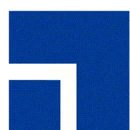
XBurst® Instruction Set Architecture

MIPS eXtended Architecture

Programming Manual

Release Date: June 2, 2017



北京君正集成电路股份有限公司
Ingenic Semiconductor Co.,Ltd.

XBurst® MIPS eXtended Architecture

Programming Manual

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

Disclaimer

This documentation is provided for use with Ingenic products. No license to Ingenic property rights is granted. Ingenic assumes no liability, provides no warranty either expressed or implied relating to the usage, or intellectual property right infringement except as provided for by Ingenic Terms and Conditions of Sale.

Ingenic products are not designed for and should not be used in any medical or life sustaining or supporting equipment.

All information in this document should be treated as preliminary. Ingenic may make changes to this document without notice. Anyone relying on this documentation should contact Ingenic for the current documentation and errata.

Ingenic Semiconductor Co., Ltd.

Ingenic Headquarters, East Bldg. 14, Courtyard #10
Xibeiwang East Road, Haidian District, Beijing, China,
Tel: 86-10-56345000
Fax:86-10-56345001
Http: //www.ingenic.com

CONTENTS

1 Overview of XBurst® MXA 3
2 Programming model 4
2.1 Data Formats 5
2.1.1 MXA Covert Interchage Control Value 5
2.1.2 XBurst 16-bit Floating-point data type 5
2.1.3 Floating-point complex data format 6
2.2 Data Register 6
2.3 Control register 6
2.3.1 MSA Implementation Register (MSAIR, register 0) 6
2.3.2 MSA Control and Status Register (MSACSR, register 31) 6
2.4 Exceptions 7
3 Instruction Set 9
3.1 Instruction Syntax and Encoding 10
3.1.1 Instruction Syntax 10
3.1.2 Instruction Encoding 10
3.2 Instruction Set Summary by Gategory 11
3.3 Alphabetical list of Instructions 14
ACCSL.df 15
ACCSR.df 16
ACCUL.df 17
ACCUR.df 18
ADDSL.df 19
ADDSR.df 20
ADDUL.df 21
ADDUR.df 22
BEXP.df 23
BEXT.df 24
BMR.V 25
BMU.V 26
FCMUL.df 27
FEXUPLC.df 29
FEXUPRC.df 30
LDINS.df 31
LDINSS.df 32
LDINSSU.df 33
LDINSSU2.df 34
LDINSU.df 35
MOVBP.B 36
MOVBT.B 37

MOVE.W 38
MULSL.df 39
MULSR.df 40
MULUL.df 41
MULUR.df 42
PCNT.V 43
SATSS.df 44
SATUS.df 45
SATUU.df 46
SLL.V 47
SLLI.V 48
SRL.V 49
SRLI.V 50
STEXT.df 51
STEXTU.df 52
SUBSL.df 53
SUBSR.df 54
SUBUL.df 55
SUBUR.df 56
VSHFR.B 57
VSLDI.B 58
Appendix A 59
A.1 COP2 Encoding of rs Field 59
A.2 COP2 Encoding of Function Field when rs = 3RFC2 60
A.3 COP2 Encoding of Function Field when rs = 2RFC2 61
A.4 COP2 Encoding of Function Field when rs = 3RC2 62
A.5 COP2 Encoding of Function Field when rs = 2RC2 63
A.6 COP2 Encoding of df/n field when rs = ELMC 64
A.7 COP2 Encoding of df/n field when rs = ELM2RC2 65
Revision History 66

1 Overview of XBurst® MXA

The XBurst® MIPS eXtended Architecture (MXA) module is as the enhanced part of the standard MIPS architecture that is programmer-friendly. The MXA is a software-programmable solution to handle heavy-duty speech, image processing.

The following complex floating-point operations deal with subnormal, NaN, Inf by the rule:

The normal operand is as the same as IEEE Standard for Floating-point Arithmetic 754-2008. The subnormal operand/result is saturated into plus/minus zero. The NaN or Inf result is saturated into plus/minus maximum normal number.

2 Programming model

This chapter describes MXA Programming model. This chapter includes the following sections:

- Data Formats
- Reigster File
- Control register
- Exceptions

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

2.1 Data Formats

The MXA instructions support the following data type:

- 2-bit, 4-bit signed integers, 8-bit, 16-bit, 32-bit, and 64-bit signed and unsigned integers
- 32-bit single precision floating point
- XBurst half-single precision floating point

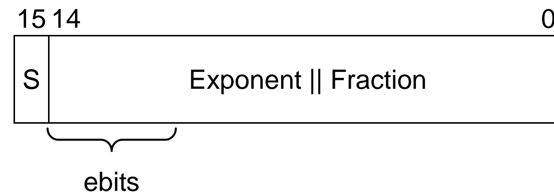
2.1.1 MXA Covert Interchage Control Value

Bits	Name	Description
10:4	bias	The bias of 16-bit floating-point. The valid value is from 0 to 2^{ebits}-1. However, if ebits' value is 0x8, the bias only can set to 0x7f.
3:0	ebits	The exponent bits of 16-bit floating-point. The valid value is from 2 to 8.

If the value of bias or ebits is invalid, the instructions FEXUPLC.W, FEXUPRC.W result value is **UNPREDICTABLE**.

2.1.2 XBurst 16-bit Floating-point data type

XHS(XBurst half single) is 16-bit floating-point data type.



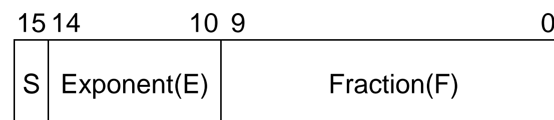
the XHS's fields are:

- 1-bit sign, s = $(-1)^S$
- Biased exponent, e= E - bias
- Binary fraction , f = 1 + 0.F

The XHS's value is s * 2^e * f.

if Exponent = 0 and Fraction = 0, the value is zero.

for example,



Above the field, ebits' value should set to 0x5, the parameter of XHS Floating-point data types:

Parameter	bias set to		
	0	15	31
Bits of mantissa precision, p	10		
Maximum exponent, emax	31	16	0
Minimum exponent, emin	0	-15	-31
Representation of fraction	1.F		

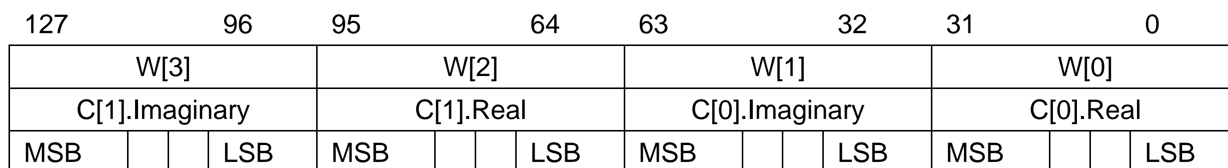
When the software down-covert single or double type to XHS, the source data values exceed the largest or smallest valid value of XHS and then the destination value is saturated into the largest or

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

smallest XHS' value.

2.1.3 Floating-point complex data format

The complex data type is compatible with GNU-gcc complex arithmetic. The vector register layout for elements of 32-bit floating-point complex data format as follow:



127 ... 96				95 ... 64				63 ... 32				31 ... 0			
W[3]				W[2]				W[1]				W[0]			
C[1].Imaginary				C[1].Real				C[0].Imaginary				C[0].Real			
MSB			LSB	MSB			LSB	MSB			LSB	MSB			LSB

A vector includes two 32-bit floating-point complex data. The layout is the same as the complex type of GNU-gcc.

2.2 Data Register

The MXA operates on 32 128-bit wide vector registers which shared with the MSA vector registers.

2.3 Control register

The control registers are used to record and manage the MXA state and resources. The MXA sharing the control register with the MSA:

MSAIR - MSA implementation and revision register

MSACSR - MSA control and status register

2.3.1 MSA Implementation Register (MSAIR, register 0)

The MSAIR Register is a 32-bit read-only register.

MSAIR

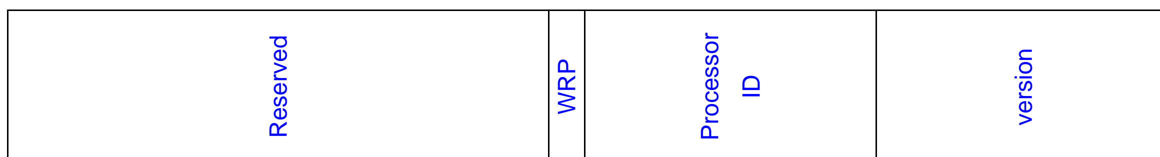


Table 2.1 MSA_MIR Register Field Description

Bits	Name	Description	R/W
31:17	Reserved	Writing has no effect, read as zero.	R
16	WRP	Vector Registers Partitioning.	
15:8	Processor ID	Processor ID number	R
7:0	Version	Version number.	R

2.3.2 MSA Control and Status Register (MSACSR, register 31)

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

MSACSR

Bit	31–25	24	23	22–21	20–19	18	17–12	11–7	6–2	1–0
	Reserved	FS	Reserved	Impl	Reserved	NX	Causes	Enables	Flags	RM

Table 2.2 MSA_MCSR Register Field Description

Bits	Name	Description	R/W
31:25	Reserved	Writing has no effect, read as zero.	R
24	FS	Flush to zero for denormalized number result. 0: do not flush to zero; 1: flush to zero.	RW
23	Reserved	Writing has no effect, read as zero.	R
22:21	Impl	Available to control implementation dependent features.	RW
20:19	Reserved	Reserved for future use; reads as zero and must be written as zero.	R
18	NX	Non-trapping floating point exception mode. In normal exception mode, the destination register is not written and the floating point exceptions set the Cause bits and trap. In non-trapping exception mode, the operations which would normally signal floating point exceptions do not write the Cause bits and do not trap. 0:Normal exception mode; 1:Non-trapping exception mode;	RW
17:12	Cause	Cause bits. These bits indicate the exception conditions arise during the execution of FP arithmetic instructions. Setting 1 if corresponding exception condition arises otherwise setting 0.	RW
11:7	Enables	Enable exception (in IEEE754, enable trap) bits. The exception shall occur when both enable bit and corresponding cause bit are set during the execution of an arithmetic FPU instruction.	RW
6:2	Flags	Flag bits. When an arithmetic FP operation arises an exception condition but does not trigger exception due to corresponding Enable bit is set value 0, then the corresponding bit in the Flag field is set value 1, while the others remains unchanged. The value of Flag field can last until software explicitly modifies it.	RW
1:0	RM	Round mode. 0: round to nearest. 1: round toward zero. 2: round toward $+\infty$. 3: round toward $-\infty$.	RW

2.4 Exceptions

MXA instruction can generate the following exceptions:

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

Reserved Instruction, if bit $Config3_{MSAP}$(CP0 Register 16, Select 3, bit 28) is not set, or if the usable FPU operates in 32-bit mode, i.e. bit $Status_{CU1}$(CP Register 12, Select 0, bit 29) is set and bit $Status_{FR}$ (CP Register 12,Select 0, bit 26) is not set. This exception uses the common exception vector with ExcCode field in Cause CP0 register set to 0x0a.

MXA Disabled, if bit $Config5_{MSAEn}$(CP0 Register 16, Select 5, bit 27) is not set or, when vector registers partitioning is enabled (i.e.$MSAIR_{WRP}$ set), if any MXA vector register accessed by the instruction is either not available or needs to be saved/restored due to a software context switch. This exception uses the common exception vector with ExcCode field in Cause CP0 register set to 0x15.

Address Error, an aligned load or aligned store to an address that is not naturally aligned for the data item causes an Address Error exception, the ExcCode field of CP0 Cause will set to 0x4(load) or 0x5(store).

Table 2.3 MXA Exception Code Values

Mnemonic	ExcCode		Description
	Decimal	Hexadecimal	
AdEL	4	0x04	Address error exception(load)
AdES	5	0x05	Address error exception(store)
RI	10	0x0a	Reserved Instruction exception
MXADis	21	0x15	MXA Disabled exception

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

3 Instruction Set

This chapter describes MXA Instruction. The MXA consists of integer, and floating-point instructions, all encoded in the MXA major opcode space.

Most MXA instructions operate vector element by vector element in a typical SIMD manner. Few instructions handle the operands as bit vectors because the elements don't make sense, e.g. for the bitwise logical operations.

For certain instructions, the source operand could be an immediate value or a specific vector element selected by an immediate index. The immediate or vector element is being used as a fixed operand across all destination vector elements.

The next sections list MXA instructions grouped by category and provide individual instruction descriptions arranged in alphabetical order. The constant WRLEN used in all instruction descriptions is set to 128, i.e. the vector register width in bits.

This chapter includes the following sections:

- Instruction Syntax and Encoding
- Instruction Set Summary by Gategory
- Alphabetical list of Instructions

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

3.1 Instruction Syntax and Encoding

3.1.1 Instruction Syntax

The MXA assembly language coding uses the following syntax elements:

- *func* : function/instruction name, e.g. ADDS_S or adds_s for signed saturated add
- *df* : destination data format, which could be a 2-bit(.2B), 4-bit(.4B), byte(.B), halfword(.H), word(.W), doubleword(.D), or the vector itself(.V)
- *wd* : destination vector registers, e.g. *$w0, …, $w31*
- *ws,wt,wr* : source, and target vector registers, e.g. *$w0, …, $w31*
- *ws[n]* : vector register element of index *n*, where *n* is a valid index value for elements of data format *df*
- m : immediate value valid as a bit index for the data format *df*

MXA instructions have two or three register, immediate, or element operands.

3.1.1.1 Vector Element Selection

MXA instructions of the form *func.df wd,ws[n]* select the nth element in the vector register ws based on the data format *df*. The valid element index values for various data formats and vector register sizes are shown in Table 3.1. The vector element is being used as a fixed operand across all destination vector elements.

Table 3.1 Valid Element Index Values

Mnemonic	Element Index
2-bit	n=0,.....,63
4-bit	n=0,.....,31
byte	n=0,.....,15
halfword	n=0,.....,7
word	n=0,.....,3
doubleword	n=0,1

3.1.2 Instruction Encoding

3.1.2.1 Data Format and Index Encoding

Most of the MXA instructions operate on byte, halfword, word or doubleword data formats. Internally, the data format *df* is coded by a 2-bit field as shown in Table 3.2.

MXA instructions using a specific vector element code both data format and element index in a 6-bit field df/n as shown in Table 3.3. All invalid index values or data formats will generate a Reserved Instruction Exception. For example, a vector register has 16 byte elements while the byte data format can code up to 32 byte elements. Selecting any vector byte element other than 0, 1, …, 15 generates a Reserved Instruction Exception.

The combinations marked Vector (".V") are used for coding certain instructions with data formats other

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

than byte, halfword, word, or doubleword.

If an instruction specifies a bit position, the data format and bit index *df/m* are coded as shown in Table 3.4.

Table 3.2 Two-bit Data Format Field Encoding

df	Bit 0	
Bit 1	0	1
0	Byte	Halfword
1	Word	Doubleword

Table 3.3 Data Format and Element Index Field Encoding

df/n	Bits 5...0			
	00nnnn	100nnn	1100nn	11100n
	Byte	Halfword	Word	Doubleword

Table 3.4 Data Format and Bit Index Field Encoding

df/m	Bits 6...0			
	0mmmmmm	10mmmmm	110mmmm	1110mmm
	Doubleword	Word	Halfword	Byte

3.1.2.2 Instruction Formats

COP2 Encoding of rs Field for all MXA instructions, see Appendix A.1.

Each allocated minor opcode is associated specific instruction formats as follows:

- 3RF (see Appendix A.2): 3-register floating-point or fixed-point operations coded in bits 25…22 with data format *df* coded in bit 21
- 2RF (see Appendix A.3): 2-register floating-point operations coded in bits 25…17 with data format *df* coded in bit 16
- 3R (see Appendix A.4): 3-register operations coded in bits 25…23 with data format *df* is coded in bits 22…21
- 2R (see Appendix A.5): 2-register operations coded in bits 25…18 with data format *df* is coded in bits 17…16
- ELM (see Appendix A.6): instructions with an immediate element index and data format *df/n* coded in bits 21…16, where the operation is coded in bits 25…22

3.2 Instruction Set Summary by Gategory

The MXA instruction set implements the following categories of instructions:

- MXA floating-point arithmetic (Table 3.5)
- MXA bitwise (Table 3.6)
- MXA floating-point conversions (Table 3.7)
- MXA Element Permute (Table 3.8)
- MXA load/store (Table 3.9)
- MXA Move (Table 3.10)
- MXA integer arithmetic(Table 3.11)

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

Table 3.5 MXA Floating-point Arithmetic Instructions

Mnemonic	Instruction
FCMUL_W	Floating-point Complex Multiplication

Table 3.6 MXA Bitwise Instructions

Mnemonic	Instruction
BMR.V	Vector Bit Move based on register
PCNT.V	Vector Population Count
BEXT.df	Vector Bit Extract
BEXP.df	Vector Bit Expand
BMU.V	Vector Bit Mask Update
SLL.V	Vector Shift Left
SLLI.V	Immediate Vector Shift Left
SRL.V	Vector Shift Right
SRLI.V	Immediate Vector Shift Right
MOVBT.B	Move byte extract by bit-mask
MOVBP.B	Move byte expand by bit-mask

Table 3.7 MXA Convert Instructions

Mnemonic	Instruction
FEXUPLC_W	Configurable Vector Floating-point Up-Convert Interchange Format Left
FEXUPRC_W	Configurable Vector Floating-point Up-Convert Interchange Format Right

Table 3.8 MXA Element Permute instructions

Mnemonic	Instruction
VSHFR.B	Vector Data Preserving Shuffle based on the register control vector
VSLDI.B	Immediate Vector Slide

Table 3.9 MXA Load/Store Instructions

Mnemonic	Instruction
LDINSS.df	Load element insert element from sparse memory
LDINSSU.df	Load element insert element and update base address

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

LDINSSU2.df	Load element insert element and update base address
LDINS.df	Load element insert element
LDINSU.df	Load element insert element and update base address
STEXT.df	Store extracted element
STEXTU.df	Store extracted element and update base address

Table 3.10 MXA Move Instructions

Mnemonic	Instruction
MOVE.W	Vector Move

Table 3.11 MXA Integer Arithmetic Instructions

Mnemonic	Instruction
ADDSL.df	Vector Add Left of signed Values
ADDSR.df	Vector Add Right of signed Values
ADDUL.df	Vector Add Left of unsigned Values
ADDUR.df	Vector Add Right of unsigned Values
ACCSL.df	Vector Accumulate Left of signed Values
ACCSR.df	Vector Accumulate Right of signed Values
ACCUL.df	Vector Accumulate Left of unsigned Values
ACCUR.df	Vector Accumulate Right of unsigned Values
SATSS.df	Fixed Signed Saturate
SATUS.df	Fixed Unsigned Saturate of Signed
SATUU.df	Fixed Unsigned Saturate of Unsigned
SUBSL.df	Vector Subtract Left of signed Values
SUBSR.df	Vector Subtract Right of signed Values
SUBUL.df	Vector Subtract Left of unsigned Values
SUBUR.df	Vector Subtract Right of unsigned Values
MULSL.df	Vector Multiply Left of signed Values
MULSR.df	Vector Multiply Right of signed Values
MULUL.df	Vector Multiply Left of unsigned Values
MULUR.df	Vector Multiply Right of unsigned Values

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

3.3 Alphabetical list of Instructions

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

ACCSL.df

Vector Accumulate Left of Signed Values

31 26	25 21	20 16	15 11	10 6	5 2	1 0
COP2 010010	2RC2 11010	funct1 00111	ws	wd	funct0 0000	df

Format: ACCSL.df

ACCSL.H wd, ws

ACCSL.W wd, ws

Description: wd[i] ← signed(left_half(ws)[i]) + (wd)[i]

The elements in vector wd are added to the left half signed integer elements in vector ws.

The operands and results are values in integer data format *df*.

Operation:

ACCSL.H

for i in 0... WRLEN/16-1

a ← $WR[ws]_{8i+7+WRLEN/2...8i+WRLEN/2}$

$WR[wd]_{16i+15...16i}$ ← $(a_7)^8$||a + $WR[wd]_{16i+15...16i}$

endfor

ACCSL.W

for i in 0... WRLEN/32-1

a ← $WR[ws]_{16i+15+WRLEN/2...16i+WRLEN/2}$

$WR[wd]_{32i+31...32i}$ ← $(a_{15})^{16}$||a + $WR[wd]_{32i+31...32i}$

endfor

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

ACCSR.df

Vector Accumulate Right of Signed Values

31 ... 26	25 ... 21	20 ... 16	15 ... 11	10 ... 6	5 ... 2	1 ... 0
COP2 010010	2RC2 11010	funct1 01000	ws	wd	funct0 0000	df

Format: ACCSR.df

ACCSR.H wd, ws

ACCSR.W wd, ws

Description: wd[i] ← signed(right_half(ws)[i]) + wd[i]

The elements in vector wd are added to the right half signed integer elements in vector ws. The operands and results are values in integer data format *df*.

Operation:

ACCSR.H

for i in 0... WRLEN/16-1

$a \leftarrow WR[ws]_{8i+7...8i}$

$WR[wd]_{16i+15...16i} \leftarrow (a_7)^8 || a + WR[wd]_{16i+15...16i}$

endfor

ACCSR.W

for i in 0... WRLEN/32-1

$a \leftarrow WR[ws]_{16i+15...16i}$

$WR[wd]_{32i+31...32i} \leftarrow (a_{15})^{16} || a + WR[wd]_{32i+31...32i}$

endfor

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

ACCUL.df

Vector Accumulate Left of Unsigned Values

31 — 26	25 — 21	20 — 16	15 — 11	10 — 6	5 — 2	1 — 0
COP2 010010	2RC2 11010	funct1 00101	ws	wd	funct0 0000	df

Format: ACCUL.df

ACCUL.H wd, ws

ACCUL.W wd, ws

Description: wd[i] ← unsigned(left_half(ws)[i]) + wd[i]

The elements in vector wd are added to the left half unsigned integer elements in vector ws. The operands and results are values in integer data format *df*.

Operation:

ACCUL.H

for i in 0... WRLEN/16-1

    a ← $WR[ws]_{8i+7+WRLEN/2...8i+WRLEN/2}$

    $WR[wd]_{16i+15...16i}$ ← 0^{8}||a + $WR[wd]_{16i+15...16i}$

endfor

ACCUL.W

for i in 0... WRLEN/32-1

    a ← $WR[ws]_{16i+15+WRLEN/2...16i+WRLEN/2}$

    $WR[wd]_{32i+31...32i}$ ← 0^{16}||a + $WR[wd]_{32i+31...32i}$

endfor

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

ACCUR.df

Vector Accumulate Right of Unsigned Values

31 26	25 21	20 16	15 11	10 6	5 2	1 0
COP2 010010	3RC2 11010	funct1 00110	ws	wd	funct0 0000	df

Format: ACCUR.df

ACCUR.H wd, ws

ACCUR.W wd, ws

Description: wd[i] ← unsigned(right_half(ws)[i]) + wd[i]

The elements in vector wd are added to the right half unsigned integer elements in vector ws. The operands and results are values in integer data format *df*.

Operation:

ACCUR.H

for i in 0... WRLEN/16-1

$a \leftarrow WR[ws]_{8i+7...8i}$

$WR[wd]_{16i+15...16i} \leftarrow 0^{8}||a + WR[wd]_{16i+15...16i}$

endfor

ACCUR.W

for i in 0... WRLEN/32-1

$a \leftarrow WR[ws]_{16i+15...16i}$

$WR[wd]_{32i+31...32i} \leftarrow 0^{16}||a + WR[wd]_{32i+31...32i}$

endfor

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

ADDSL.df

Vector Add Left of Signed Values

31 — 26	25 — 21	20 — 16	15 — 11	10 — 6	5 — 2	1 — 0
COP2 010010	3RC2 10001	wt	ws	wd	funct 0000	df

Format: ADDSL.df

ADDSL.H wd, ws, wt

ADDSL.W wd, ws, wt

Description: wd[i] ← signed(left_half(ws)[i]) + signed(left_half(wt)[i])

The left half signed integer elements in vector wt are added to the left half signed integer elements in vector ws.

The operands and results are values in integer data format *df*.

Operation:

ADDSL.H

for i in 0... WRLEN/16-1

$a \leftarrow WR[ws]_{8i+7+WRLEN/2...8i+WRLEN/2}$

$b \leftarrow WR[wt]_{8i+7+WRLEN/2...8i+WRLEN/2}$

$WR[wd]_{16i+15...16i} \leftarrow (a_7)^8||a + (b_7)^8||b$

endfor

ADDSL.W

for i in 0... WRLEN/32-1

$a \leftarrow WR[ws]_{16i+15+WRLEN/2...16i+WRLEN/2}$

$b \leftarrow WR[wt]_{16i+15+WRLEN/2...16i+WRLEN/2}$

$WR[wd]_{32i+31...32i} \leftarrow (a_{15})^{16}||a + (b_{15})^{16}||b$

endfor

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

ADDSR.df

Vector Add Right of Signed Values

31 ... 26	25 ... 21	20 ... 16	15 ... 11	10 ... 6	5 ... 2	1 0
COP2 010010	3RC2 10001	wt	ws	wd	funct 0001	df

Format: ADDSR.df

ADDSR.H wd, ws, wt

ADDSR.W wd, ws, wt

Description: wd[i] ← signed(right_half(ws)[i]) + signed(right_half(wt)[i])

The right half signed integer elements in vector wt are added to the left half signed integer elements in vector ws.

The operands and results are values in integer data format *df*.

Operation:

ADDSR.H

for i in 0... WRLEN/16-1

$a \leftarrow WR[ws]_{8i+7...8i}$

$b \leftarrow WR[wt]_{8i+7...8i}$

$WR[wd]_{16i+15...16i} \leftarrow (a_7)^8||a + (b_7)^8||b$

endfor

ADDSR.W

for i in 0... WRLEN/32-1

$a \leftarrow WR[ws]_{16i+15...16i}$

$b \leftarrow WR[wt]_{16i+15...16i}$

$WR[wd]_{32i+31...32i} \leftarrow (a_{15})^{16}||a + (b_{15})^{16}||b$

endfor

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

ADDUL.df

Vector Add Left of Unsigned Values

31 … 26	25 … 21	20 … 16	15 … 11	10 … 6	5 … 2	1 0
COP2 010010	3RC2 10001	wt	ws	wd	funct 0010	df

Format: ADDUL.df

ADDUL.H wd, ws, wt

ADDUL.W wd, ws, wt

Description: wd[i] ← unsigned(left_half(ws)[i]) + unsigned(left_half(wt)[i])

The left half unsigned integer elements in vector wt are added to the left half unsigned integer elements in vector ws.

The operands and results are values in integer data format *df*.

Operation:

ADDUL.H

for i in 0... WRLEN/16-1

a ← $WR[ws]_{8i+7+WRLEN/2...8i+WRLEN/2}$

b ← $WR[wt]_{8i+7+WRLEN/2...8i+WRLEN/2}$

$WR[wd]_{16i+15...16i}$ ← $0^8||a + 0^8||b$

endfor

ADDUL.W

for i in 0... WRLEN/32-1

a ← $WR[ws]_{16i+15+WRLEN/2...16i+WRLEN/2}$

b ← $WR[wt]_{16i+15+WRLEN/2...16i+WRLEN/2}$

$WR[wd]_{32i+31...32i}$ ← $0^{16}||a + 0^{16}||b$

endfor

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

ADDUR.df

Vector Add Right of Unsigned Values

31 26	25 21	20 16	15 11	10 6	5 2	1 0
COP2 010010	3RC2 10001	wt	ws	wd	funct 0011	df

Format: ADDUR.df

ADDUR.H wd, ws, wt

ADDUR.W wd, ws, wt

Description: wd[i] ← unsigned(right_half(ws)[i]) + unsigned(right_half(wt)[i])

The right half unsigned integer elements in vector wt are added to the right half unsigned integer elements in vector ws.

The operands and results are values in integer data format *df*.

Operation:

ADDUR.H

for i in 0... WRLEN/16-1

a ← $WR[ws]_{8i+7...8i}$

b ← $WR[wt]_{8i+7...8i}$

$WR[wd]_{16i+15...16i}$ ← $0^8||a + 0^8||b$

endfor

ADDUR.W

for i in 0... WRLEN/32-1

a ← $WR[ws]_{16i+15...16i}$

b ← $WR[wt]_{16i+15...16i}$

$WR[wd]_{32i+31...32i}$ ← $0^{16}||a + 0^{16}||b$

endfor

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

BEXP.df

Vector Bit Expand

31 26	25 21	20 16	15 11	10 6	5 2	1 0
COP2 010010	2RC2 11010	funct1 000<10|11>	ws	wd	funct0 0000	df

Format: BEXP.df

BEXP.2B wd, ws
BEXP.4B wd, ws
BEXP.B wd, ws
BEXP.H wd, ws
BEXP.W wd, ws

Description: wd ← bit_expand(ws)

The least significant bit of vector ws is expanded into all bits of vector wd elements. The destination element position is given by the bit position of vector ws.

The operands and results are values in integer data format *df*

Operation:

BEXP.2B
WR[wd] ← bit_expand(WR[ws],2)
BEXP.4B
WR[wd] ← bit_expand(WR[ws],4)
BEXP.B
WR[wd] ← bit_expand(WR[ws],8)
BEXP.H
WR[wd] ← bit_expand(WR[ws],16)
BEXP.W
WR[wd] ← bit_expand(WR[ws],32)

function bit_expand(a,df)
for i in WRLN/df-1..0
$z_{df*i+(df-1)..df*i} \leftarrow (a_i)^{df}$
endfor
return z
endfunction

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

BEXT.df

Vector Bit Extract

31 ... 26	25 ... 21	20 ... 16	15 ... 11	10 ... 6	5 ... 2	1 0
COP2 010010	2RC2 11010	funct1 00001	ws	wd	funct0 0000	df

Format: BEXT.df

BEXT.B wd, ws

BEXT.H wd, ws

BEXT.W wd, ws

Description: wd ← bit_extract(ws)

The most significant bit of vector ws elements is extracted into vector wd. The destination bit position is given by the element position. The other destination bits are set to 0.

The operands and results are values in integer data format *df*

Operation:

```
BEXT.B
      WR[wd] ← bit_extract(WR[ws],8)
BEXT.H
      WR[wd] ← bit_extract(WR[ws],16)
BEXT.W
      WR[wd] ← bit_extract(WR[ws],32)
function bit_extract(a,df)
      z ← 0
      for i in WRLN/df-1..0
            z_i ← a_df*i+(df-1)
      endfor
      return z
endfunction
```

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

BMR.V

Vector Bit Move based on register

31 ... 26	25 ... 21	20 ... 16	15 ... 11	10 ... 6	5 ... 1	0
COP2 010010	4R 11101	wt	ws	wd	wr	1

Format: BMR.V

BMR.V   wd, ws, wt, wr

Description: wd ← (ws AND wr) OR (wt AND NOT wr)

Copy to destination vector wd all bits from source vector wt for which the corresponding bits from target vector wr are 0 or from source vector ws for which the corresponding target vector wr are 1.

The operands and results are bit vector values.

Operation:

BMR.V

WR[wd] ← (WR[ws] and WR[wr]) or (WR[wt] and not WR[wr])

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

BMU.V

Vector Bit Mask Update

31 ... 26	25 ... 21	20 ... 16	15 ... 11	10 ... 6	5 ... 2	1 0
COP2 010010	3RC2 10010	wt	ws	wd	funct 0000	df 01

Format: BMU.V

BMU.V   wd, ws, wt

Description: wd ← bit_mask_upate(ws, wd, wt)

Update the vector wd low 0~wt[3:0] bits. Copy the vector ws to the corresponding position of the vector wd from low to high if corresponding bit is 1 of the vector wd, set else to zero.

The operands and results are bit vector values.

Operation:

```
BMU.V
    WR[wd] ← bit_mask_update(WR[ws],WR[wd],WR[wt])
function bit_mask_update(a,b,c)
    z ← b
    j ← 0
    for i in 0...c3...0
        if bi = 1 then
            zi ← aj
            j ← j + 1
        endif
    endfor
    return z
endfunction
```

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

FCMUL.df

Floating-point Complex Multiplication

31 … 26	25 … 21	20 … 16	15 … 11	10 … 6	5 … 1	0
COP2 010010	3RFC2 10000	wt	ws	wd	funct 00011	df

Format: FCMUL.df

FCMUL.W   wd, ws, wt

Description: wd[i] ← complex_mul(ws[i], wt[i])

The elements in vector wt are complex multiplied to the elements in vector ws.

The normal operand is as the same as IEEE Standard for Floating-point Arithmetic 754-2008. The subnormal operand and result are saturated into plus/minus zero. The NaN or Inf result is saturated into plus/minus maximum normal number.

The operands are not affected by the bit FS or NX in MSA Control and Status Register MSACSR.

The operands and results are values in floating-point data format *df*.

Operation:

```
FCMUL.W
for i in 0 ... WRLEN/64 -1
    WR[wd]64i+63...64i ← complex_mul( WR[ws]64i+63...64i, WR[wt]64i+63...64i )
endfor
function operand_update(a)
    if isinf(a) or isnan(a) then
        z[31] ← copysign(a)
        z[30:0] ← 1^7 || 0 || 1^23
    endif
    if isdenormal (a) then
        z[31] ← copysign(a)
        z[30:0] ←0^31
    else
        z ← a
    endif
    return z
endfunction
function operand_saturate(a)
  if isdenormal (a) then
        z[31] ← copysign(a)
        z[30:0] ←0^31
    else
        z ← a
    endif
    return z
endfunction
function complex_mul(a,b)
```


Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

```
    areal     ←  operand_saturate( a[31:00] )
    aimag     ←  operand_saturate( a[63:32] )
    breal     ←  operand_saturate( b[31:00] )
    bimag     ←  operand_saturate( b[63:32] )
    z[31:0]   ←  operand_update(areal * breal   - aimag * bimag)
    z[63:32]  ←  operand_update(areal * bimag  + aimag * breal)
    return z
endfucntion
```

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

FEXUPLC.df

Configurable Vector Floating-Point Up-Convert Interchange Format Left

31 … 26	25 … 21	20 … 16	15 … 11	10 … 6	5 … 1	0
COP2 010010	3RFC2 10000	wt	ws	wd	funct 00000	df

Format: FEXUPLC.df

FEXUPLC.W   wd, ws,wt

Description: wd[i] ← up_convert_xhs(left_half(ws)[i], left_half(wt)[i].ebits, left_half(wt)[i].bias)

The left half floating-point elements in vector ws are up-converted to a larger interchange format. The result is written to vector wd.

The format up-covert operation is defined by XBurst and is different with IEEE standard for Floating-point Arithmetic 754-2008.

The ws format is 16-bit XHS. The wt format is XBurst 16-bit control value. The wd format is IEEE standard 32-bit floating-point data.

Operation:

FEXUPLC.W

for i in 0 ... WRLEN/32-1

bias ← $WR[wt]_{16i+10+WRLEN/2...16i+4+WRLEN/2}$

ebits ← $WR[wt]_{16i+3+WRLEN/2...16i+WRLEN/2}$

f ← up_covert_xhs($WR[ws]_{16i+15+WRLEN/2...16i+WRLEN/2}$, bias, ebits)

$WR[wd]_{32i+31...32i}$ ← f

endfor

function up_covert_xhs(a, ebits, bias)

/*Implementation XHS format up-conversion */

endfunction

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

FEXUPRC.df

Configurable Vector Floating-Point Up-Convert Interchange Format Right

31 … 26	25 … 21	20 … 16	15 … 11	10 … 6	5 … 1	0
COP2 010010	3RFC2 10000	wt	ws	wd	funct 00100	df

Format: FEXUPRC.df

FEXUPRC.W wd, ws, wt

Description: wd[i] ← up_convert_xhs(right_half(ws)[i], right_half(wt)[i].ebits, right_half(wt)[i]bias)

The right half floating-point elements in vector ws are up-converted to a larger interchange format. The result is written to vector wd.

The format up-covert operation is defined by XBurst and is different with IEEE standard for Floating-point Arithmetic 754-2008.

The ws format is 16-bit XHS. The wt format is XBurst 16-bit control value. The wd format is IEEE standard 32-bit floating-point data.

Operation:

```
FEXUPRC.W
    for i in 0... WRLEN/32-1
        bias ← WR[wt]16i+10...16i+4
        ebits ← WR[wt]16i+3...16i
        f ← up_covert_xhs(WR[ws]16i+15...16i, ebits, bias)
        WR[wd]32i+31...32i ← f
    endfor
function up_covert_xhs(a, ebits, bias)
    /*Implementation XHS format up-conversion */
endfunction
```

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

LDINS.df

Load element insert element

31 28	25 21	20 16	15 11	10 6	5 0
COP2 010010	ELMC2 10101	base	index	wd	df/n

Format: LDINS.df

LDINS.W wd[n],index(base)

LDINS.D wd[n],index(base)

Description: wd[n] ← memory(GPR[base]+GPR[index])

The contents of the vector at the memory location specified by the aligned effective address are fetched and placed into the vector wd[n]. The contents of GPR *index* and GPR *base* are added to form the effective address.

Operation:

LDINS.W

vAddr ← GPR[base]+GPR[index]

if $vAddr_{1...0} \neq 0^2$

SignalException(AddressError)

endif

(pAdd,CCA) ← AddressTranslation(vAddr, DATA, LOAD)

memword ← LoadMemory(CCA, WORD, pAddr, vAddr, DATA)

$WR[wd]_{32n+31...32n}$ ← memword

LDINS.D

vAddr ← GPR[base]+GPR[index]

if $vAddr_{2...0} \neq 0^3$

SignalException(AddressError)

endif

(pAdd,CCA) ← AddressTranslation(vAddr, DATA, LOAD)

memdword ← LoadMemory(CCA, DWORD, pAddr, vAddr, DATA)

$WR[wd]_{64n+63...64n}$ ← memdword

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception, Address Error Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

LDINSS.df

Load element insert element from sparse memory

31 26	25 21	20 16	15 11	10 6	5 0
COP2 010010	ELMC2 10101	base	index	wd	df/n

Format: LDINSS.df

LDINSS.W wd[n],index(base)

Description: wd[n] ← memory(GPR[base]+ unsigned((GPR$[index]_{8n+7...8n}$+1) || 0^2))

The contents of the vector at the memory location specified by the aligned effective address are fetched and placed into the vector wd[n]. The contents of unsigned GPR *index* specify modulo the size of the element in bits and GPR *base* are added to form the effective address.

Operation:

```
LDINSS.W
    vAddr ← GPR[base] + unsigned((GPR[index]8n+7...8n+1) || 0^2)
    if vAddr1...0 ≠ 0^2
        SignalException(AddressError)
    endif
    (pAdd,CCA) ← AddressTranslation(vAddr, DATA, LOAD)
    memword ← LoadMemory(CCA, WORD, pAddr, vAddr, DATA)
    WR[wd]32n+31...32n←memword
```

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception, Address Error Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

LDINSSU.df

Load element insert element and update base address

31 — 26	25 — 21	20 — 16	15 — 11	10 — 6	5 — 0
COP2 010010	ELMC2 10110	base	index	wd	df/n

Format: LDINSSU.df

LDINSSU.W wd[n],index(base)

Description: wd[n] ← memory(GPR[base] + unsigned((GPR[index]$_{8n+7...8n}$+1) || 0^2))

GPR[base] ←GPR[base] + unsigned((GPR[index]$_{8n+7...8n}$+1) || 0^2)

The contents of the vector at the memory location specified by the aligned effective address are fetched and placed into the vector wd[n], and update base address. The contents of unsigned GPR *index* specify modulo the size of the element in bits and GPR *base* are added to form the effective address.

Operation:

```
LDINSSU.W
    vAddr ← GPR[base] + unsigned(GPR[index]8n+7...8n+1 || 0^2)
    if vAddr1...0 ≠ 0^2
        SignalException(AddressError)
    endif
    (pAdd,CCA) ← AddressTranslation(vAddr, DATA, LOAD)
    memword ← LoadMemory(CCA, WORD, pAddr, vAddr, DATA)
    WR[wd]32n+31...32n ← memword
    GPR[base] ← vAddr
```

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception, Address Error Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

LDINSSU2.df

Load element insert element and update base address

31 — 26	25 — 21	20 — 16	15 — 11	10 — 6	5 — 0
COP2 010010	ELMC2 10111	base	index	wd	df/n

Format: LDINSSU2.df

LDINSSU2.W   wd[n],index(base)

Description: wd[n] ← memory(GPR[base] + unsigned((GPR[index]$_{4n+3...4n}$+1) || 0^2))

GPR[base] ←GPR[base] + unsigned((GPR[index]$_{4n+3...4n}$+1) || 0^2)

The contents of the vector at the memory location specified by the aligned effective address are fetched and placed into the vector wd[n], and update base address. The contents of unsigned GPR *index* specify modulo the size of the element in bits and GPR *base* are added to form the effective address.

Operation:

```
LDINSSU2.W
    vAddr ← GPR[base] + unsigned(GPR[index]4n+3...4n+1 || 0^2)
    if vAddr1...0 ≠ 0^2
        SignalException(AddressError)
    endif
    (pAdd,CCA) ← AddressTranslation(vAddr, DATA, LOAD)
    memword ← LoadMemory(CCA, WORD, pAddr, vAddr, DATA)
    WR[wd]32(n MOD 4)+31...32(n MOD 4) ← memword
    GPR[base] ← vAddr
```

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception, Address Error Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

LDINSU.df

Load element insert element and update base address

31 26	25 21	20 16	15 11	10 6	5 0
COP2 010010	ELMC2 10111	base	index	wd	df/n

Format: LDINSU.df

LDINSU.W wd[n], index(base)

LDINSU.D wd[n], index(base)

Description: wd[n] ← memory(GPR[base]+ GPR[index])

GPR[base] ← GPR[base]+ GPR[index]

The contents of the vector at the memory location specified by the aligned effective address are fetched and placed into the vector wd[n] and update base address. The contents of GPR *index* and GPR *base* are added to form the effective address.

Operation:

```
LDINSU.W
    vAddr ← GPR[base] + GPR[index]
    if vAddr_{1...0} ≠ 0^2
        SignalException(AddressError)
    endif
    (pAdd,CCA) ← AddressTranslation(vAddr, DATA, LOAD)
    memword ← LoadMemory(CCA, WORD, pAddr, vAddr, DATA)
    WR[wd]_{32n+31...32n} ← memword
    GPR[base] ← vAddr
LDINSU.D
    vAddr ← GPR[base] + GPR[index]
    if vAddr_{2...0} ≠ 0^3
        SignalException(AddressError)
    endif
    (pAdd,CCA) ← AddressTranslation(vAddr, DATA, LOAD)
    memdword ← LoadMemory(CCA, DWORD, pAddr, vAddr, DATA)
    WR[wd]_{64n+63...64n}←memdword
    GPR[base] ← vAddr
```

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception, Address Error Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

MOVBP.B

Move byte expand by bit-mask

31 ... 26	25 ... 21	20 ... 16	15 ... 11	10 ... 6	5 ... 2	1 ... 0
COP2 010010	3RC2 10010	wt	ws	wd	funct 1001	df 00

Format: MOVBP.B

MOVBP.B wd, ws, wt

Description: wd ←move_expand(ws,wt)

Move byte expand by bit-mask. Move a byte of vector ws to the corresponding position of the vector wd from low to high if the low bit of vector wt is 1. Set else to zero.

The operands and results are in integer data format byte.

Operation:

```
MOVBP.B
    WR[wd] ← move_expand(WR[ws], WR[wt], 8)
function move_expand(a,b,df)
    z ← 0
    j ← 0
    for i in 0...WRLN/df-1
        if b_i = 1 then
            z_{df*i+(df-1)...df*i} ← a_{df*j+(df-1)...dfj}
            j ← j + 1
        endif
    endfor
endfunction
```

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

MOVBT.B

Move byte extract by bit-mask

31 26	25 21	20 16	15 11	10 6	5 2	1 0
COP2 010010	3RC2 10010	wt	ws	wd	funct 1000	df 00

Format: MOVBT.B

MOVBT.B wd, ws, wt

Description: wd ←move_extract(ws,wt)

Move byte extract by bit-mask. Move the corresponding position of byte in vector ws to the position of the vector wd from low to high if the low bit of vector wt is 1. Set else to zero.

The operands and results are in integer data format byte..

Operation:

```
MOVBT.B
    WR[wd] ← move_extract(WR[ws], WR[wt], 8)
function move_extract(a,b,df)
    z ← 0
    j ← 0
    for i in 0...WRLN/df-1
        if b_i = 1 then
            z_{df*j+(df-1)...df*j} ← a_{df*i+(df-1)...df*i}
            j ← j + 1
        endif
    endfor
endfunction
```

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

MOVE.W

Vector Move

31 — 26	25 — 21	20 — 16	15 — 11	10 — 6	5 — 0
COP2 010010	ELM2RC2 10100	funct 00000	ws	wd	df/mn

Format: MOVE.W

MOVE.W wd[m], ws[n]

Description: wd[m] ← ws[n]

Vector Move. Move the *n*th elements of vector ws to the *m*th elements of vector wd.

The operands and results are in integer data format word.

Operation:

MOVE.W

$WR[wd]_{32m+31...32m} \leftarrow WR[ws]_{32n+31...32n}$

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

MULSL.df

Vector Multiply Left of Signed Values

31 26	25 21	20 16	15 11	10 6	5 2	1 0
COP2 010010	3RC2 10001	wt	ws	wd	funct 1100	df

Format: MULSL.df

MULSL.H wd, ws, wt

MULSL.W wd, ws, wt

Description: wd[i] ← signed(left_half(ws)[i]) * signed(left_half(wt)[i])

The signed integer elements in left half vector wt are multiplied by signed integer elements in left half vector ws. The result is written to vector wd.

The operands and results are values in integer data format *df*

Operation:

MULSL.H

for i in 0... WRLEN/16-1

$a \leftarrow WR[ws]_{8i+7+WRLEN/2...8i+WRLEN/2}$

$b \leftarrow WR[wt]_{8i+7+WRLEN/2...8i+WRLEN/2}$

$WR[wd]_{16i+15...16i} \leftarrow (a_7)^8||a * (b_7)^8||b$

endfor

MULSL.W

for i in 0... WRLEN/32-1

$a \leftarrow WR[ws]_{16i+15+WRLEN/2...16i+WRLEN/2}$

$b \leftarrow WR[wt]_{16i+15+WRLEN/2...16i+WRLEN/2}$

$WR[wd]_{32i+31...32i} \leftarrow (a_{15})^{16}||a * (b_{15})^{16}||b$

endfor

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

MULSR.df

Vector Multiply Right of Signed Values

31 — 26	25 — 21	20 — 16	15 — 11	10 — 6	5 — 2	1 — 0
COP2 010010	3RC2 10001	wt	ws	wd	funct 1101	df

Format: MULSR.df

MULSR.H wd, ws, wt

MULSR.W wd, ws, wt

Description: wd[i] ← signed(right_half(ws)[i]) * signed(right_half(wt)[i])

The signed integer elements in right half vector wt are multiplied by signed integer elements in right half vector ws. The result is written to vector wd.

The operands and results are values in integer data format *df.*

Operation:

MULSR.H

for i in 0... WRLEN/16-1

$a \leftarrow WR[ws]_{8i+7...8i}$

$b \leftarrow WR[wt]_{8i+7...8i}$

$WR[wd]_{16i+15...16i} \leftarrow (a_7)^8||a * (b_7)^8||b$

endfor

MULSR.W

for i in 0... WRLEN/32-1

$a \leftarrow WR[ws]_{16i+15...16i}$

$b \leftarrow WR[wt]_{16i+15...16i}$

$WR[wd]_{32i+31...32i} \leftarrow (a_{15})^{16}||a * (b_{15})^{16}||b$

endfor

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

MULUL.df

Vector Multiply Left of Unsigned Values

31 … 26	25 … 21	20 … 16	15 … 11	10 … 6	5 … 2	1 … 0
COP2 010010	3RC2 10001	wt	ws	wd	funct 1110	df

Format: MULUL.df

MULUL.H wd, ws, wt

MULUL.W wd, ws, wt

Description: wd[i] ← unsigned(left_half(ws)[i]) * unsigned(left_half(wt)[i])

The unsigned integer elements in left half vector wt are multiplied by unsigned integer elements in left half vector ws. The result is written to vector wd.

The operands and results are values in integer data format *df*

Operation:

MULUL.H

for i in 0... WRLEN/16-1

a ← $WR[ws]_{8i+7+WRLEN/2...8i+WRLEN/2}$

b ← $WR[wt]_{8i+7+WRLEN/2...8i+WRLEN/2}$

$WR[wd]_{16i+15...16i}$ ← 0^8||a * 0^8||b

endfor

MULUL.W

for i in 0... WRLEN/32-1

a ← $WR[ws]_{16i+15+WRLEN/2...16i+WRLEN/2}$

b ← $WR[wt]_{16i+15+WRLEN/2...16i+WRLEN/2}$

$WR[wd]_{32i+31...32i}$ ← 0^{16}||a * 0^{16}||b

endfor

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

MULUR.df

Vector Multiply Right of Unsigned Values

31 … 26	25 … 21	20 … 16	15 … 11	10 … 6	5 … 2	1 0
COP2 010010	3RC2 10001	wt	ws	wd	funct 1111	df

Format: MULUR.df

MULUR.H wd, ws, wt

MULUR.W wd, ws, wt

Description: wd[i] ← unsigned(right_half(ws)[i]) * unsigned(right_half(wt)[i])

The unsigned integer elements in right half vector wt are multiplied by unsigned integer elements in right half vector ws. The result is written to vector wd.

The operands and results are values in integer data format *df.*

Operation:

MULUR.H

for i in 0... WRLEN/16-1

$a \leftarrow WR[ws]_{8i*7...8i}$

$b \leftarrow WR[wt]_{8i*7...8i}$

$WR[wd]_{16i*15...16i} \leftarrow 0^8||a * 0^8||b$

endfor

MULUR.W

for i in 0... WRLEN/32-1

$a \leftarrow WR[ws]_{16i*15...16i}$

$b \leftarrow WR[wt]_{16i*15...16i}$

$WR[wd]_{32i*31...32i} \leftarrow 0^{16}||a * 0^{16}||b$

endfor

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

PCNT.V

Vector Population Count

31 26	25 21	20 16	15 11	10 6	5 2	1 0
COP2 010010	2RC2 11010	funct1 00000	ws	wd	funct0 0000	df 00

Format: PCNT.V
PCNT.V wd, ws

Description: wd ← population_count(ws)

The number of bits set to 1 for vector ws is stored to the byte element 0 in the vector wd, and the other destination byte elements are set to 0.

The operands and results are values in integer data format *df*.

Operation:

```
PCNT.V
    WR[wd]7...0 ← population_count(WR[ws])
    for i in 1 ... WRLEN/8-1
        WR[wd]8i+7...8i ← 0
function population_count(a)
    z ← 0
    for i in WRLEN-1..0
        if a_i =1 then
            z ← z + 1
        endif
    endfor
    return z
endfunction
```

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

SATSS.df

Fixed Signed Saturate of Signed Values

31 … 26	25 … 21	20 … 16	15 … 11	10 … 6	5 … 2	1 … 0
COP2 010010	3RC2 10010	wt	ws	wd	funct 0010	df

Format: SATSS.df

SATSS.B wd, ws, wt

SATSS.H wd, ws, wt

Description: wd ← saturate_signed_s(signed(ws),signed(wt),df)

Signed elements in vector ws and wt are saturated to singed values of *df* bits without changing the data width. The result is written to vector wd.

The operands and results are values in integer data format *df*.

Operation:

SATSS.B

v ←WR[ws] || WR[wt]

for i in 0...WRLN*2/16-1

$WR[wd]_{8*i+7...8*i} \leftarrow$ saturate_signed_s($v_{16*i+15...16*i}$, 8)

endfor

SATSS.H

v ←WR[ws] || WR[wt]

for i in 0...WRLN*2/32-1

$WR[wd]_{16*i+15...16*i} \leftarrow$ saturate_signed_s($v_{32*i+31...32*i}$, 16)

endfor

function saturate_signed_s(a,df)

if $a_{2*df-1} = 0$ and $a_{2*df-2..df-1} \neq 0^{df}$ then

$t \leftarrow 0 \,||\, 1^{df-1}$

else if $tt_{2*df-1} = 1$ and $tt_{2*df-2..df-1} \neq 1^{df}$ then

$t \leftarrow 1 \,||\, 0^{df-1}$

else

$t \leftarrow a_{df-1 \,...\, 0}$

endif

endif

return t

endfunction

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

SATUS.df

Fixed Unsigned Saturate of Signed Value

31 ... 26	25 ... 21	20 ... 16	15 ... 11	10 ... 6	5 ... 2	1 ... 0
COP2 010010	3RC2 10010	wt	ws	wd	funct 0100	df

Format: SATUS.df

SATUS.B   wd, ws, wt

SATUS.H   wd, ws, wt

Description: wd ← saturate_unsigned_s(signed(ws),signed(wt),df)

Signed elements in vector ws and wt are saturated to unsinged values of *df* bits without changing the data width. The result is written to vector wd.

The operands and results are values in integer data format *df*.

Operation:

SATUS.B

v ←WR[ws] || WR[wt]

for i in 0...WRLN*2/16-1

$WR[wd]_{8*i+7...8*i}$ ← saturate_unsigned_s($v_{16*i+15...16*i}$, 8)

endfor

SATUS.H

v ←WR[ws] || WR[wt]

for i in 0...WRLN*2/32-1

$WR[wd]_{16*i+15...16*i}$ ← saturate_unsigned_s($v_{32*i+31...32*i}$, 16)

endfor

function saturate_unsigned_s(a,df)

if a_{2*df-1} = 0 and $a_{2*df-2..df} \neq 0^{df-1}$ then

t← 1^{df}

else if tt_{2*df-1} = 1 then

t ← 0^{df}

else

t ← $a_{df-1 \ldots 0}$

endif

endif

return t

endfunction

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

SATUU.df

Fixed Unsigned Saturate of Unsigned Value

31 26	25 21	20 16	15 11	10 6	5 2	1 0
COP2 010010	3RC2 10010	wt	ws	wd	funct 0011	df

Format: SATUU.df

SATUU.B wd, ws, wt

SATUU.H wd, ws, wt

Description: wd ← saturate_unsigned_u(unsigned(ws),unsigned(wt),df)

Unsigned elements in vector ws and wt are saturated to unsigned values of *df* bits without changing the data width. The result is written to vector wd.

The operands and results are values in integer data format *df*.

Operation:

```
SATUU.B
    v ←WR[ws] || WR[wt]
    for i in 0...WRLN/16-1
        WR[wd]_{8*i+7...8*i} ← saturate_unsigned_u(v_{16*i+15...16*i} , 8)
    endfor
SATUU.H
    v ←WR[ws] || WR[wt]
    for i in 0...WRLN/32-1
        WR[wd]_{16*i+15...16*i} ← saturate_unsigned_u(v_{32*i+31...32*i} , 16)
    endfor
function saturate_unsigned_u(a, df)
    if a_{2*df-1..df} ≠ 0^df then
        t_{df*i+df-1...df*i} ← 1^df
    else
        t_{df*i+df-1...df*i} ← tt_{df-1 ... 0}
    endif
    return t
endfunction
```

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

SLL.V

Vector Shift Left

31 … 26	25 … 21	20 … 16	15 … 11	10 … 6	5 … 2	1 0
COP2 010010	3RC2 10010	wt	ws	wd	funct 0000	df 00

Format: SLL.V

SLL.V wd, ws, wt

Description: wd ← ws << wt

The vector ws are shifted left by the number of bits the elements in vector wt specify modulo the size of the element in bits. The result is written to vector wd.

The operands and results are values in bit vector values.

Operation:

SLL.V

WR[wd] ← WR[ws] << $WR[wt]_{6...0}$

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

SLLI.V

Immediate Vector Shift Left

31 26	25 21	20 16	15 11	10 6	5 2	1 0
COP2 010010	2RC2 11010	m	ws	wd	funct 1110	df 00

Format: SLLI.V

SLLI.V wd, ws, m

Description: wd ← ws << m

The vector ws are shifted left by *m* bits. The result is written to vector wd.

The operands and results are values in bit vector values.

Operation:

SLLI.V

WR[wd] ← WR[ws] << m

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

SRL.V

Vector Shift Right

31 … 26	25 … 21	20 … 16	15 … 11	10 … 6	5 … 2	1 0
COP2 010010	3RC2 10010	wt	ws	wd	funct 0001	df 00

Format: SRL.V

SRL.V wd, ws, wt

Description: wd ← ws >> wt

The vector ws are shifted right by the number of bits the elements in vector wt specify modulo the size of the element in bits. The result is written to vector wd.

The operands and results are values in bit vector values.

Operation:

SRL.V

WR[wd] ← WR[ws] >> $WR[wt]_{6...0}$

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

SRLI.V

Immediate Vector Shift Right

31 26	25 21	20 16	15 11	10 6	5 2	1 0
COP2 010010	2RC2 11010	m	ws	wd	funct 1111	df 00

Format: SRLI.V

SRLI.V wd, ws, m

Description: wd ← ws >> m

The vector ws are shifted right by *m* bits. The result is written to vector wd.

The operands and results are values in bit vector values.

Operation:

SRLI.V

WR[wd] ← WR[ws] >> m

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

STEXT.df

Store extracted element

31 … 28	25 … 21	20 … 16	15 … 11	10 … 6	5 … 0
COP2 010010	ELMC2 10110	base	index	wd	df/n

Format: STEXT.df

STEXT.W   wd[n], index(base)

STEXT.D   wd[n], index(base)

Description: memory(GPR[base]+GPR[index]) ← wd[n]

The *n*th elements of vector wd is stored in memory at the location specified by the aligned effective address. The contents of GPR *index* and GPR *base* are added to form the effective address.

Operation:

```
STEXT.W
    vAddr ← GPR[base] + GPR[index]
    if vAddr_{1...0} ≠ 0^2
        SignalException(AddressError)
    endif
    (pAdd,CCA) ← AddressTranslation(vAddr, DATA, LOAD)
    dataword ← WR[wd]_{32n+31...32n}
    StoreMemory(CCA, WORD, dataword, pAddr, vAddr, DATA)
STEXT.D
    vAddr ← GPR[base] + GPR[index]
    if vAddr_{2...0} ≠ 0^3
        SignalException(AddressError)
    endif
    (pAdd,CCA) ← AddressTranslation(vAddr, DATA, LOAD)
    datadword ← WR[wd]_{64n+63...64n}
    StoreMemory(CCA, DWORD, datadword, pAddr, vAddr, DATA)
```

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception, Address Error Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

STEXTU.df

Store extracted element and update base address

31 — 26	25 — 21	20 — 16	15 — 11	10 — 6	5 — 0
COP2 010010	ELM2RC2 10100	funct 00001	base	wd	df/n

Format: STEXTU.df

STEXTU.W wd[n], base

STEXTU.D wd[n], base

Description: memory(GPR[base]+sizeof(df)) ← wd[n]

GPR[base] ← GPR[base]+sizeof(df)

The *n*th elements of vector wd is stored in memory at the location specified by the aligned effective address, and update base address. The contents of GPR *index* and GPR *base* are added to form the effective address.

Operation:

```
STEXTU.W
    vAddr ← GPR[base] + 4
    if vAddr_{1...0} ≠ 0^2
        SignalException(AddressError)
    endif
    (pAdd,CCA) ← AddressTranslation(vAddr, DATA, LOAD)
    dataword ← WR[wd]_{32n+31...32n}
    StoreMemory(CCA, WORD, dataword, pAddr, vAddr, DATA)
    GPR[base] ← vAddr
STEXTU.D
    vAddr ← GPR[base] + 8
    if vAddr_{2...0} ≠ 0^3
        SignalException(AddressError)
    endif
    (pAdd,CCA) ← AddressTranslation(vAddr, DATA, LOAD)
    datadword ← WR[wd]_{64n+63...64n}
    StoreMemory(CCA, DWORD, datadword, pAddr, vAddr, DATA)
    GPR[base] ← vAddr
```

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception, Address Error Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

SUBSL.df

Vector Subtract Left of Signed Values

31 26	25 21	20 16	15 11	10 6	5 2	1 0
COP2 010010	3RC2 10001	wt	ws	wd	funct 1000	df

Format: SUBSL.df

SUBSL.H wd, ws, wt

SUBSL.W wd, ws, wt

Description: wd[i] ← signed(left_half(ws)[i]) - signed(left_half(wt)[i])

The left half elements in signed integer vector wt are subtracted from the left half elements in signed integer vector ws.

The operands and results are values in integer data format *df.*

Operation:

SUBSL.H

for i in 0... WRLEN/16-1

$a \leftarrow WR[ws]_{8i+7+WRLEN/2...8i+WRLEN/2}$

$b \leftarrow WR[wt]_{8i+7+WRLEN/2...8i+WRLEN/2}$

$WR[wd]_{16i+15...16i} \leftarrow (a_7)^8||a - (b_7)^8||b$

endfor

SUBSL.W

for i in 0... WRLEN/32-1

$a \leftarrow WR[ws]_{16i+15+WRLEN/2...16i+WRLEN/2}$

$b \leftarrow WR[wt]_{16i+15+WRLEN/2...16i+WRLEN/2}$

$WR[wd]_{32i+31...32i} \leftarrow (a_{15})^{16}||a - (b_{15})^{16}||b$

endfor

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

SUBSR.df

Vector Subtract Right of Signed Values

31 ... 26	25 ... 21	20 ... 16	15 ... 11	10 ... 6	5 ... 2	1 0
COP2 010010	3RC2 10001	wt	ws	wd	funct 1001	df

Format: SUBSR.df

SUBSR.H wd, ws, wt

SUBSR.W wd, ws, wt

Description: wd[i] ← signed(right_half(ws)[i]) - signed(right_half(wt)[i])

The right half elements in signed integer vector wt are subtracted from the right half elements in signed integer vector ws.

The operands and results are values in integer data format *df.*

Operation:

SUBSR.H

for i in 0... WRLEN/16-1

$a \leftarrow WR[ws]_{8i+7...8i}$

$b \leftarrow WR[wt]_{8i+7...8i}$

$WR[wd]_{16i+15...16i} \leftarrow (a_7)^8||a - (b_7)^8||b$

endfor

SUBSR.W

for i in 0... WRLEN/32-1

$a \leftarrow WR[ws]_{16i+15...16i}$

$b \leftarrow WR[wt]_{16i+15...16i}$

$WR[wd]_{32i+31...32i} \leftarrow (a_{15})^{16}||a - (b_{15})^{16}||b$

endfor

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

SUBUL.df

Vector Subtract Left of Unsigned Values

31 … 26	25 … 21	20 … 16	15 … 11	10 … 6	5 … 2	1 0
COP2 010010	3RC2 10001	wt	ws	wd	funct 1010	df

Format: SUBUL.df

SUBUL.H wd, ws, wt

SUBUL.W wd, ws, wt

Description: wd[i] ← unsigned(left_half(ws)[i]) - unsigned(left_half(wt)[i])

The left half elements in unsigned integer vector wt are subtracted from the left half elements in unsigned integer vector ws.

The operands and results are values in integer data format *df.*

Operation:

SUBUL.H

for i in 0... WRLEN/16-1

$a \leftarrow WR[ws]_{8i+7+WRLEN/2...8i+WRLEN/2}$

$b \leftarrow WR[wt]_{8i+7+WRLEN/2...8i+WRLEN/2}$

$WR[wd]_{16i+15...16i} \leftarrow 0^{8}||a - 0^{8}||b$

endfor

SUBUL.W

for i in 0... WRLEN/32-1

$a \leftarrow WR[ws]_{16i+15+WRLEN/2...16i+WRLEN/2}$

$b \leftarrow WR[wt]_{16i+15+WRLEN/2...16i+WRLEN/2}$

$WR[wd]_{32i+31...32i} \leftarrow 0^{16}||a - 0^{16}||b$

endfor

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

SUBUR.df

Vector Subtract Right of Unsigned Values

31 26	25 21	20 16	15 11	10 6	5 2	1 0
COP2 010010	3RC2 10001	wt	ws	wd	funct 1011	df

Format: SUBUR.df

SUBUR.H   wd, ws, wt

SUBUR.W   wd, ws, wt

Description: wd[i] ← unsigned(right_half(ws)[i]) - unsigned(right_half(wt)[i])

The right half elements in unsigned integer vector wt are subtracted from the right half elements in unsigned integer vector ws.

The operands and results are values in integer data format *df.*

Operation:

SUBUR.H

for i in 0... WRLEN/16-1

$a \leftarrow WR[ws]_{8i+7...8i}$

$b \leftarrow WR[wt]_{8i+7...8i}$

$WR[wd]_{16i+15...16i} \leftarrow 0^8||a - 0^8||b$

endfor

SUBUR.W

for i in 0... WRLEN/32-1

$a \leftarrow WR[ws]_{16i+15...16i}$

$b \leftarrow WR[wt]_{16i+15...16i}$

$WR[wd]_{32i+31...32i} \leftarrow 0^{16}||a - 0^{16}||b$

endfor

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

VSHFR.B

Vector Data Preserving Shuffle based on the register control vector

31 ... 26	25 ... 21	20 ... 16	15 ... 11	10 ... 6	5 ... 1	0
COP2 010010	4R 11101	wt	ws	wd	wr	0

Format: VSHFR.B

VSHFR.B wd, ws, wt, wr

Description: wd ← vector_shuffle(control(wr),ws,wt)

The instruction function is as the same as the MSA's VSHF.B

The vector shuffle instructions selectively copy data elements from the concatenation of vectors ws and wt into vector wd based on the corresponding control element in wr.

The least significant 6 bits in wr control elements modulo the number of elements in the concatenated vectors ws, wt specify the index of the source element. If bit 6 or bit 7 is 1, there will be no copy, but rather the destination element is set to 0.

The operands and results are values in integer data format *df*.

Operation:

VSHFR.B

v ← WR[ws] || WR[wt]

for i in 0 ... WRLEN/8 - 1

k ← $WR[wr]_{8i+5...8i}$ MOD (2*WRLEN/8)

if $WR[wr]_{8i+7...8i+6} \neq 0$ then

$WR[wd]_{8i+7...8i} \leftarrow 0^8$

else

$WR[wd]_{8i+7...8i} \leftarrow v_{8k+7...8k}$

endif

endfor

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

VSLDI.B

Immediate Vector Slide

31 … 26	25 … 21	20 … 16	15 … 11	10 … 6	5 … 0
COP2 010010	ELMC2 10110	wt	ws	wd	df/n

Format: VSLDI.B

VSLDI.B   wd, ws, wt, n

Description: wd ← slide(ws, wt, n)

The instruction function is as the same as the MSA's SLDI.B .

Vector registers wt and ws contain 2-dimensional byte arrays (rectangles) stored row-wise, with as many rows as bytes in integer data format *df*.

The slide instructions manipulate the content of vector registers wt and ws as byte elements, with data format *df* indicating the 2-dimensional byte array layout.

The two source rectangles wt and ws are concatenated horizontally in the order they appear in the syntax, i.e. first wt and then ws. Place a new destination rectangle over ws and then slide it to the left over the concatenation of wt and ws by *n* columns. The result is written to vector wd.

Operation:

VSLDI.B

    v ← WR[wt] || WR[ws]
    for i in 0 ... WRLEN/8 - 1
        $WR[wd]_{8i+7 \ldots 8i} \leftarrow v_{8(i+n)+7 \ldots 8(i+n)}$
    endfor

Exceptions:

Reserved Instruction Exception, MXA Disabled Exception.

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

Appendix A

A.1 COP2 Encoding of rs Field

rs		Bits23...21							
		0	1	2	3	4	5	6	7
	Bits25...24	000	001	010	011	100	101	110	111
0	00	MFC2	β	CFC2	MFHC2	MTC2	β	CTC2	MTHC2
1	01	BC2	BC2EQZ	LWC2	SWC2	θ	BC2NEZ	LDC2	SDC2
2	10	3RFC2	3RC2	3RC2	θ	ELM2RC	ELMC2	ELMC2	ELMC2
3	11	2RFC2	θ	2RC2	θ	θ	4R	θ	θ

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

A.2 COP2 Encoding of Function Field when rs = 3RFC2

3RFC2		Bits2...0							
		0	1	2	3	4	5	6	7
	Bits5...3	000	001	010	011	100	101	110	111
0	000	FEXUPLC	θ	θ	θ	θ	θ	FCMUL.W	θ
1	001	FEXUPRC	θ	θ	θ	θ	θ	θ	θ
2	010	θ	θ	θ	θ	θ	θ	θ	θ
3	011	θ	θ	θ	θ	θ	θ	θ	θ
4	100	θ	θ	θ	θ	θ	θ	θ	θ
5	101	θ	θ	θ	θ	θ	θ	θ	θ
6	110	θ	θ	θ	θ	θ	θ	θ	θ
7	111	θ	θ	θ	θ	θ	θ	θ	θ

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

A.3 COP2 Encoding of Function Field when rs = 2RFC2

2RFC2		Bits20...16				
		0	1	2	3	4...31
Bits5...1	Bits0	00000	00001	00010	00011	00100...11111
00000	0	θ	θ	θ	θ	θ
00000	1	θ	θ	θ	θ	θ
00001	0	θ	θ	θ	θ	θ
00001	1	θ	θ	θ	θ	θ
00010...11111	0...1	θ	θ	θ	θ	θ

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

A.4 COP2 Encoding of Function Field when rs = 3RC2

3RC2 10010		Bits2...0							
		0	1	2	3	4	5	6	7
Bits5...3		000	001	010	011	100	101	110	111
0	000	SLL.V	BMU.V	θ	θ	SRL.V	θ	θ	θ
1	001	SATSS.B	SATSS.H	θ	θ	SATUU.B	SATUU.	θ	θ
2	010	SATUS.B	SATUS.H	θ	θ	θ	θ	θ	θ
3	011	θ	θ	θ	θ	θ	θ	θ	θ
4	100	MOVBT.	θ	θ	θ	MOVBP.	θ	θ	θ
5	101	θ	θ	θ	θ	θ	θ	θ	θ
6	110	θ	θ	θ	θ	θ	θ	θ	θ
7	111	θ	θ	θ	θ	θ	θ	θ	θ

3RC2 10001		Bits2...0							
		0	1	2	3	4	5	6	7
Bits5...3		000	001	010	011	100	101	110	111
0	000	θ	ADDSL.H	ADDSL.W	θ	θ	ADDSR.H	ADDSR.W	θ
1	001	θ	ADDUL.H	ADDUL.W	θ	θ	ADDUR.H	ADDUR.W	θ
2	010	θ	θ	θ	θ	θ	θ	θ	θ
3	011	θ	θ	θ	θ	θ	θ	θ	θ
4	100	θ	SUBSL.H	SUBSL.W	θ	θ	SUBSR.H	SUBSR.W	θ
5	101	θ	SUBUL.H	SUBUL.W	θ	θ	SUBUR.H	SUBUR.W	θ
6	110	θ	MULSL.H	MULSL.W	θ	θ	MULSR.H	MULSR.W	θ
7	111	θ	MULUL.H	MULUL.W	θ	θ	MULUR.H	MULUR.W	θ

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

A.5 COP2 Encoding of Function Field when rs = 2RC2

2RC2 11010		Bits20...16				
		0	1	2	3	4
Bits5...2	Bits1...	00000	00001	00010	00011	00100
0000	00	PCNT.V	BEXT.B	BEXP.B	BEXP.2B	θ
0000	01	θ	BEXT.H	BEXP.H	BEXP.4B	θ
0000	10	θ	BEXT.W	BEXP.W	θ	θ
0000	11	θ	θ	θ	θ	θ
0001...1100	00...11	θ	θ	θ	θ	θ

2RC2 11010		Bits20...16				
		5	6	7	8	9...31
Bits5...2	Bits1...	00101	00110	00111	01000	01001...11111
0000	00	θ	θ	θ	θ	θ
0000	01	ACCUL.H	ACCUR.H	ACCSL.H	ACCSR.H	θ
0000	10	ACCUL.W	ACCUR.W	ACCSL.W	ACCSR.W	θ
0000	11	θ	θ	θ	θ	θ
0001...1100	00...11	θ	θ	θ	θ	θ

2RC2 11010		
Bits5...2	Bits1...	00000
1101	00...11	θ
1110	00	SLLI.V
1110	01...11	θ
1111	00	SRLI.V
1111	01...11	θ

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

A.6 COP2 Encoding of df/n field when rs = ELMC

ELMC2	Bits25...21		
	21	22	23
Bits5...0	10101	10110	10111
00nnnnn	θ	VSLDI.B	θ
100nnn	θ	θ	LDINSSU2.W
1100nn	LDINS.W	STEXT.W	LDINSU.W
11100n	LDINS.D	STEXT.D	LDINSU.D
1111nn	LDINSS.W	LDINSSU.W	θ
111110	×	×	θ
111111	×	×	θ

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

A.7 COP2 Encoding of df/n field when rs = ELM2RC2

ELM2RC2	Bits20...16				
10100	0	1	2	3	4...31
Bits5...0	00000	00001	00010	00011	00100...11111
00nnnnn	×	θ	θ	θ	θ
100nnn	θ	θ	θ	θ	θ
1100nn	θ	STEXTU.W	θ	θ	θ
11100n	θ	STEXTU.D	θ	θ	θ
1111nn	θ	θ	θ	θ	θ
111110	θ	θ	θ	θ	θ
111111	θ	θ	θ	θ	θ
00mmnn	MOVE.W	×	×	×	×

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.

君正 Ingenic

Revision History

Revision	Date	Description
00.00	July 21, 2016	• Initial version
00.01	August 1, 2016	• Fixed some typos in the operation • Modify the name prefix floating-point complex operation into “FC” • Add load insert instruction LDINSSU2 • Delete instruction VSLDI.B for it’s function is the same as the SLDI.B • Add new instructions for the speech • Add new instructions for the image • Add CTC2/CFC2 instructions • Add description about XBurst half single FP • Add description about FP complex data and operation
00.02	August 1, 2016	• internal release
00.03	August 5, 2016	• Recover LDINS.df, STEXT.df
00.04	August 8, 2016	• Fixed some typos • Update LDINSU.df • Delete FCMADD.df, FCMSUB.df, FCUMSUB.df
00.05	August 9, 2016	• Fixed some typos
00.06	September 21, 2016	• Fixed some error descriptions • Named MIPS eXtended Architecture (MXA)
00.07	October 26, 2016	• Fixed the error function saturate_sigend • Fixed some typos L-->R • Change SATS.B.H/SATS.H.W into SATSS.B/SATSS.H • Change SATU.B.H/SATU.H.W into SATUU.B/SATUU.H • Add SATUS.B/SATUS.H
00.08	December 30,2016	• Add Programming model context and instruction detailed descriptions • Delete unused functions: complex_sub, complex_add, complex_neg • Fixed some description errors
00.09	June 2,2017	• Change the document name

Copyright © 2005-2016 Ingenic Semiconductor Co., Ltd. All rights reserved.