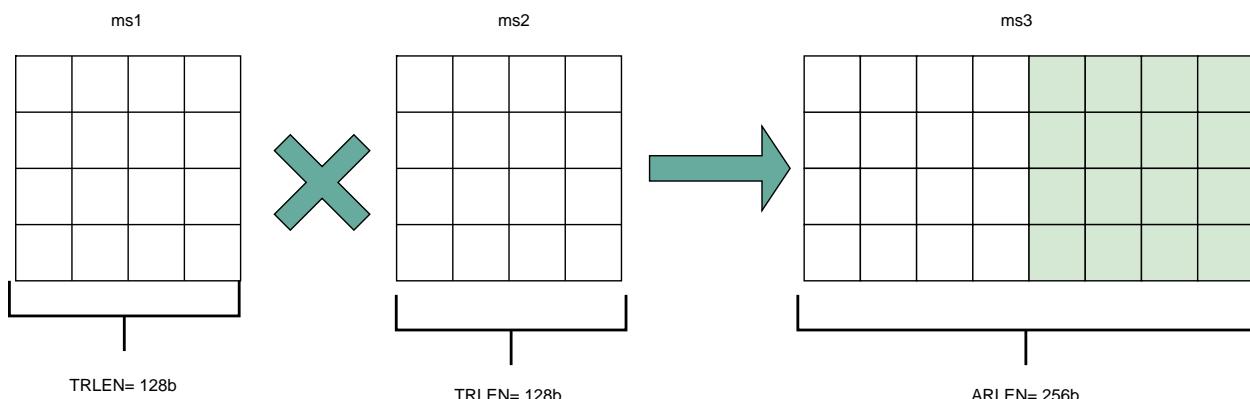
As ELEN is 32bit, only half accumulation register is used, the instruction writes back to the first half of md and the other parts of md are updated to 0.

If ELEN is 64bits, the ALEN is 1024 bits. The operation is shown below:

- mfmacc.d

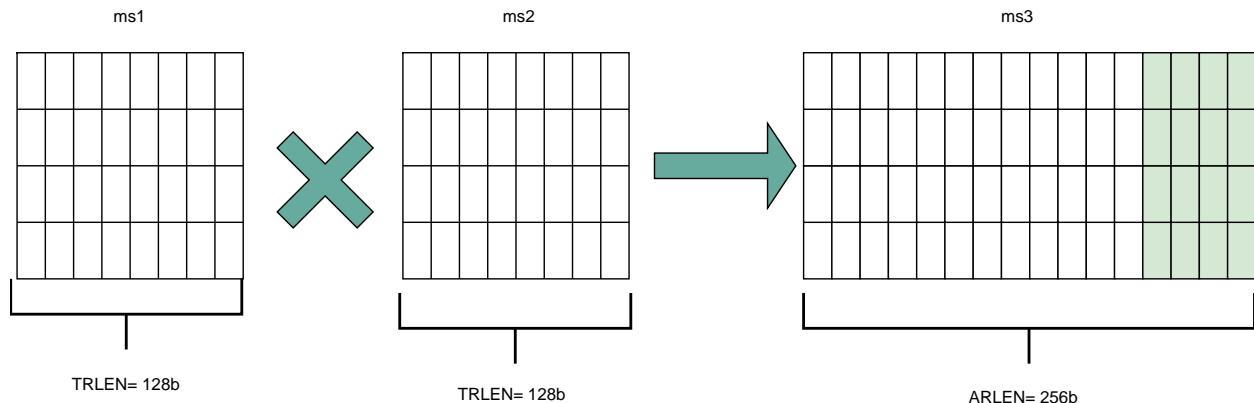


- mfmacc.s



As ELEN is 64-bit, only half accumulation register is used, the instruction writes back to the lowest half columns of md and the other parts of md are updated to 0.

- mfmacc.h



As ELEN is 64-bit, only 1/4 accumulation register are used. The instruction writes back to the lowest quarter columns of md and the other parts of md are updated to 0.

5.2.2. Float Matrix Multiplication(double-widen)

Double-widen float matrix multiplication indicates destination operand data width is twice that of the source operand. The data width of source operand is in instruction encoding.

- mfmacc.h.<e4/e5>: fp8(e4m3/e5m2) floating-point source and fp16 result ,illegal if 'mmf8f16' of xmisa register is 0.
- mfmacc.bf16.<e4/e5>: fp8(e4m3/e5m2) floating-point source and bf16 result ,illegal if 'mmf8bf16' of xmisa register is 0.
- mfmacc.s.h: fp16 floating-point source and fp32 result ,illegal if 'mmf16f32' of xmisa register is 0.
- mfmacc.s.bf16: bf16 floating-point source and fp32 result ,illegal if 'mmbf16f32' of xmisa register is 0.
- **mfmacc.d.s: fp32 floating-point source and fp64 result , illegal if 'mmf32f64' of xmisa register is 0.**

```
#float matrix multiplication, output double_widen, md = md + ms1*ms2
mfmacc.h.e4 md, ms2, ms1
mfmacc.h.e5 md, ms2, ms1
mfmacc.bf16.e4 md, ms2, ms1
mfmacc.bf16.e5 md, ms2, ms1
mfmacc.s.bf16 md, ms2, ms1
mfmacc.s.h md, ms2, ms1
mfmacc.d.s md, ms2, ms1
```

For `mfmacc.<h/bf16>.<e5/e4>`, 8-bit float widen matrix multiplication and add instruction. The legal matrix shape is:

- matrixA: $M \leq TLEN/TRLEN$, $K \leq TRLEN/8$
- matrixB: $N \leq TLEN/TRLEN$, $K \leq TRLEN/8$
- matrixC: $M \leq TLEN/TRLEN$, $N \leq TLEN/TRLEN$

For `mfmacc.s.h`, 16-bit float widen matrix multiplication and add instruction. The legal matrix shape is:

- matrixA: $M \leq TLEN/TRLEN$, $K \leq TRLEN/16$
- matrixB: $N \leq TLEN/TRLEN$, $K \leq TRLEN/16$
- matrixC: $M \leq TLEN/TRLEN$, $N \leq TLEN/TRLEN$

For `mfmacc.d.s`, 32-bit float widen matrix multiplication and add instruction, The legal matrix shape is:

- matrixA: $M \leq TLEN/TRLEN$, $K \leq TRLEN/32$
- matrixB: $N \leq TLEN/TRLEN$, $K \leq TRLEN/32$
- matrixC: $M \leq TLEN/TRLEN$, $N \leq TLEN/TRLEN$

Taking $TLEN=512$, $TRLEN=128$ as an example, the max matrix shapes are:

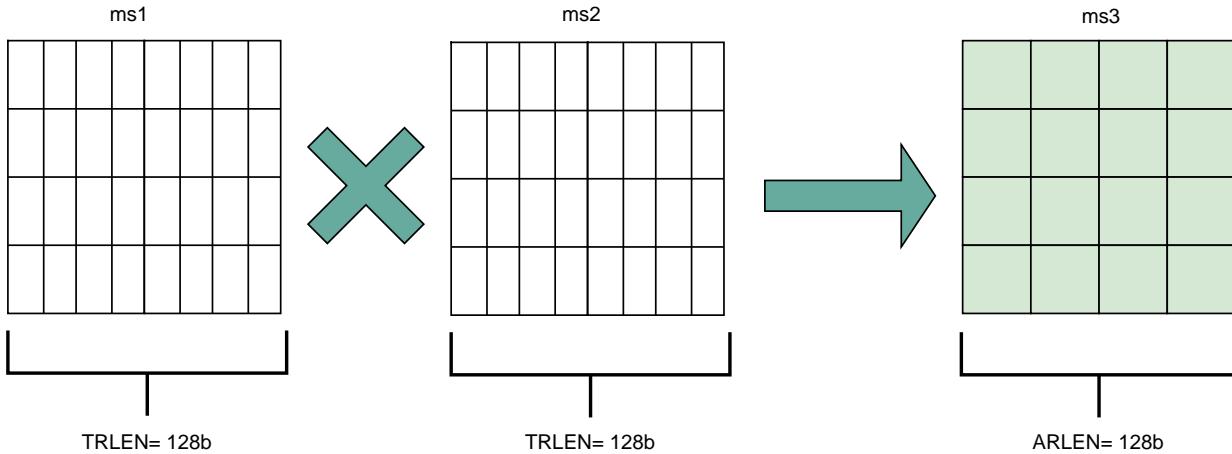
- `mfmacc.<h/bf16>.<e5/e4>`
 - matrixA: $M = 4$, $K = 16$
 - matrixB: $N = 4$, $K = 16$
 - matrixC: $M = 4$, $N = 4$
- `mfmacc.s.<h/bf16>`
 - matrixA: $M = 4$, $K = 8$
 - matrixB: $N = 4$, $K = 8$
 - matrixC: $M = 4$, $N = 4$
- `mfmacc.d.s`
 - matrixA: $M = 4$, $K = 4$
 - matrixB: $N = 4$, $K = 4$
 - matrixC: $M = 4$, $N = 4$

If ELEN is 32bits, the ALEN is 512 bits. The operation is shown below, where each cell in ms1/sm2 is the source operand width, and each cell in md is the destinations operand width:

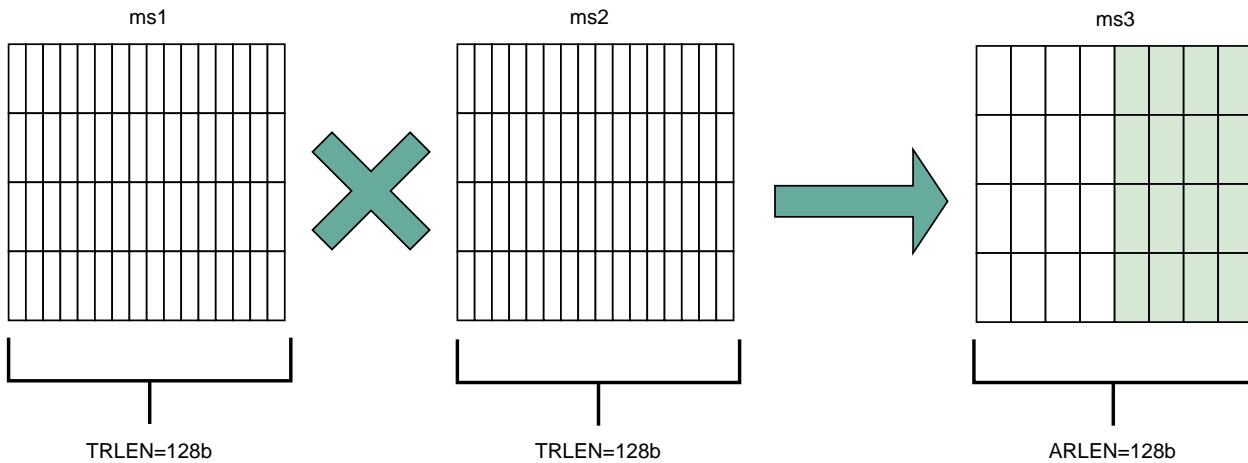
- `mfmacc.d.s`

When ELEN=32, mfmacc.d.s is reserved

- mfmacc.s.<h/bf16>



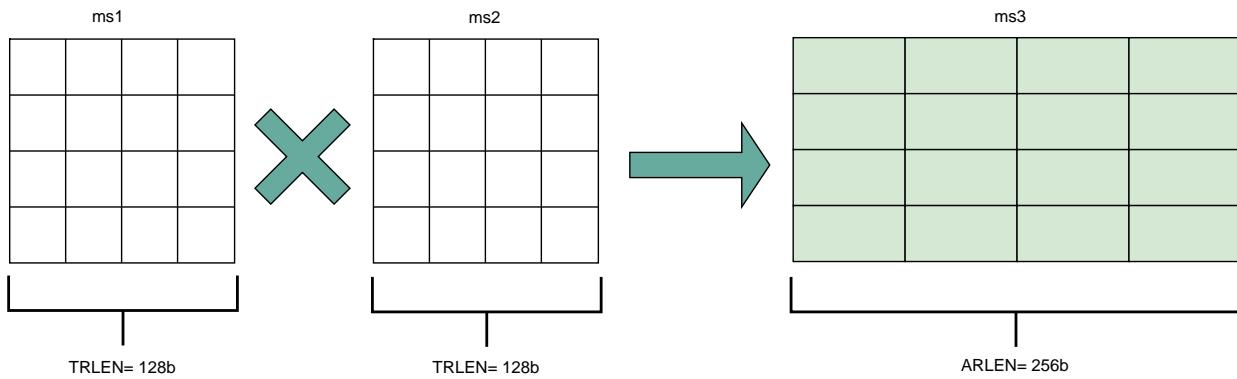
- mfmacc.<h/bf16>.<e5/e4>



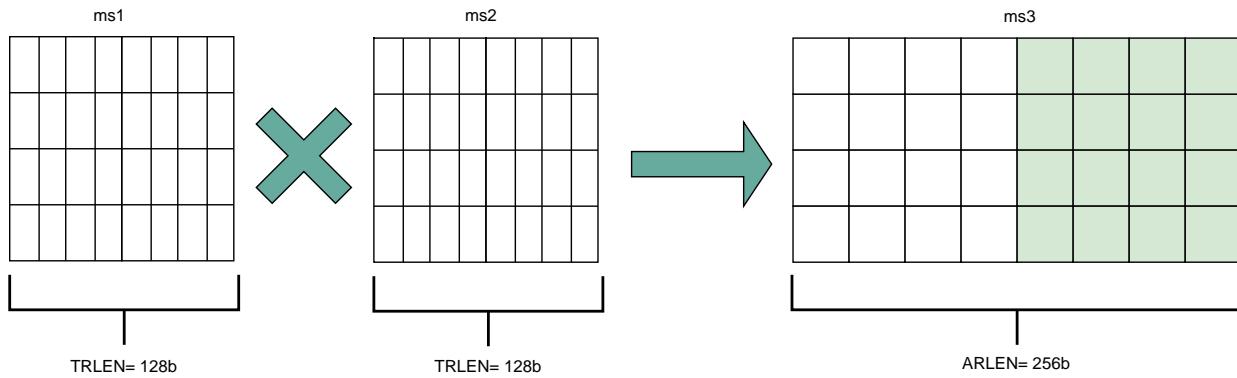
As ELEN is 32-bit, only half accumulation register is used, the instruction writes back to the lowest half columns of md and the other parts of md are updated to 0.

If ELEN is 64bits, the ALEN is 1024 bits. The operation is shown below, where each cell in ms1/sm2 is the source operand width, and each cell in md is the destinations operand width:

- mfmacc.d.s

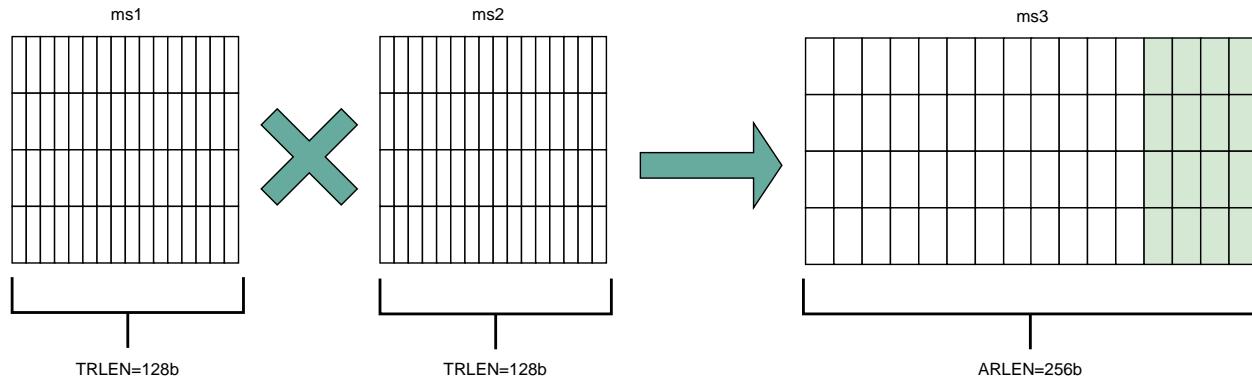


- mfmacc.s.<h/bf16>



As ELEN is 64-bit, only half accumulation register is used, the instruction writes back to the lowest half columns of md and the other parts of md are updated to 0.

- mfmacc.<h/bf16>.<e5/e4>



As ELEN is 64-bit, only quarter accumulation register is used, the instruction writes back to the lowest quarter columns of md and the other parts of md are updated to 0.

5.2.3. Float Matrix Multiplication(quad-widen)

Quad-widen float matrix multiplication indicates destination operand data width is quadruple that of the source operand. The data width of source operand is in instruction encoding.

- mfmacc.s.<e4/e5>: fp8(e4m3/e5m2) floating-point source and fp32 result ,illegal if 'mmf8f32' of xmisa register is 0.

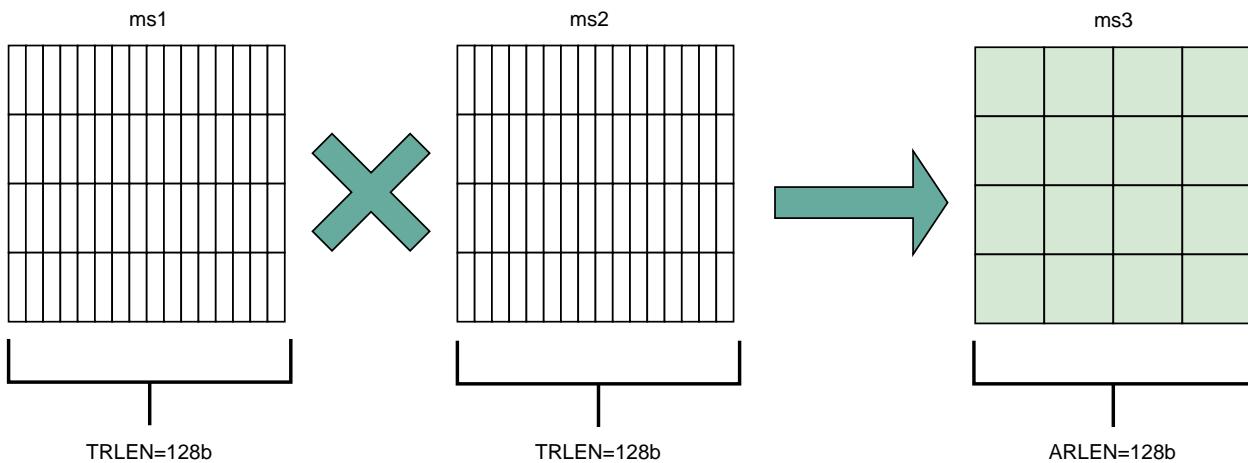
```
#float matrix multiplication, output quad_widen, md = md + ms1*ms2
mfmacc.s.e4 md, ms2, ms1
mfmacc.s.e5 md, ms2, ms1
```

For mfmacc.s.<e4/e5>, the legal matrix shape is:

- matrixA: $M \leq TLEN/TRLEN, K \leq TRLEN/8$
- matrixB: $N \leq TLEN/TRLEN, K \leq TRLEN/8$
- matrixC: $M \leq TLEN/TRLEN, N \leq TLEN/TRLEN$

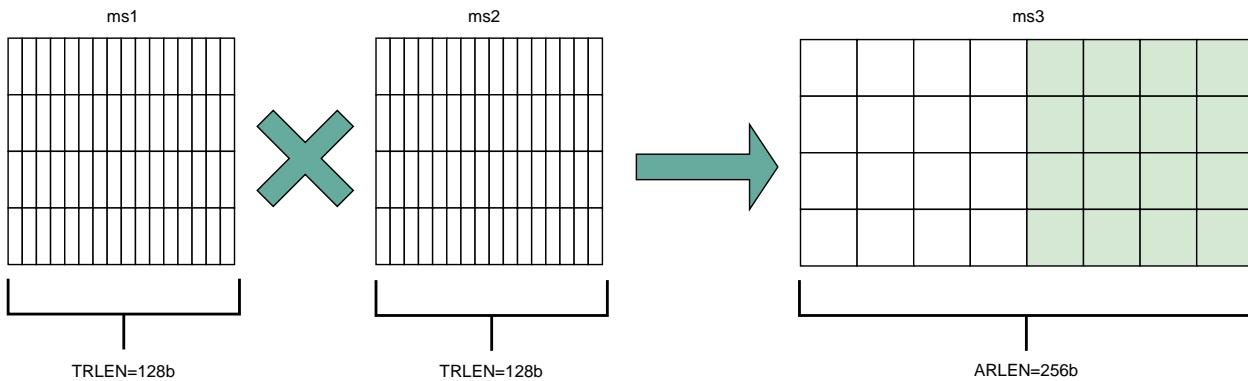
If ELEN is 32bits, the ALEN is 512 bits. The operation is shown below, where each cell in ms1/ms2 is 8-bit, and each cell in md is 32bit:

- mfmacc.s.<e4/e5>



If ELEN is 64bits, the ALEN is 1024 bits. The operation is shown below, where each cell in ms1/ms2 is 8-bit, and each cell in md is 32bit:

- mfmacc.s.<e4/e5>



As ELEN is 64-bit, only half accumulation register is used, the instruction writes back to the lowest half columns of md and the other parts of md are updated to 0.

5.2.4. Integer Matrix Multiplication

The integer matrix multiplication with destination data width is quad-widen that of the source data width. The source operand data width in instruction encoding supported are int8, 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. The overflow is processed depends on the msaten mode. If the msaten is 1, the overflow result should be saturated to the maximum number, and if the msaten is 0, the overflow result should be wrap around.

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

```
#8bit data width
mmaacc.w.b    md, ms2, ms1          #signed matrix multiply
mmaaccu.w.b   md, ms2, ms1          #unsigned matrix multiply
mmaaccus.w.b  md, ms2, ms1          #unsigned-signed matrix multiply
mmaaccsu.w.b  md, ms2, ms1          #signed-unsigned matrix multiply
```

For int8 quad-widen matrix-multiplication, the maximum matrix shape is:

- matrixA: $M \leq TLEN/TRLEN, K \leq TRLEN/8$
- matrixB: $N \leq TLEN/TRLEN, K \leq TRLEN/8$
- matrixC: $M \leq TLEN/TRLEN, N \leq TLEN/TRLEN$

Taking $TLEN=512$, $TRLEN=128$ as an example, the max matrix shapes are:

- mmaqa.b/mmaqau.b/mmaqaus.b/mmaqasu.b
 - matrixA: $M = 4, K = 16$
 - matrixB: $N = 4, K = 16$

- matrixC: M = 4, N = 4

5.2.5. Examples Summary

Summary for max Matrix size of matrix multiply and add instructions for typical TLEN and TRLEN, assuming ELEN is 32bit:

TLEN	TRLEN	ALEN	Instruction	Matrix A:MxK	Matrix B: KxN	Matrix C: MxN
512	128	512	mfmacc.h	4x8	8x4	4x4
512	128	512	mfmacc.s.h	4x8	8x4	4x4
512	128	512	mmacc.w.b	4x16	16x4	4x4
2048	256	2048	mfmacc.h	8x16	16x8	8x8
2048	256	2048	mfmacc.s.h	8x16	16x8	8x8
2048	256	2048	mmacc.w.b	8x32	32x8	8x8
8192	512	8192	mfmacc.h	16x32	32x16	16x16
8192	512	8192	mfmacc.s.h	16x32	32x16	16x16
8192	512	8192	mmacc.w.b	16x64	64x16	16x16

5.3. Load and Store Instructions

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

The element data width is in instruction encoding, including 8/16/32/64 bits, 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.

Due to the difference in matrix tile shapes of matrix A, B, and C, three types of load/store instructions are provided to separately implement memory access operations for matrix A(mla*/msa*), B(mlb*/msb*), and C(mlc*/msc*).

The shape of the tile that load to or store from matrix register is based on **C = A x BT** matrix multiplication mode, the shape of left matrix A that load to matrix register is **mtilem x mtilek**, the shape of right matrix B that load to matrix register is **mtilen x mtilek**, and the shape of output matrix C that load to matrix register is **mtilem x mtilen**.

5.3.1. Load Instructions

```
# md destination, rs1 base address, rs2 row byte stride
```

```

# For left matrix, A
mlae8 md, (rs1), rs2          # 8-bit left tile load
mlae16 md, (rs1), rs2          # 16-bit left tile load
mlae32 md, (rs1), rs2          # 32-bit left tile load
mlae64 md, (rs1), rs2          # 64-bit left tile load

# For right matrix, B
mlbe8 md, (rs1), rs2          # 8-bit right tile load
mlbe16 md, (rs1), rs2          # 16-bit right tile load
mlbe32 md, (rs1), rs2          # 32-bit right tile load
mlbe64 md, (rs1), rs2          # 64-bit right tile load

# For output matrix, C
mlce8 md, (rs1), rs2          # 8-bit output tile load
mlce16 md, (rs1), rs2          # 16-bit output tile load
mlce32 md, (rs1), rs2          # 32-bit output tile load
mlce64 md, (rs1), rs2          # 64-bit output tile load

```

5.3.2. Store Instructions

```

# ms3 store data, rs1 base address, rs2 row byte stride

# For left matrix, A
msae8 ms3, (rs1), rs2          # 8-bit left tile store
msae16 ms3, (rs1), rs2          # 16-bit left tile store
msae32 ms3, (rs1), rs2          # 32-bit left tile store
msae64 ms3, (rs1), rs2          # 64-bit left tile store

# For right matrix, B
msbe8 ms3, (rs1), rs2          # 8-bit right tile store
msbe16 ms3, (rs1), rs2          # 16-bit right tile store
msbe32 ms3, (rs1), rs2          # 32-bit right tile store
msbe64 ms3, (rs1), rs2          # 64-bit right tile store

# For output matrix, C
msce8 ms3, (rs1), rs2          # 8-bit output tile store
msce16 ms3, (rs1), rs2          # 16-bit output tile store
msce32 ms3, (rs1), rs2          # 32-bit output tile store
msce64 ms3, (rs1), rs2          # 64-bit output tile store

```

In addition, considering that matrices are stored in memory either in row-major or column-major order, load/store instructions provide both non-transposed and transposed instructions. For non-transposed instructions, the matrix tile in memory is row-major of A, BT, and C matrix. And for transposed instructions, the matrix tile in memory is column major.

5.3.3. Load Transposed Instructions

```
# md destination, rs1 base address, rs2 row byte stride

# For left matrix, A
mlate8 md, (rs1), rs2      # 8-bit left tile load
mlate16 md, (rs1), rs2     # 16-bit left tile load
mlate32 md, (rs1), rs2     # 32-bit left tile load
mlate64 md, (rs1), rs2     # 64-bit left tile load

# For right matrix, B
mlbte8 md, (rs1), rs2      # 8-bit right tile load
mlbte16 md, (rs1), rs2     # 16-bit right tile load
mlbte32 md, (rs1), rs2     # 32-bit right tile load
mlbte64 md, (rs1), rs2     # 64-bit right tile load

# For output matrix, C
mlcte8 md, (rs1), rs2      # 8-bit output tile load
mlcte16 md, (rs1), rs2     # 16-bit output tile load
mlcte32 md, (rs1), rs2     # 32-bit output tile load
mlcte64 md, (rs1), rs2     # 64-bit output tile load
```

5.3.4. Store Transposed Instructions

```
# ms3 store data, rs1 base address, rs2 row byte stride

# For left matrix, A
# tile size = mtilek * mtilem
msate8 ms3, (rs1), rs2      # 8-bit left tile store
msate16 ms3, (rs1), rs2     # 16-bit left tile store
msate32 ms3, (rs1), rs2     # 32-bit left tile store
msate64 ms3, (rs1), rs2     # 64-bit left tile store

# For right matrix, B
# tile size = mtilen * mtilek
msbte8 ms3, (rs1), rs2      # 8-bit right tile store
msbte16 ms3, (rs1), rs2     # 16-bit right tile store
msbte32 ms3, (rs1), rs2     # 32-bit right tile store
msbte64 ms3, (rs1), rs2     # 64-bit right tile store

# For output matrix, C
# tile size = mtilen * mtilem
mscte8 ms3, (rs1), rs2      # 8-bit output tile store
mscte16 ms3, (rs1), rs2     # 16-bit output tile store
mscte32 ms3, (rs1), rs2     # 32-bit output tile store
mscte64 ms3, (rs1), rs2     # 64-bit output tile store
```

Executing matrix load or store instructions will result in illegal instruction exception under unsupported corresponding mtilem/mtilek/mtilen settings.

For mla*/msa*/mlb*/msb*, the destination matrix register or the source matrix register is tile register, and the index of ms3/md must be 0-3. For mlc*/msc*, the destination matrix register or the source matrix register is accumulation register, and the index of ms3/md must be 4-7.

5.3.5. Whole Matrix Load & Store Instructions

Load a whole Tile/Accumulation matrix from memory without considering the tile size.

```
mlme<8/16/32/64>    md, (rs1), rs2      # <8/16/32/64>-bit whole tile matrix load
```

Store a whole Tile/Accumulation matrix register to memory without considering the tile size.

```
msme<8/16/32/64>    ms3, (rs1), rs2      # <8/16/32/64>-bit whole tile matrix store
```

The destination matrix register or the source matrix register for mlme<8/16/32/64> and msme<8/16/32/64> can be either tile register or accumulation register.



Whole matrix load and store instructions are usually used for context saving and restoring.

5.3.6. Examples for Load and Store Instructions

The legal matrix shapes of ml<a/b/c/at/bt/ct>e<8/16/32/64> instructions are:

- mlae<8/16/32/64>: matrixA, mtilem \leq TLEN/TRLEN, mtilek \leq TRLEN/<8/16/32/64>
- mlbe<8/16/32/64>: matrixB, mtilen \leq TLEN/TRLEN, mtilek \leq TRLEN/<8/16/32/64>
- mlbc<8/16/32/64>: matrixC, mtilem \leq TLEN/TRLEN, mtilen \leq ARLEN/<8/16/32/64>

The matrix tile loading to tile register is X rows and Y columns elements.

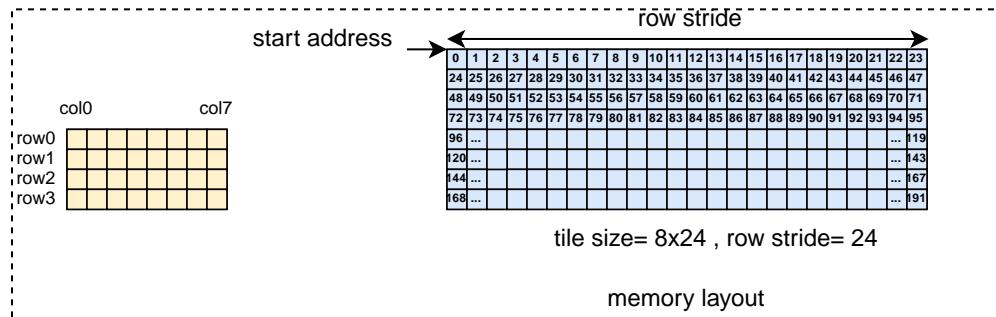
- ml<a/at>e<8/16/32/64>: X = mtilem, Y = mtilek
- ml<b/bt>e<8/16/32/64>: X = mtilen, Y = mtilek
- ml<c/ct>e<8/16/32/64>: X = mtilem, Y = mtilen

Taking TLEN=512, TRLEN=128 , ELEN=32 and ALEN=512 as an example, the max matrix shapes are:

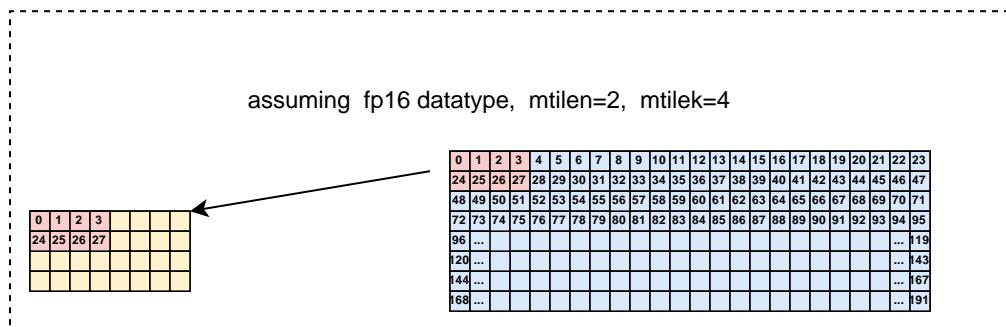
- ml<a/b/c/at/bt/ct>e8
 - matrixA: M = 4, K = 16

- matrixB: N = 4, K = 16
- matrixC: M = 4, N = 16
- ml<a/b/c/at/bt/ct>e16
 - matrixA: M = 4, K = 8
 - matrixB: N = 4, K = 8
 - matrixC: M = 4, N = 8
- ml<a/b/c/at/bt/ct>e32
 - matrixA: M = 4, K = 4
 - matrixB: N = 4, K = 4
 - matrixC: M = 4, N = 4
- ml<a/b/c/at/bt/ct>e64
 - matrixA: M = 4, K = 2
 - matrixB: N = 4, K = 2
 - matrixC: M = 4, N = 2

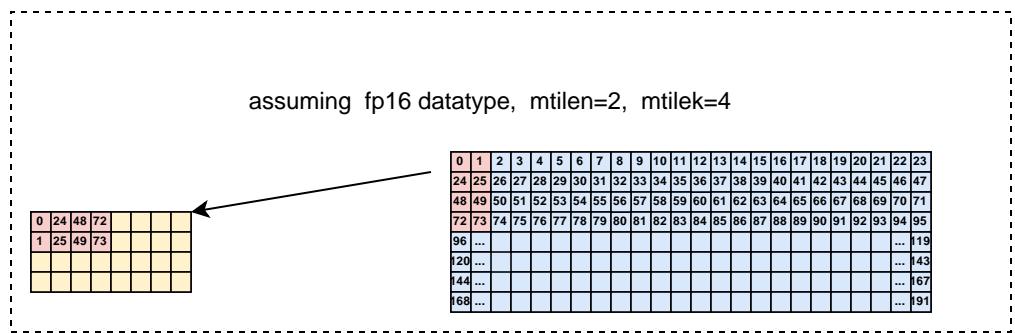
The memory layout and matrix register layout is shown below:



with mtile is 2 and mtilek is 4, the mlbe16 operation is shown below:



with mtile is 2 and mtilek is 4, the mlbte16 operation is shown below:



5.4. MISC Instructions

5.4.1. Mzero Instructions

Mzero instruction sets the destination register to zero. imm3 field is reused to specify the register number, encoding how many matrix registers to zero. md register index should be aligned with the register number.

imm3	register number
000	1
001	2
011	4
111	8
others	reserved

```
#zero 1 matrix register
mzero md
#zero 2 matrix registers
mzero2r md
#zero 4 matrix registers
mzero4r md
#zero 8 matrix registers
mzero8r md
```

5.4.2. Data Move Instructions

Matrix data move Instructions are provided to define the data movement between scalar registers and matrix registers. Matrix data move instructions ignore the matix size configuration.

Data Move Instructions between Matrix Registers

Data move instructions between matrix registers are used to move elements between two tile

registers, two accumulation registers, or one tile register and one accumulation register. The suffix of mmov.mm is ".mm", which means the source and destination matrix register can be either tile register or accumulation register.

When the source register and destination register are all tile registers or accumulation registers, there are no out-of-bound issue. When the source register and destination register are not the same type, the effective length in a row is minimum(TRLEN, ARLEN), the out-of-bound bits of md stay undisturbed.

```
#md = ms1, md and ms1 can be tile register or accumulation register
mmov.mm md, ms1
```

Data Move Instructions between Integer and Matrix

Data move instructions between matrix registers and integer registers are used to move elements between matrix registers and integer registers.

The mmov<b/h/w/d>.m.x instruction moves a scalar data to an element of the destination tile or accumulation 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. When the destination register is a tile register, the low log2(TLEN/EEW) bits are used. When the destination register is an accumulation register, the low log2(ALEN/EEW) bits are used.

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

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

The mmov<b/h/s/d>.x.m instruction moves a scalar data from a tile or accumulation 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. When the destination register is a tile register, the low log2(TLEN/EEW) bits are used. When the destination register is an accumulation register, the low log2(ALEN/EEW) bits are used.

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

```
#scalar-matrix move
mmov<b/h/w/d>.x.m rd, ms2, rs1
```

5.4.3. Data Broadcast Instructions

An element of a matrix register can be broadcasted to fill a whole matrix register. The element comes from the first row and the first column of the source matrix. The source register and destination register must be both tile registers or accumulation registers.

```
#matrix element broadcast, md[i,j] = ms1[0,0]
mbce8 md, ms1[imm3]
mbce16 md, ms1[imm3]
mbce32 md, ms1[imm3]
mbce64 md, ms1[imm3]
```

A row of a matrix register can be broadcasted to fill a whole matrix register. The mrbc.mv.i instruction moves and duplicates a vector to every row of the destination tile/accumulation register. The vector data is a row of tile/accumulation register, indexed by imm3. Only log2(ROWNUM) bits of imm3 are used. The source register and destination register must be both tile registers or accumulation registers.

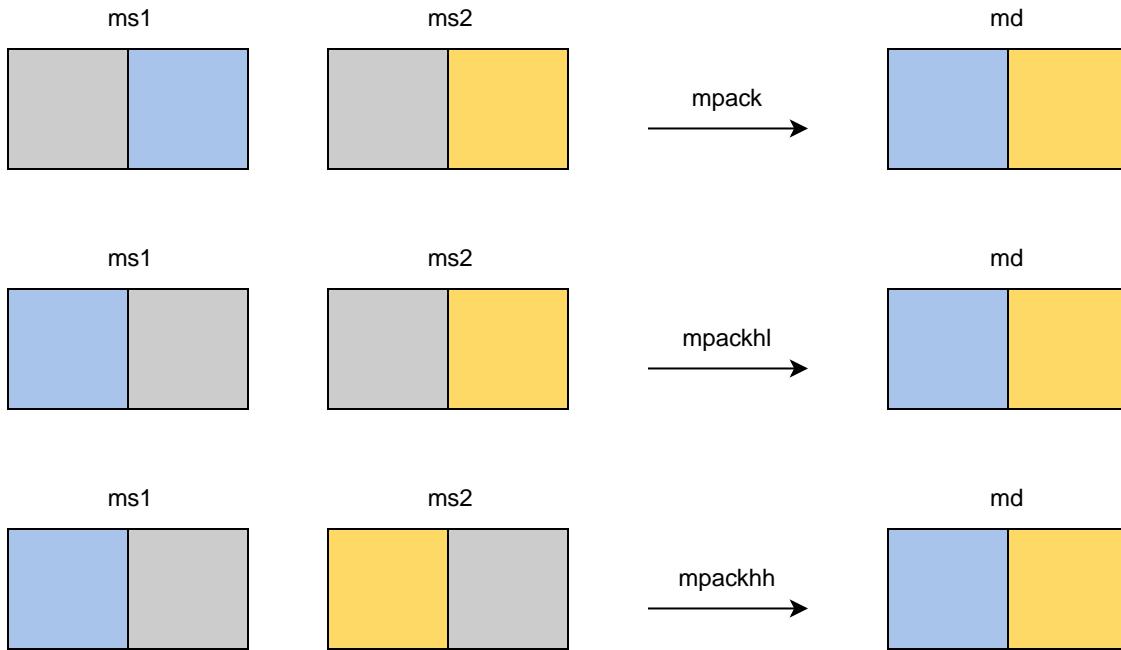
```
#matrix row broadcast, md[i,:]=ms1[uimm3,:]
mrbc.mv.i md, ms1[uimm3]
```

A column of a matrix register can be broadcasted to fill a whole matrix register. The columns are divided by EEW so that an TRLEN or ARLEN row contains TRLEN/EEW or ARLEN/EEW columns. Only log2(TRLEN/EEW) or log2(ARLEN/EEW) bits of uimm3 are used. The source register and destination register must be both tile registers or accumulation registers.

```
#matrix column broadcast, md[:,j]=ms1[:,imm3]
mcbce8.mv.i md, ms1[imm3]
mcbce16.mv.i md, ms1[imm3]
mcbce32.mv.i md, ms1[imm3]
mcbce64.mv.i md, ms1[imm3]
```

5.4.4. Matrix Pack Instructions

Matrix register pack instructions merge half column of ms1 and ms2. The source matrix registers and destination matrix registers must be all tile registers or accumulation registers. The ms1, ms2 and md must be the same register type.



```
#pack the low part of ms2 and low part of ms1 into md
mpack.mm md, ms2, ms1
```

```
#pack the high part of ms2 and low part of ms1 into md
mpackhl.mm md, ms2, ms1
```

```
#pack the high part of ms2 and high part of ms1 into md
mpackhh.mm md, ms2, ms1
```

5.4.5. Matrix Slide Instructions

Matrix register row slide down instructions move the row[i+imm3] of ms1 to the row[i] of md for each i from 0 to ROWNUM-imm3-1. The remaining rows of md are set to zero. Only log2(ROUNUM) bits of imm3 are used. The source register and destination register must be both tile registers or accumulation registers.

Matrix register row slide up instructions move the row[i-imm3] of ms1 to the row[i] of md for each i from imm3 to ROWNUM-1. The low imm3 rows of md are set to zero. Only log2(ROUNUM) bits of imm3 are used. The source register and destination register must be both tile registers or accumulation registers.

```
#matrix row slidedown, md[i,:]=ms1[i+imm3,:], when i > ROWNUM-imm3-1, md[i,:] is
zero
mrslidedowne<8/16/32/64> md, ms1, imm3
```

```
#matrix row slideup, md[i,:]=ms1[i-imm3,:], when i < imm3, md[i,:] is zero
mrslideupe<8/16/32/64> md, ms1, imm3
```

Matrix register column slide down instructions move the column[i+imm3] of ms1 to the column[i] of md for each i from 0 to TRLEN/EEW-imm3-1 or ARLEN/EEW-imm3-1, where the columns are divided by EEW so that a tile register row contains TRLEN/EEW columns or an accumulation register row contains ARLEN/EEW columns. The top imm3 columns of md are set to zero. Only log2(TRLEN/EEW) or log2(ARLEN/EEW) bits of imm3 are used. The src register and dst register must be both tile registers or accumulation registers.

Matrix register column slide up instructions move the column[i-imm3] of ms1 to the column[i] of md for each i from imm3 to TRLEN/EEW-1 or ARLEN/EEW-1, where the columns are divided by EEW so that a tile register row contains TRLEN/EEW columns or an accumulation register row contains ARLEN/EEW columns. The low imm3 columns of md are set to zero. Only log2(TRLEN/EEW) or log2(ARLEN/EEW) bits of imm3 are used. The src register and dst register must be both tile registers or accumulation registers.

```
#matrix column slidedown, md[:,i]=ms1[:,i+imm3], when i > ROWNUM-imm3-1, md[:,i] is zero
mcslidedowne<8/16/32/64>.<b/h/w/d> md, ms1, imm3

#matrix column slideup, md[:,i]=ms1[:,i-imm3], when i < imm3, md[:,i] is zero for
mcslideupe<8/16/32/64> md, ms1, imm3
```

5.5. Element-Wise Instructions

5.5.1. Integer Arithmetic Instructions

For integer matrix element-wise arithmetic instructions, matrix-matrix/matrix-vector instruction format are provided. 32-bit integer instructions are optionally supported. Both the source and destination registers of integer matrix element-wise instructions are accumulation registers.

- 32-bit width operand: legal if 'miew' and 'mmi8i32' of xmisa register are both set to 1

The matrix operand shape is configured by mtilem and mtlen. The legal matrix shapes for 32-bit width operand are:

- mtilem \leq ALEN/ARLEN
- mtlen \leq ARLEN/32

The elements out of bound of `mtilem x mtlen` are set to 0 (agnostic).

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 $md[i, j] = ms2[i, j] \text{ op } ms1[0, j]$.

madd performs the addition of src1 and src2. msb performs the subtraction of src2 from src1. mmul performs the multiplication of src1 and src2. If xmsaten is 0, overflows are ignored and the low MSEW bits of results are written to the destination md. If xmsaten is 1, overflows will generate the

saturated results. 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/unsigned compares, writing the max or min element to the destination respectively.

```
#matrix-matrix add
madd.w.mm md, ms2, ms1
#matrix-vector add
madd.w.mv.i md, ms2, ms1[imm3]

#matrix-matrix sub
msub.w.mm md, ms2, ms1
#matrix-vector sub
msub.w.mv.i md, ms2, ms1[imm3]

#matrix-matrix mul
mmul.w.mm md, ms2, ms1
#matrix-vector mul
mmul.w.mv.i md, ms2, ms1[imm3]

#matrix-matrix mulh
mmulh.w.mm md, ms2, ms1
#matrix-vector mulh
mmulh.w.mv.i md, ms2, ms1[imm3]

#matrix-matrix max
mmax.w.mm md, ms2, ms1
#matrix-vector max
mmax.w.mv.i md, ms2, ms1[imm3]

#matrix-matrix unsigned max
numax.w.mm md, ms2, ms1
#matrix-vector unsigned max
numax.w.mv.i md, ms2, ms1[imm3]

#matrix-matrix min
mmin.w.mm md, ms2, ms1
#matrix-vector min
mmin.w.mv.i md, ms2, ms1[imm3]

#matrix-matrix unsigned min
numin.w.mm md, ms2, ms1
#matrix-vector unsigned min
numin.w.mv.i md, ms2, ms1[imm3]
```

Matrix shift instructions including msll/msrl/msra. 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[imm3].

```
#matrix-matrix logical left shift
msll.w.mm md, ms2, ms1
#matrix-vector logical left shift
msll.w.mv.i md, ms2, ms1[imm3]

#matrix-matrix logic right shift
msrl.w.mm md, ms2, ms1
#matrix-vector logic right shift
msrl.w.mv.i md, ms2, ms1[imm3]

#matrix-matrix arithmetic right shift
msra.w.mm md, ms2, ms1
#matrix-vector arithmetic right shift
msra.w.mv.i md, ms2, ms1[imm3]
```

5.5.2. Float-point Arithmetic Instructions

For float matrix element-wise arithmetic instructions, matrix-matrix/matrix-vector instruction format are provided. 16-bit, 32-bit and 64-bit floating point element-wise operations are optionally supported. Both the source and destination registers of float matrix element-wise instructions are accumulation registers.

- fp16 operand: legal at follow cases
 - legal if 'mfew' and 'mmf16f16' of xmisa register are both set to 1
 - legal if 'mfew' and 'mmf8f16' of xmisa register are both set to 1
 - legal if 'mfew' and 'mmf16f32' of xmisa register are both set to 1
- fp32 operand: legal at follow cases
 - legal if 'mfew' and 'mmf32f32' of xmisa register are both set to 1
 - legal if 'mfew' and 'mmf16f32' of xmisa register are both set to 1
 - legal if 'mfew' and 'mmbf16f32' of xmisa register are both set to 1
 - legal if 'mfew' and 'mmf32f64' of xmisa register are both set to 1
 - legal if 'mfew' and 'mmf8f32' of xmisa register are both set to 1
- fp64 operand: legal at follow cases
 - legal if 'mfew' and 'mmf64f64' of xmisa register are both set to 1
 - legal if 'mfew' and 'mmf32f64' of xmisa register are both set to 1

The matrix operand shape is configured by mtilem and mtlen. The elements out of bound of **mtilem** x **mtlen** are set to 0 (agnostic).

The legal matrix shapes for 16-bit width operand are:

- mtilem \leq ALEN/ARLEN
- mtlen \leq ARLEN/16

The legal matrix shapes for 32-bit width operand are:

- mtilem \leq ALEN/ARLEN
- mtlen \leq ARLEN/32

The legal matrix shapes for 64-bit width operand are:

- mtilem \leq ALEN/ARLEN
- mtlen \leq ARLEN/64

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 element-wise instructions follow the IEEE-754/2008 standard. All floating-point element-wise instructions that perform rounding can select the rounding mode using the frm field.

```
#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]
```

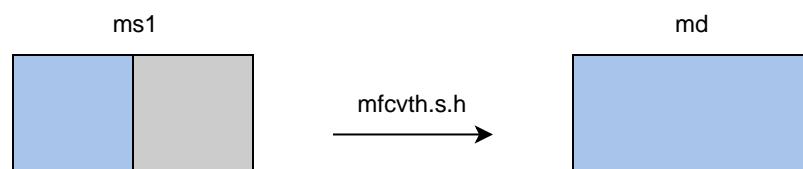
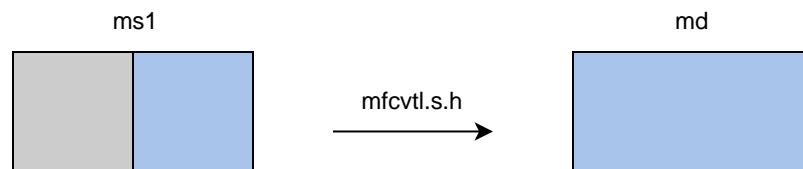
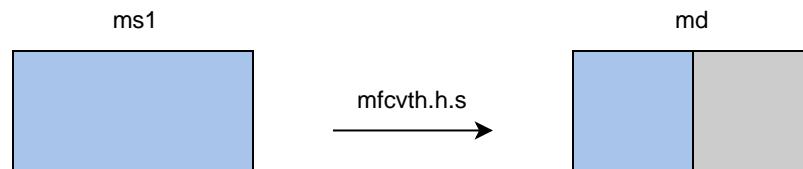
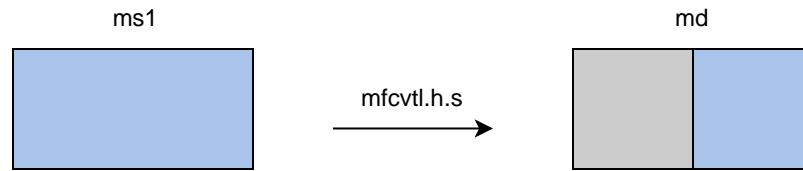
5.5.3. Type-Convert Instructions

The input and output matrices of type-convert instructions are both accumulation registers. And the matrix size configurations are ignored.

Float-point Conversion Instructions

The convert instruction converts each element from ms1 to destination register md. Narrow conversion is supported by specifying the source operand width and result operand width, source elements are twice as wide as destination elements. Low or high half columns of matrix register is used by destination matrix specified by md with another half columns kept undisturbed. Widen conversion is also supported by specifying the source operand width and result operand width, and destination elements are twice as wide as source elements. Low or high half columns of matrix register are used by source matrix specified by ms1.

- mfcvt<h/l>.h.<e4/e5>: legal if 'mfew' and 'mmf8f16' of xmisa register are both set to 1
- mfcvt<h/l>.<e4/e5>.h: legal if 'mfew' and 'mmf8f16' of xmisa register are both set to 1
- mfcvt<h/l>.h.s: legal if 'mfew' and 'mmf16f32' of xmisa register are both set to 1
- mfcvt<h/l>.s.h: legal if 'mfew' and 'mmf16f32' of xmisa register are both set to 1
- mfcvt<h/l>.bf16.s: legal if 'mfew' and 'mmbf16f32' of xmisa register are both set to 1
- mfcvt<h/l>.s.bf16: legal if 'mfew' and 'mmbf16f32' of xmisa register are both set to 1
- mfcvt<h/l>.s.d: legal if 'mfew' and 'mmf32f64' of xmisa register are both set to 1
- mfcvt<h/l>.d.s: legal if 'mfew' and 'mmf32f64' of xmisa register are both set to 1



```
#matrix-matrix floating point narrow convert(fp16tofp8, fp32tofp16, fp64tofp32)
#write back to the 2nd half of md
mfcvth.e4.h md, ms1
mfcvth.e5.h md, ms1
mfcvth.h.s md, ms1
mfcvth.bf16.s md, ms1
mfcvth.s.d md, ms1
#write back to the 1st half of md
mfcvth.e4.h md, ms1
mfcvth.e5.h md, ms1
mfcvth.h.s md, ms1
mfcvth.bf16.s md, ms1
mfcvth.s.d md, ms1

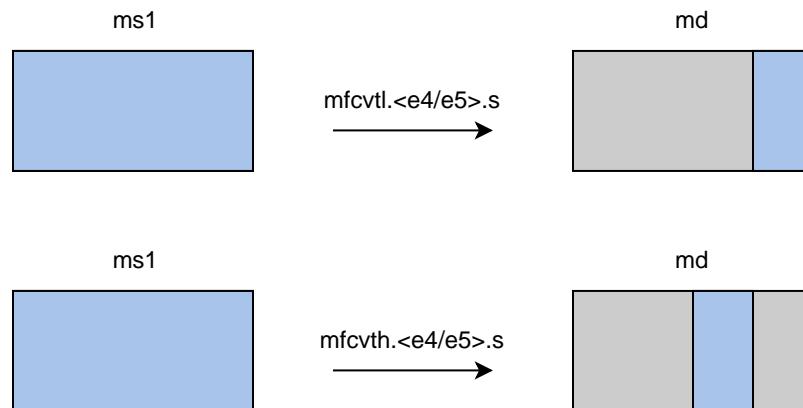
#matrix-matrix floating point widen convert(fp8tofp16, fp16tofp32, fp32tofp64)
#convert the 2nd half of ms1
mfcvth.h.e4 md, ms1
mfcvth.h.e5 md, ms1
mfcvth.s.h md, ms1
mfcvth.s.bf16 md, ms1
mfcvth.d.s md, ms1
#convert the 1st half of ms1
```

```
mfcvtl.h.e4 md, ms1
mfcvtl.h.e5 md, ms1
mfcvtl.s.h md, ms1
mfcvtl.s.bf16 md, ms1
mfcvtl.d.s md, ms1
```

For conversion from fp32 to fp8, legal if 'mfew' and 'mmf8f32' of xmisa register are both set to 1, illegal in all other cases.

The mfcvt<l/h>.<e5/e4>.s instructions are used to convert a single float-point source into a 4x narrower destination. The mfcvtl.<e5/e4>.s write back to the first quarter of md and the mfcvth.<e5/e4>.s write back to the second quarter of md, the rest parts of md are kept undisturbed.

```
#matrix-matrix floating point 4x narrow convert(fp32tofp8)
#convert the 2nd half of ms1
mfcvth.e5.s md, ms1
mfcvth.e4.s md, ms1
#convert the 1st half of ms1
mfcvtl.e5.s md, ms1
mfcvtl.e4.s md, ms1
```



Float-point Integer Instructions

Float integer conversion instructions are optionally supported.

- m<fs/fu/sf/uf>cvt.<s/w>.<w/s>: legal at follow cases
 - legal if 'mfic' and 'mmf32f32' of xmisa register are both set to 1
 - legal if 'mfic' and 'mmf16f32' of xmisa register are both set to 1
 - legal if 'mfic' and 'mmbf16f32' of xmisa register are both set to 1
 - legal if 'mfic' and 'mmf8f32' of xmisa register are both set to 1
- m<fs/fu/sf/uf>cvt<h/l>.<h/b>.<b/h>: legal at follow cases

- legal if 'mfic' and 'mmf16f16' of xmisa register are both set to 1
- legal if 'mfic' and 'mmf8f16' of xmisa register are both set to 1
- legal if 'mfic' and 'mmf16f32' of xmisa register are both set to 1

The conversion from integer to floating point supports non-widen conversion/double widen conversion. Integers support signed and unsigned integers. For double widen conversion, Low or high half columns of matrix register are used by source matrix specified by ms1.

```
#matrix-matrix unsigned integer floating point convert(uint32 to fp32)
mufcvt.s.w md, ms1
#matrix-matrix unsigned integer floating point widen convert (uint8 to fp16), the
low half of ms1
mufcvtl.h.b md, ms1
#matrix-matrix unsigned integer floating point widen convert (uint8 to fp16), the
high half of ms1
mufcvth.h.b md, ms1

#matrix-matrix signed integer floating point convert(sint32 to fp32)
msfcvt.s.w md, ms1
#matrix-matrix signed integer floating point widen convert(sint8 to fp16), the low
half
msfcvtl.h.b md, ms1
#matrix-matrix signed integer floating point widen convert(sint8 to fp16), the high
half
msfcvth.h.b md, ms1
```

The conversion from floating point to integer supports non-narrow conversion/half narrow conversion. Integers support signed and unsigned integers. For half narrow conversion, Low or high half columns of matrix register is used by destination matrix specified by md with another half columns kept undisturbed.

```
#matrix-matrix floating point unsigned integer convert(fp32 to uint32)
mfucvt.w.s md, ms1
#matrix-matrix floating point unsigned integer narrow convert(fp16 to uint8)
mfucvtl.b.h md, ms1
mfucvth.b.h md, ms1

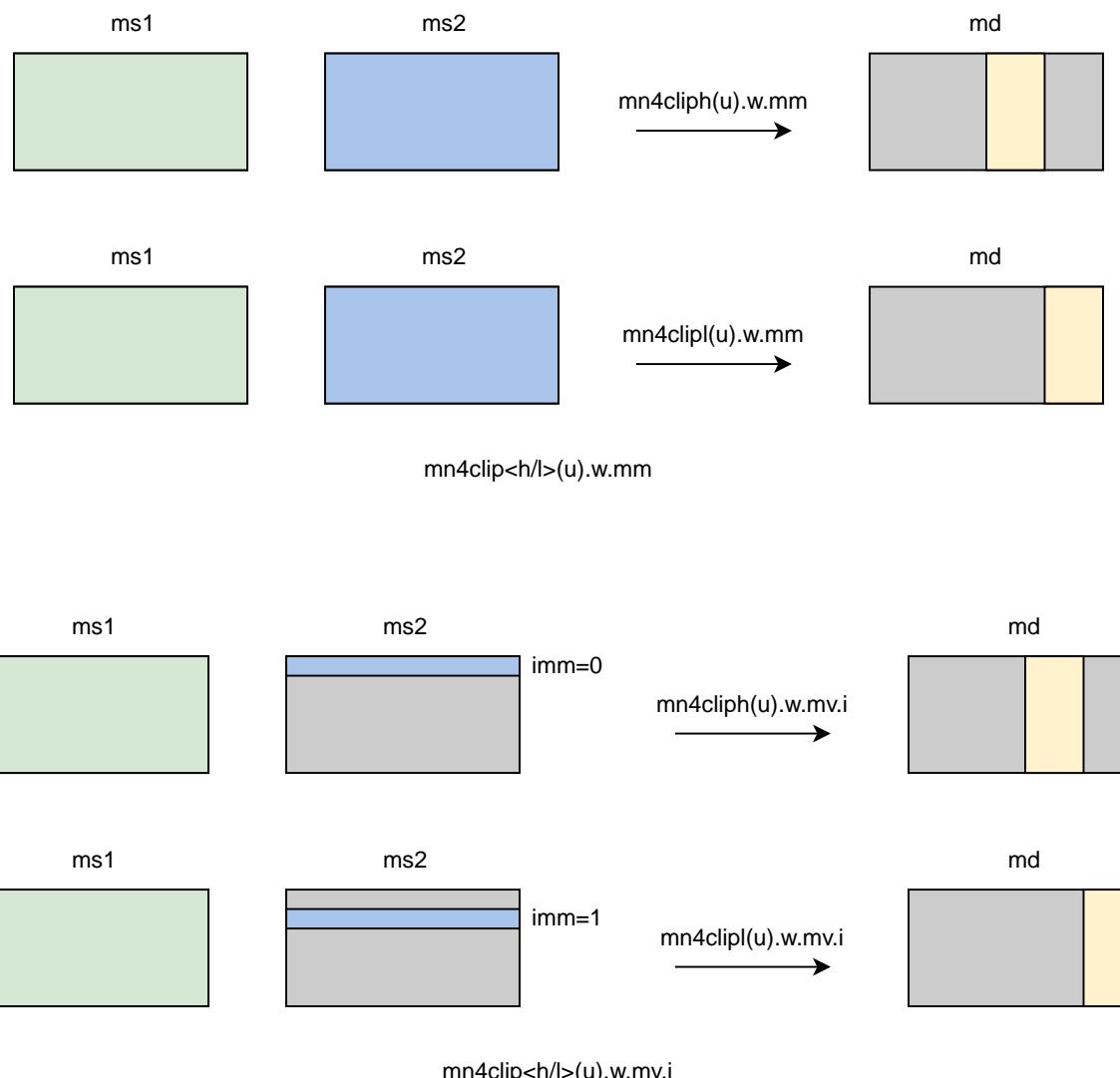
#matrix-matrix floating point signed integer convert(fp32 to sint32)
mfscvt.w.s md, ms1
#matrix-matrix floating point signed integer narrow convert(fp16 to sint8)
mfscvtl.b.h md, ms1
mfscvth.b.h md, ms1
```

Fixed-point Conversion Instructions

The mn4clip instructions belong to fixed-point conversion instructions, and support rounding with rounding mode specified in the xmxrm CSR. For clip instructions, rounding occurs before saturation.

- mn4clip<h/l>(u).w.mm: legal if 'miew' and 'mmi8i32' of xmisa register are both set to 1

mn4cliph/mn4cliphu/mn4clipl/mn4cliplu 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[imm3]). 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. mn4clipl/mn4clipul write back to the first quarter of md and the mn4cliph/mn4clipuh to the second quarter of md, the other parts of md are kept undisturbed.



```
#matrix-matrix signed clip
mn4cliph.w.mm md, ms2, ms1
mn4clipl.w.mm md, ms2, ms1
```

```
#matrix-vector signed clip,uimm3
mn4cliph.w.mv.i md, ms2, ms1[uimm3]
mn4clipl.w.mv.i md, ms2, ms1[uimm3]

#matrix-matrix unsigned clip
mn4cliphu.w.mm md, ms2, ms1
mn4cliplu.w.mm md, ms2, ms1
#matrix-vector unsigned clip,uimm3
mn4cliphu.w.mv.i md, ms2, ms1[uimm3]
mn4cliplu.w.mv.i md, ms2, ms1[uimm3]
```

Chapter 6. Matrix Extension

6.1. Zmint4: INT4 Extension

For int4 matrix multiplication instructions, 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, and the mtilek should always be an even value, otherwise reserved. The matrix multiplication instructions mmacc.w.q/mmaccu.w.q/mmaccus.w.q/mmaccsu.w.q are illegal if 'mmi4i32' of xmisa register is 0.

The legal matrix shape of int4 matrix multiplication instructions is:

- matrixA: M ≤ TLEN/TRLEN, K ≤ TRLEN/4 (mtilek must be even)
- matrixB: N ≤ TLEN/TRLEN, K ≤ TRLEN/4 (mtilek must be even)
- matrixC: M ≤ TLEN/TRLEN, N ≤ TLEN/TRLEN

The Zmint4 extension requires ELEN>=32.

```
#signed matrix multiply
mmacc.w.q md, ms2, ms1

#unsigned matrix multiply
mmaccu.w.q md, ms2, ms1

#unsigned-signed matrix multiply
mmaccus.w.q md, ms2, ms1

#signed-unsigned matrix multiply
mmaccsu.w.q md, ms2, ms1
```

Integer conversion instruction is optionally supported in Zmint4 extension. For double widen conversion, Low or high half columns of matrix register are used by source matrix specified by ms1. The matrix size configurations are ignored.

```
#matrix-matrix signed integer widen convert(sint4 to sint8)
msctl.w.b.q md, ms1
mscvth.w.b.q md, ms1

#matrix-matrix unsigned integer widen convert(uint4 to uint8)
mucctl.w.b.q md, ms1
mucvth.w.b.q md, ms1
```

For conversion between int8 and int4, legal if 'miew' and 'mmi4i32' of xmisa register are both set to 1, illegal in all other cases.

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.

Chapter 8. Instruction List

Table 2. Matrix Configuration Instructions

Format	31 28	27 26	25	24 20	19 15	14 12	11 7	6 0
	func	uop	ctrl	rs2	rs1	fun c3	nop	opcode
mrelease	0000	00	0	0		000	00000	major opcode
msettileki	0001	00	0	uimm10[9:0]		000	00000	major opcode
msettilek	0001	00	1	00000	rs1	000	00000	major opcode
msettilemi	0010	00	0	uimm10[9:0]		000	00000	major opcode
msettilem	0010	00	1	00000	rs1	000	00000	major opcode
msettileni	0011	00	0	uimm10[9:0]		000	00000	major opcode
msettilen	0011	00	1	00000	rs1	000	00000	major opcode

Table 3. Matrix MISC Instructions

Format	31 28	27 26	25	24 23	22 20	19 18	17 15	14 12	11 10	9 7	6 0
	func	uop	ctrl	ctrl	ms 2	s_si ze	ms 1	fun c3	d_s ize	md	opcode
mzero	0000	11	uimm3		000	00	000	000	00	md	major opcode
mmov.mm	0001	11	0	00	000	00	ms1	000	00	md	major opcode
mmovb.x.m	0010	11	0	00	ms2	rs1		000	rd		major opcode
mmovh.x.m	0010	11	0	01	ms2	rs1		000	rd		major opcode
mmovw.x.m	0010	11	0	10	ms2	rs1		000	rd		major opcode
mmovd.x.m	0010	11	0	11	ms2	rs1		000	rd		major opcode
mmovb.m.x	0011	11	1	rs2		rs1		000	00	md	major opcode
mmovh.m.x	0011	11	1	rs2		rs1		000	01	md	major opcode
mmovw.m.x	0011	11	1	rs2		rs1		000	10	md	major opcode
mmovd.m.x	0011	11	1	rs2		rs1		000	11	md	major opcode
mdupb.m.x	0011	11	0	rs2		00000		000	00	md	major opcode
mduph.m.x	0011	11	0	rs2		00000		000	01	md	major opcode

mdupw.m.x	0011	11	0	rs2		00000		000	10	md	major opcode
mdupd.m.x	0011	11	0	rs2		00000		000	11	md	major opcode
mpack	0100	11	0	00	ms2	00	ms1	000	00	md	major opcode
mpackhl	0100	11	0	10	ms2	00	ms1	000	00	md	major opcode
mpackhh	0100	11	0	11	ms2	00	ms1	000	00	md	major opcode
mrslidedbackward	0101	11	uimm3		000	00	ms1	000	00	md	major opcode
mrslideup	0110	11	uimm3		000	00	ms1	000	00	md	major opcode
mcslidedbackward.b	0111	11	uimm3		000	00	ms1	000	00	md	major opcode
mcslidedbackward.h	0111	11	uimm3		000	01	ms1	000	01	md	major opcode
mcslidedbackward.w	0111	11	uimm3		000	10	ms1	000	10	md	major opcode
mcslidedbackward.d	0111	11	uimm3		000	11	ms1	000	11	md	major opcode
mcslideup.b	1000	11	uimm3		000	00	ms1	000	00	md	major opcode
mcslideup.h	1000	11	uimm3		000	01	ms1	000	01	md	major opcode
mcslideup.w	1000	11	uimm3		000	10	ms1	000	10	md	major opcode
mcslideup.d	1000	11	uimm3		000	11	ms1	000	11	md	major opcode
mrbc.a.mv.i	1001	11	uimm3		000	00	ms1	000	00	md	major opcode
mcbc.b.mv.i	1010	11	uimm3		000	00	ms1	000	00	md	major opcode
mcbc.ah.mv.i	1010	11	uimm3		000	01	ms1	000	01	md	major opcode
mcbc.aw.mv.i	1010	11	uimm3		000	10	ms1	000	10	md	major opcode
mcbc.ad.mv.i	1010	11	uimm3		000	11	ms1	000	11	md	major opcode

Table 4. Matrix Multiplication Instructions

Format	31	28	27	25	22	19	17	14	11	9	7	6	0
	func	uop	size_su_p	ms2	s_size	ms1	func3	d_size	md/ms3	opcode			
mfmacc.h.e5	0000	01	000	ms2	00	ms1	000	01	md/ms3	major opcode			

mfmacc.h.e4	0000	01	001	ms2	00	ms1	000	01	md/ms3	major opcode
mfmacc.bf16. e5	0000	01	100	ms2	00	ms1	000	01	md/ms3	major opcode
mfmacc.bf16. e4	0000	01	101	ms2	00	ms1	000	01	md/ms3	major opcode
mfmacc.s.e5	0000	01	000	ms2	00	ms1	000	10	md/ms3	major opcode
mfmacc.s.e4	0000	01	001	ms2	00	ms1	000	10	md/ms3	major opcode
mfmacc.h	0000	01	000	ms2	01	ms1	000	01	md/ms3	major opcode
mfmacc.s.h	0000	01	000	ms2	01	ms1	000	10	md/ms3	major opcode
mfmacc.s.bf1 6	0000	01	001	ms2	01	ms1	000	10	md/ms3	major opcode
mfmacc.s.tf3 2	0000	01	001	ms2	10	ms1	000	10	md/ms3	major opcode
mfmacc.s	0000	01	000	ms2	10	ms1	000	10	md/ms3	major opcode
mfmacc.d.s	0000	01	000	ms2	10	ms1	000	11	md/ms3	major opcode
mfmacc.d	0000	01	000	ms2	11	ms1	000	11	md/ms3	major opcode
mmacc.w.b	0001	01	011	ms2	00	ms1	000	10	md/ms3	major opcode
mmaccu.w.b	0001	01	000	ms2	00	ms1	000	10	md/ms3	major opcode
mmaccus.w.b	0001	01	001	ms2	00	ms1	000	10	md/ms3	major opcode
mmaccsu.w.b	0001	01	010	ms2	00	ms1	000	10	md/ms3	major opcode
pmmacc.w.b	0001	01	111	ms2	00	ms1	000	10	md/ms3	major opcode
pmmaccu.w.b	0001	01	100	ms2	00	ms1	000	10	md/ms3	major opcode
pmmaaccus. w.b	0001	01	101	ms2	00	ms1	000	10	md/ms3	major opcode
pmmaccsu.w. b	0001	01	110	ms2	00	ms1	000	10	md/ms3	major opcode
mmacc.d.h	0001	01	011	ms2	01	ms1	000	11	md/ms3	major opcode
mmaccu.d.h	0001	01	000	ms2	01	ms1	000	11	md/ms3	major opcode
mmaccus.d.h	0001	01	001	ms2	01	ms1	000	11	md/ms3	major opcode
mmaccsu.d.h	0001	01	010	ms2	01	ms1	000	11	md/ms3	major opcode
mmacc.w.bp	0010	01	011	ms2	00	ms1	000	10	md/ms3	major opcode

mmaccu.w.bp	0010	01	000	ms2	00	ms1	000	10	md/ms3	major opcode
-------------	------	----	-----	-----	----	-----	-----	----	--------	--------------

Table 5. Matrix Load/Store Instructions

Format	31	28	27	25	24	19	14	11	9	7	6	0
	func	uop	ls	rs2	rs1	fun c3	d_si ze	md/ms3	opcode			
mlae8	0000	01	0	rs2	rs1	000	00	md	major opcode			
mlae16	0000	01	0	rs2	rs1	000	01	md	major opcode			
mlae32	0000	01	0	rs2	rs1	000	10	md	major opcode			
mlae64	0000	01	0	rs2	rs1	000	11	md	major opcode			
mlbe8	0001	01	0	rs2	rs1	000	00	md	major opcode			
mlbe16	0001	01	0	rs2	rs1	000	01	md	major opcode			
mlbe32	0001	01	0	rs2	rs1	000	10	md	major opcode			
mlbe64	0001	01	0	rs2	rs1	000	11	md	major opcode			
mlce8	0010	01	0	rs2	rs1	000	00	md	major opcode			
mlce16	0010	01	0	rs2	rs1	000	01	md	major opcode			
mlce32	0010	01	0	rs2	rs1	000	10	md	major opcode			
mlce64	0010	01	0	rs2	rs1	000	11	md	major opcode			
msae8	0000	01	1	rs2	rs1	000	00	ms3	major opcode			
msae16	0000	01	1	rs2	rs1	000	01	ms3	major opcode			
msae32	0000	01	1	rs2	rs1	000	10	ms3	major opcode			
msae64	0000	01	1	rs2	rs1	000	11	ms3	major opcode			
msbe8	0001	01	1	rs2	rs1	000	00	ms3	major opcode			
msbe16	0001	01	1	rs2	rs1	000	01	ms3	major opcode			
msbe32	0001	01	1	rs2	rs1	000	10	ms3	major opcode			
msbe64	0001	01	1	rs2	rs1	000	11	ms3	major opcode			
msce8	0010	01	1	rs2	rs1	000	00	ms3	major opcode			
msce16	0010	01	1	rs2	rs1	000	01	ms3	major opcode			
msce32	0010	01	1	rs2	rs1	000	10	ms3	major opcode			
msce64	0010	01	1	rs2	rs1	000	11	ms3	major opcode			

mlme8	0011	01	0	000 0	rs1	000	00	md	major opcode
mlme16	0011	01	0	000 0	rs1	000	01	md	major opcode
mlme32	0011	01	0	000 0	rs1	000	10	md	major opcode
mlme64	0011	01	0	000 0	rs1	000	11	md	major opcode
msme8	0011	01	1	000 0	rs1	000	00	ms3	major opcode
msme16	0011	01	1	000 0	rs1	000	01	ms3	major opcode
msme32	0011	01	1	000 0	rs1	000	10	ms3	major opcode
msme64	0011	01	1	000 0	rs1	000	11	ms3	major opcode
mlate8	0100	01	0	rs2	rs1	000	00	md	major opcode
mlate16	0100	01	0	rs2	rs1	000	01	md	major opcode
mlate32	0100	01	0	rs2	rs1	000	10	md	major opcode
mlate64	0100	01	0	rs2	rs1	000	11	md	major opcode
mlbte8	0101	01	0	rs2	rs1	000	00	md	major opcode
mlbte16	0101	01	0	rs2	rs1	000	01	md	major opcode
mlbte32	0101	01	0	rs2	rs1	000	10	md	major opcode
mlbte64	0101	01	0	rs2	rs1	000	11	md	major opcode
mlcte8	0110	01	0	rs2	rs1	000	00	md	major opcode
mlcte16	0110	01	0	rs2	rs1	000	01	md	major opcode
mlcte32	0110	01	0	rs2	rs1	000	10	md	major opcode
mlcte64	0110	01	0	rs2	rs1	000	11	md	major opcode
msate8	0100	01	1	rs2	rs1	000	00	ms3	major opcode
msate16	0100	01	1	rs2	rs1	000	01	ms3	major opcode
msate32	0100	01	1	rs2	rs1	000	10	ms3	major opcode
msate64	0100	01	1	rs2	rs1	000	11	ms3	major opcode

msbte8	0101	01	1	rs2	rs1	000	00	ms3	major opcode
msbte16	0101	01	1	rs2	rs1	000	01	ms3	major opcode
msbte32	0101	01	1	rs2	rs1	000	10	ms3	major opcode
msbte64	0101	01	1	rs2	rs1	000	11	ms3	major opcode
mscte8	0110	01	1	rs2	rs1	000	00	ms3	major opcode
mscte16	0110	01	1	rs2	rs1	000	01	ms3	major opcode
mscte32	0110	01	1	rs2	rs1	000	10	ms3	major opcode
mscte64	0110	01	1	rs2	rs1	000	11	ms3	major opcode

Table 6. Matrix Element-Wise Instructions

Format	31	28	27	25	22	19	17	14	11	9	7	6	0
	func	uop	ctrl	ms 2	s_si ze	ms 1	fun c3	d_s ize	md	opcode			
mfcvtl.h.e4	0000	00	000	000	00	ms1	001	01	md	major opcode			
mfcvth.h.e4	0000	00	010	000	00	ms1	001	01	md	major opcode			
mfcvtl.h.e5	0000	00	001	000	00	ms1	001	01	md	major opcode			
mfcvth.h.e5	0000	00	011	000	00	ms1	001	01	md	major opcode			
mfcvtl.e4.h	0000	00	000	000	01	ms1	001	00	md	major opcode			
mfcvth.e4.h	0000	00	010	000	01	ms1	001	00	md	major opcode			
mfcvtl.e5.h	0000	00	001	000	01	ms1	001	00	md	major opcode			
mfcvth.e5.h	0000	00	011	000	01	ms1	001	00	md	major opcode			
mfcvtl.s.h	0000	00	000	000	01	ms1	001	10	md	major opcode			
mfcvth.s.h	0000	00	010	000	01	ms1	001	10	md	major opcode			
mfcvtl.s.bf16	0000	00	001	000	01	ms1	001	10	md	major opcode			
mfcvth.s.bf16	0000	00	011	000	01	ms1	001	10	md	major opcode			
mfcvtl.e4.s	0000	00	000	000	10	ms1	001	00	md	major opcode			
mfcvth.e4.s	0000	00	010	000	10	ms1	001	00	md	major opcode			
mfcvtl.e5.s	0000	00	100	000	10	ms1	001	00	md	major opcode			
mfcvth.e5.s	0000	00	110	000	10	ms1	001	00	md	major opcode			
mfcvtl.h.s	0000	00	000	000	10	ms1	001	01	md	major opcode			

mfcvth.h.s	0000	00	010	000	10	ms1	001	01	md	major opcode
mfcvtl.bf16.s	0000	00	100	000	10	ms1	001	01	md	major opcode
mfcvth.bf16.s	0000	00	110	000	10	ms1	001	01	md	major opcode
mfcvt.tf32.s	0000	00	110	000	10	ms1	001	10	md	major opcode
mfcvt.s.tf32	0000	00	001	000	10	ms1	001	10	md	major opcode
mfcvtl.d.s	0000	00	000	000	10	ms1	001	11	md	major opcode
mfcvth.d.s	0000	00	010	000	10	ms1	001	11	md	major opcode
mfcvtl.s.d	0000	00	000	000	11	ms1	001	10	md	major opcode
mfcvth.s.d	0000	00	010	000	11	ms1	001	10	md	major opcode
msfcvtl.h.b	0001	00	001	000	00	ms1	001	01	md	major opcode
msfcvth.h.b	0001	00	011	000	00	ms1	001	01	md	major opcode
mufcvtl.h.b	0001	00	000	000	00	ms1	001	01	md	major opcode
mufcvth.h.b	0001	00	010	000	00	ms1	001	01	md	major opcode
msfcvt.s.w	0001	00	001	000	10	ms1	001	10	md	major opcode
mufcvt.s.w	0001	00	000	000	10	ms1	001	10	md	major opcode
mfscvt.w.s	0001	00	101	000	10	ms1	001	10	md	major opcode
mfucvt.w.s	0001	00	100	000	10	ms1	001	10	md	major opcode
mfucvtl.b.h	0001	00	100	000	01	ms1	001	00	md	major opcode
mfucvth.b.h	0001	00	110	000	01	ms1	001	00	md	major opcode
mfscvtl.b.h	0001	00	101	000	01	ms1	001	00	md	major opcode
mfscvth.b.h	0001	00	111	000	01	ms1	001	00	md	major opcode
mn4clipl.w.m m	0010	00	111	ms2	10	ms1	001	10	md	major opcode
mn4clipl.w.m v.i	0010	00	uim m3	ms2	10	ms1	001	10	md	major opcode
mn4cliph.w. mm	0011	00	111	ms2	10	ms1	001	10	md	major opcode
mn4cliph.w. mv.i	0011	00	uim m3	ms2	10	ms1	001	10	md	major opcode
mn4cliplu.w. mm	0100	00	111	ms2	10	ms1	001	10	md	major opcode

mn4cliplu.w. mv.i	0100	00	uim m3	ms2	10	ms1	001	10	md	major opcode
mn4cliphu.w. mm	0101	00	111	ms2	10	ms1	001	10	md	major opcode
mn4cliphu.w. mv.i	0101	00	uim m3	ms2	10	ms1	001	10	md	major opcode
mscvtl.b.p	0110	00	001	000	00	ms1	001	00	md	major opcode
mscvth.b.p	0110	00	011	000	00	ms1	001	00	md	major opcode
mucvtl.b.p	0110	00	000	000	00	ms1	001	00	md	major opcode
mucvth.b.p	0110	00	010	000	00	ms1	001	00	md	major opcode
madd.w.mm	0000	01	111	ms2	10	ms1	001	10	md	major opcode
madd.w.mv.i	0000	01	uim m3	ms2	10	ms1	001	10	md	major opcode
msub.w.mm	0001	01	111	ms2	10	ms1	001	10	md	major opcode
msub.w.mv.i	0001	01	uim m3	ms2	10	ms1	001	10	md	major opcode
mmul.w.mm	0010	01	111	ms2	10	ms1	001	10	md	major opcode
mmul.w.mv.i	0010	01	uim m3	ms2	10	ms1	001	10	md	major opcode
mmulh.w.mm	0011	01	111	ms2	10	ms1	001	10	md	major opcode
mmulh.w.mv. i	0011	01	uim m3	ms2	10	ms1	001	10	md	major opcode
mmax.w.mm	0100	01	111	ms2	10	ms1	001	10	md	major opcode
mmax.w.mv.i	0100	01	uim m3	ms2	10	ms1	001	10	md	major opcode
mumax.w.m m	0101	01	111	ms2	10	ms1	001	10	md	major opcode
mumax.w.mv .i	0101	01	uim m3	ms2	10	ms1	001	10	md	major opcode
mmin.w.mm	0110	01	111	ms2	10	ms1	001	10	md	major opcode
mmin.w.mv.i	0110	01	uim m3	ms2	10	ms1	001	10	md	major opcode
mumin.w.mm	0111	01	111	ms2	10	ms1	001	10	md	major opcode

mumin.w.mv.i	0111	01	uim m3	ms2	10	ms1	001	10	md	major opcode
msrl.w.mm	1000	01	111	ms2	10	ms1	001	10	md	major opcode
msrl.w.mv.i	1000	01	uim m3	ms2	10	ms1	001	10	md	major opcode
msll.w.mm	1001	01	111	ms2	10	ms1	001	10	md	major opcode
msll.w.mv.i	1001	01	uim m3	ms2	10	ms1	001	10	md	major opcode
msra.w.mm	1010	01	111	ms2	10	ms1	001	10	md	major opcode
msra.w.mv.i	1010	01	uim m3	ms2	10	ms1	001	10	md	major opcode
mfadd.h.mm	0000	10	111	ms2	01	ms1	001	01	md	major opcode
mfadd.h.mv.i	0000	10	uim m3	ms2	01	ms1	001	01	md	major opcode
mfadd.s.mm	0000	10	111	ms2	10	ms1	001	10	md	major opcode
mfadd.s.mv.i	0000	10	uim m3	ms2	10	ms1	001	10	md	major opcode
mfadd.d.mm	0000	10	111	ms2	11	ms1	001	11	md	major opcode
mfadd.d.mv.i	0000	10	uim m3	ms2	11	ms1	001	11	md	major opcode
mfsub.h.mm	0001	10	111	ms2	01	ms1	001	01	md	major opcode
mfsub.h.mv.i	0001	10	uim m3	ms2	01	ms1	001	01	md	major opcode
mfsub.s.mm	0001	10	111	ms2	10	ms1	001	10	md	major opcode
mfsub.s.mv.i	0001	10	uim m3	ms2	10	ms1	001	10	md	major opcode
mfsub.d.mm	0001	10	111	ms2	11	ms1	001	11	md	major opcode
mfsub.d.mv.i	0001	10	uim m3	ms2	11	ms1	001	11	md	major opcode
mfmul.h.mm	0010	10	111	ms2	01	ms1	001	01	md	major opcode
mfmul.h.mv.i	0010	10	uim m3	ms2	01	ms1	001	01	md	major opcode
mfmul.s.mm	0010	10	111	ms2	10	ms1	001	10	md	major opcode

mfmul.s.mv.i	0010	10	uim m3	ms2	10	ms1	001	10	md	major opcode
mfmul.d.mm	0010	10	111	ms2	11	ms1	001	11	md	major opcode
mfmul.d.mv.i	0010	10	uim m3	ms2	11	ms1	001	11	md	major opcode
mfmax.h.mm	0011	10	111	ms2	01	ms1	001	01	md	major opcode
mfmax.h.mv.i	0011	10	uim m3	ms2	01	ms1	001	01	md	major opcode
mfmax.s.mm	0011	10	111	ms2	10	ms1	001	10	md	major opcode
mfmax.s.mv.i	0011	10	uim m3	ms2	10	ms1	001	10	md	major opcode
mfmax.d.mm	0011	10	111	ms2	11	ms1	001	11	md	major opcode
mfmax.d.mv.i	0011	10	uim m3	ms2	11	ms1	001	11	md	major opcode
mfmin.s.mm	0100	10	111	ms2	01	ms1	001	01	md	major opcode
mfmin.s.mv.i	0100	10	uim m3	ms2	01	ms1	001	01	md	major opcode
mfmin.h.mm	0100	10	111	ms2	10	ms1	001	10	md	major opcode
mfmin.h.mv.i	0100	10	uim m3	ms2	10	ms1	001	10	md	major opcode
mfmin.d.mm	0100	10	111	ms2	11	ms1	001	11	md	major opcode
mfmin.d.mv.i	0100	10	uim m3	ms2	11	ms1	001	11	md	major opcode