NEC

SX-Aurora TSUBASA

Vector Engine Assembly Language Reference Manual

Proprietary Notice

Related Documents

- SX-Aurora TSUBASA Architecture Manual

Remarks

Contents

List of tables

Chapter1 Assembler Operation

This chapter describes Vector Engine assembler command-line syntax.

1.1 Command-line Syntax

nas [options] file ...

Chapter2 Assembler Options

This chapter describes Vector Engine assembler options which are supported by Vector Engine. The other options are the same as the GNU assembler options.

2.1 Options

-march=*arch*

This option specifies the target architecture. The assembler will issue an error message if an attempt is made to assemble an instruction which will not execute on the target architecture. The following architecture names are recognized:'ve1'(default), 've3'. When specifying two or more the same option, the last option becomes effective.

-mabi=*ver*

This option specifies which ABI version the produced object files should conform to. The following values are recognized:'1'(default), '2'. ‘2’ available only on ve3 or later. When specifying two or more the same option, the last option becomes effective.

-mfp16-type=*type*

This option specifies type of the half-precision floating-point to set for the produced object file. The following values are recognized:’none’(default), 'ieee'(binary16), 'bfloat'(bfloat16). ‘ieee’ and ‘bfloat’ available only on ve3 or later. When specifying two or more the same option, the last option becomes effective.

Chapter3 Pseudo-Instructions

This chapter describes Vector Engine assembler pseudo-instructions which are supported by Vector Engine.

3.1 Section definition pseudo-instructions

3.1.1 .section

Format:

.section *name*[, "*flags*"[, *@type*[, *flag_specific_arguments*]]]

3.1.2 . text

Format:

.text [*subsection*]

3.1.3 . data

Format:

.data [*subsection*]

3.1.4 . bss

Format:

.bss [.*subsection*]

3.2 Data Definition Pseudo-Instructions

3.2.1 .2byte, .4byte, 8byte

Format:

.2byte EXPRESSIONS

.4byte EXPRESSIONS

.8byte EXPERSSIONS

Description:

These directives write unaligned 2, 4 or 8 byte values to the output section.

3.2.2 .byte

Format:

.byte EXPRESSIONS

Description:

.byte expects zero or more expressions, separated by commas. Each expression is assembled into the next byte.

3.2.3 .short

Format:

.**short** EXPRESSIONS

Description:

Expect zero or more EXPRESSIONS, of any section, separated by commas. For each expression, emit a number that, at run time, is the value of that expression. The byte order is little endian and bit size of the number is 16 bits (2 bytes).

3.2.4 .word

Format:

.**word** EXPRESSIONS

Description:

Expect zero or more EXPRESSIONS, of any section, separated by commas. For each expression, emit a number that, at run time, is the value of that expression. The byte order is little endian and bit size of the number is 32 bits (4 bytes).

3.2.5 .int

Format:

.**int** EXPRESSIONS

Description:

Expect zero or more EXPRESSIONS, of any section, separated by commas. For each expression, emit a number that, at run time, is the value of that expression. The byte order is little endian and bit size of the number is 32 bits (4 bytes).

3.2.6 .long

Format:

.**long** EXPRESSIONS

Description:

Expect zero or more EXPRESSIONS, of any section, separated by commas. For each expression, emit a number that, at run time, is the value of that expression. The byte order is little endian and bit size of the number is 64 bits (8 bytes).

3.2.7 .quad

Format:

.**quad** EXPRESSIONS

Description:

Expect zero or more EXPRESSIONS, of any section, separated by commas. For each expression, emit a number that, at run time, is the value of that expression. The byte order is little endian and bit size of the number is 64 bits (8 bytes).

3.2.8 .llong

Format:

.**llong** EXPRESSIONS

Description:

Expect zero or more EXPRESSIONS, of any section, separated by commas. For each expression, emit a number that, at run time, is the value of that expression. The byte order is little endian and bit size of the number is 64 bits (8 bytes).

3.3 Miscellaneous Pseudo-Instructions

3.3.1 .file

Format:

.**file** "*string*"

3.3.2 .ident

Format:

.**ident** "*string*"

Description:

string is emitted to the ".comment" section.

3.3.3 .arch

Format:

.**arch** *arch*

Description:

This pseudo-instruction specifies the target architecture. Valid values for *arch* are the same as for the -march option. When specifying both the option and the pseudo-instruction, the pseudo-instruction becomes effective. When specifying two or more

the same pseudo-instruction, the last pseudo-instruction becomes effective.

3.3.4 .abi

Format:

.abi *ver*

Description:

This pseudo-instruction specifies which ABI version the produced object files should conform to. Valid values for *ver* are the same as for the -mabi option. When specifying both the option and the pseudo-instruction, the pseudo-instruction becomes effective. When specifying two or more the same pseudo-instruction, the last pseudo-instruction becomes effective.

3.3.5 .fp16_type

Format:

.fp16_type *type*

Description:

This option specifies type of the half-precision floating-point to set for the produced object file. Valid values for *type* are the same as for the -mfp16-type option. When specifying both the option and the pseudo-instruction, the pseudo-instruction becomes effective. When specifying two or more the same pseudo-instruction, the last pseudo-instruction becomes effective.

Chapter4 Assembler Syntax

4.1 Statement Syntax

A statement of the Vector Engine assembly language is represented as the following format.

[*label*] <*instruction*> [<*operand*>[, <*operand*> …]]

4.2 Operand Description Format

The register, immediate values, and effective address can only be described in the operand.

4.2.1 Register Notations

Table 3-1 shows the register notations used in the Vector Engine assembly language. See also Chapter 3 of SX-Aurora TSUBASA Architecture Manual for details.

Table 3-1 Register Notations

Register	Syntax
Scalar register	%s*n* (n=0 ? 63)
Vector register	%v*n* (n=0 ? 63)
Vector mask register	%vm*n* (n=0 ? 15)
Vector index register	%vix

Table 3-2 Register Aliases

Alias (actual register/immediate)	Syntax
Stack pointer (%s11)	%sp
Frame pointer (%s9)	%fp
Stack limit (%s8)	%sl
Link register (%s10)	%lr
Thread pointer (%s14)	%tp
Outer register(%s12)	%outer
Info area register(%s17)	%info
Global offset table register(%s15)	%got
Procedure linkage table register(%s16)	%plt
User clock counter (0)	%usrcc
Program status word (1)	%psw
Store address register (2)	%sar
Performance monitor mode register (7)	%pmmr

Performance monitor configuration register (8-11)	%pmcr*m* (m=0-3)
Performance monitor counter (16-30)	%pmc*n* (n=0-14)

4.2.2 Effective Address Notations

ASX are address syllable for RM and CF formats. The following are address syllable formats. See also Chapter 5 of SX-Aurora TSUBASA Architecture Manual for details.

(1) **disp**

(2) **disp** (, *base*)

(3) **disp** (*index*)

(4) **disp** (*index*, *base*)

(5) (, *base*)

(6) (*index*)

(7) (*index*, *base*)

base specifies a scalar register; *index* specifies a scalar register or immediate data in the range of -64 to 63; and **disp** specifies a label name or absolute value.
AS format is the same as ASX but it can not accept *index*. Therefore, the following are acceptable.

(1) **disp**

(2) **disp**(,*base*)

(3) (,*base*)

When it is RRM format, a comma is omitted as follows.

(1) **disp**

(2) **disp**(*base*)

(3) (*base*)

HM is address syllable for RRM format. The following are address syllable formats.

(1) *base*

(2) (*base*)

(3) **disp**(*base*)

base specifies a scalar register and **disp** specifies an immediate value.

4.2.3 Immediate value Notations

Table 3-3 shows immediate value notations. See also Chapter 5 of SX-Aurora TSUBASA Architecture Manual for details.

Table 3-3 Immediate Value Notations

Manual Description	Syntax	Meaning
I	Expression including only integer constants D (octal, decimal or hexadecimal constants)	Immediate value to be set in Ry of RR-type instruction. The lower 7 bits of the integer constant specified by the expression are set.
N	Expression including only integer constants (octal, decimal or hexadecimal constants)	
M	(m)0 or (m)1 M: Integer in the range 0-63	When (m)0 is specified, 64-bit immediate value having m zeros from left and (64-m) ones. When (m)1 is specified, 64-bit immediate value having m ones from left and (64-m) zeros.
Z		Immediate 0 (zero) value if whatever immediate value is specified
D	Expression including only a label name or integer constants (octal, decimal or hexadecimal constants).	A label name or absolute value to be set in D field of RM-type instruction.

4.2.4 Prefix Notations

Table 3-4 Prefix Notations

Prefix	Meaning
%hi(*label*)	Get upper 32-bit of the address '*label*' or the immediate value
%lo(*label*)	Get lower 32-bit of the address '*label*' or the immediate value

Note It'll be expected to eliminate this Pseudo-Instruction from now on.

4.2.5 Suffix Notations

Table 3-5 Suffix Notations

Suffix	Meaning
label **@hi**	Get upper 32-bit of the address '*label*' or the immediate value
label **@lo**	Get lower 32-bit of the address '*label*' or the immediate value
label **@pc_hi**	Get upper 32-bit of the relative address to '*label*'
label **@pc_lo**	Get lower 32-bit of the relative address to '*label*'
label **@got_hi**	Get upper 32-bit of the address of GOT entry of '*label*'
label **@got_lo**	Get lower 32-bit of the address of GOT entry of '*label*'
label **@gotoff_hi**	Get upper 32-bit of the relative address from GOT to '*label*'
label **@gotoff_lo**	Get lower 32-bit of the relative address from GOT to '*label*'
label **@plt_hi**	Get upper 32-bit of the address of PLT entry of '*label*'
label **@plt_lo**	Get lower 32-bit of the address of PLT entry of '*label*'

4.3 Instruction Mnemonics

This section describes the syntax of the instruction mnemonics.

4.3.1 List of Notations

Table 3-6 List notations used in the descriptions of the syntax of instruction mnemonics.

See also Chapter 5 of SX-Aurora TSUBASA Architecture Manual for details.

Table 3-6 List of Notations (operands)

Notation	Description
AS	Address syllable
ASX	Address syllable
HM	Address syllable for host memory
%sx, %sy, %sz	Scalar register
%vx, %vy, %vz	Vector register
%vm	Vector mask register

%vix	Vector index register
I	Immediate value (used for arithmetic operations) in the range from -64 to 63.
N	Immediate value (used for the number of shifts, element number, interelement distance, etc) in the range from 0 to 127.
M	(m)0 or (m)1 format is specified, m is an integer in the range from 0 to 63.
Z	Immediate value 0 (zero)
D	Expression including only a label name or integer constants Absolute value in the range from 0 to 4294967295.

Some instructions can change their function when their mnemonics are followed by suffixes as Table 3-7.

Table 3-7 List of Notation (Mnemonic suffix)

Suffix	Description
df	Data format l: 64 bit integer w: 32 bit integer d: 64 bit floating point s: 32 bit floating point h: 16 bit floating point (available only on ve3 or later)
ex	Extension sx: Sign extension zx or *NONE*: Zero extension
bp	Branch Prediction *NONE*: No branch prediction nt: Not taken t: Taken
cf	Condition Field af: Always false gt: Greater than lt: Less than ne: Not equal eq: Equal ge: Greater than or equal le: Less than or equal num: Is number nan: Is NaN (Not a number) gtnan: Greater than or NaN ltnan: Less then or NaN nenan: Not equal or NaN eqnan: Equal or NaN genan: Greater than equal or NaN

	lenan: Greater than equal or NaN at or *NONE*: Always true
rd	Rounding Mode *NONE*: According to PSW rz: Round toward Zero rp: Round toward Plus infinity rm: Round toward Minus infinity rn: Round to Nearest (ties to Even) ra: Round to Nearest (ties to Away)
nc	- ve1: Loaded data is likely to be released from the LLC. - ve3: Loaded data is not cached in the L3 cache.
ot	Overtake
nex	No Exception
pos	Position fst: First element lst: Last element
cp	upper/lower 32bits of Sy operand can be copied to the opposite 32bit. (available only on ve3 or later)
async	hardware returns a register value before the hardware synchronization. (available only on ve3 or later)

Table 3-8 List of notation (Description)

Operator	Description
M(A,B)	B-byte memory contents or location at the effective address given by the contents of A. B can be omitted, and in that case B is regarded as 1. Operation address, calculated by each fields of an instruction.
EA	Operation address, calculated by each fields of an instruction.
A[i:j]	from bit i to bit j of register A
A ← B	Storing (moving) of the contents of B into A.
Sx	Immediate value or S register designated by x field of instruction word.
Sy	Immediate value or S register designated by y
Sz	Immediate value or S register designated by z
mod(A, B)	The remainder of A divided by B
sext(A, B)	B-bit value is generated by expanding sign bit (Most significant bit) of A.
cond(A, B, C)	The result of comparison B and C in A condition. C can be omitted and in this case C is handled as 0. Refer to the chapter 5 for system interpretation of comparison condition
max(A, B)	Maximum value of A and B.
min(A, B)	Minimum value of A and B.

4.3.2 Instruction Set

This section lists instruction code/format, assembler mnemonic syntax and description of the instruction mnemonics. See also Chapter 8 of SX-Aurora TSUBASA Architecture Manual for details.

4.3.2.1 Transferring Instructions

Instruction	Assembler Mnemonic Syntax	Description
LEA 06 / RM	**lea** %sx, ASX **lea.sl** %sx, ASX	Load Effective Address
LDS 01 / RM	**ld** %sx, ASX	Load S
LDU 02 / RM	**ldu** %sx, ASX	Load S Upper
LDL 03 / RM	**ldl**[.*ex*] %sx, ASX	Load S Lower
LD2B 04 / RM	**ld2b**[.*ex*] %sx, ASX	Load 2B
LD1B 05 / RM	**ld1b**[.*ex*] %sx, ASX	Load 1B
STS 11 / RM	**st** %sx, ASX	Store S
STU 12 / RM	**stu** %sx, ASX	Store S Upper
STL 13 / RM	**stl** %sx,ASX	Store S Lower
ST2B 14 / RM	**st2b** %sx, ASX	Store 2B
ST1B 15 / RM	**st1b** %sx, ASX	Store 1B
DLDS 09 / RM	**dld** %sx, ASX	Dismissable Load S
DLDU 0A / RM	**dldu** %sx, ASX	Dismissable Load Upper
DLDL 0B / RM	**dldl**[.*ex*] %sx, ASX	Dismissable Load Lower

PFCH 0C / RM	**pfch** ASX	Pre Fetch
CMOV 3B / RR	**cmov.***df*[**.***cf*] %sx, {%sz | M}, {%sy | I}	Conditional Move

4.3.2.2 Fixed-Point Arithmetic Operation Instructions

Instruction	Assembler Mnemonic Syntax	Description
ADDI 07 / RM	**addi** %sx, {%sy | I}, D	Add immediate Available only on ve3 or later.
ADD 48 / RR	**addu.l** %sx, {%sy | I}, {%sz | M} **addu.w** %sx, {%sy | I}, {%sz | M}	Add
ADS 4A / RR	**adds.w**[**.***ex*] %sx, {%sy | I}, {%sz | M}	Add Single
ADX 59 / RR	**adds.l** %sx, {%sy | I}, {%sz | M}	Add
SUB 58 / RR	**subu.l** %sx, {%sy | I}, {%sz | M} **subu.w** %sx, {%sy | I}, {%sz | M}	Subtract
SBS 5A / RR	**subs.w**[**.***ex*] %sx, {%sy | I}, {%sz | M}	Subtract Single
SBX 5B / RR	**subs.l** %sx, {%sy | I}, {%sz | M}	Subtract
MPY 49 / RR	**mulu.l** %sx, {%sy | I}, {%sz | M} **mulu.w** %sx, {%sy | I}, {%sz | M}	Multiply
MPS 4B / RR	**muls.w**[**.***ex*] %sx, {%sy | I}, {%sz | M}	Multiply Single
MPX 6E / RR	**muls.l** %sx, {%sy | I}, {%sz | M}	Multiply
MPD 6B / RR	**muls.l.w** %sx, {%sy | I}, {%sz | M}	Multiply
DIV 6F / RR	**divu.l %sx, {%sy | I}, {%sz | M}** **divu.w %sx, {%sy | I}, {%sz | M}** **modu.l** %sx, {%sy | I}, {%sz | M} **modu.w** %sx, {%sy | I}, {%sz | M}	Divde Modulo Available only on ve3 or later.
DVS	**divs.w[.***ex***] %sx, {%sy | I}, {%sz | M}**	Divide Single

7B / RR	**mods.w[.*ex*]** %sx, {%sy | I}, {%sz | M}	Modulo Single Available only on ve3 or later.
DVX	**divs.l %sx, {%sy | I}, {%sz | M}**	Divide
7F / RR	**mods.l** %sx, {%sy | I}, {%sz | M}	Modulo Available only on ve3 or later.
CMP 55 / RR	**cmpu.l** %sx, {%sy | I}, {%sz | M} **cmpu.w** %sx, {%sy | I}, {%sz | M}	Compare
CPS 7A / RR	**cmps.w**[.*ex*] %sx, {%sy | I}, {%sz | M}	Compare Single
CPX 6A / RR	**cmps.l** %sx, {%sy | I}, {%sz | M}	Compare
CMS 78 / RR	**maxs.w**[.*ex*] %sx, {%sy | I}, {%sz | M} **mins.w**[.*ex*] %sx, {%sy | I}, {%sz | M}	Compare and Select Maximum/Mininum Single
CMX 68 / RR	**maxs.l** %sx, {%sy | I}, {%sz | M} **mins.l** %sx, {%sy | I}, {%sz | M}	Compare and Select Maximum/Minimum

4.3.2.3 Logical Arithmetic Operation Instructions

Instruction	Assembler Mnemonic Syntax	Description
AND 44 / RR	**and** %sx, {%sy | I}, {%sz | M}	AND
OR 45 / RR	**or** %sx, {%sy | I}, {%sz | M}	OR
XOR 46 / RR	**xor** %sx, {%sy | I}, {%sz | M}	Exclusive OR
EQV 47 / RR	**eqv** %sx, {%sy | I}, {%sz | M}	Equivalence
NND 54 / RR	**nnd** %sx, {%sy | I}, {%sz | M}	Negate AND
MRG 56 / RR	**mrg** %sx, {%sy | I}, {%sz | M}	Merge
LDZ 67 / RR	**ldz** %sx, {%sz | M}	Leading Zero Count
PCNT	**pcnt** %sx, {%sz | M}	Population Count

38 / RR		
BRV 39 / RR	**brv** %sx, {%sz | M}	Bit Reverse

4.3.2.4 Shift Instructions

Instruction	Assembler Mnemonic Syntax	Description
SLL 65 / RR	**sll** %sx, {%sz | M}, {%sy | N}	Shift Left Logical
SLD 64 / RR	**sld** %sx, {%sz | M}, {%sy | N}	Shift Left Double
SRL 75 / RR	**srl** %sx, {%sz | M}, {%sy | N}	Shift Right Logical
SRD 74 / RR	**srd** %sx, {%sz | M}, {%sy | N}	Shift Right Double
SLA 66 / RR	**sla.w**[.*ex*] %sx, {%sz | M}, {%sy | N}	Shift Left Arithmetic
SLAX 57 / RR	**sla.l** %sx, {%sz | M}, {%sy | N}	Shift Left Arithmetic
SRA 76 / RR	**sra.w**[.*ex*] %sx, {%sz | M}, {%sy | N}	Shift Right Arithmetic
SRAX 77 / RR	**sra.l** %sx, {%sz | M}, {%sy | N}	Shift Right Arithmetic

4.3.2.5 Floating-point Arithmetic Operation Instructions

Instruction	Assembler Mnemonic Syntax	Description
FAD	**fadd.d** %sx, {%sy | I}, {%sz | M}	Floating Add
4C / RR	**fadd.s** %sx, {%sy | I}, {%sz | M}	
FSB	**fsub.d** %sx, {%sy | I}, {%sz | M}	Floating Subtract
5C / RR	**fsub.s** %sx, {%sy | I}, {%sz | M}	
FMP	**fmul.d** %sx, {%sy | I}, {%sz | M}	Floating Multiply
4D / RR	**fmul.s** %sx, {%sy | I}, {%sz | M}	
FDV	**fdiv.d** %sx, {%sy | I}, {%sz | M}	Floating Divide
5D / RR	**fdiv.s** %sx, {%sy | I}, {%sz | M}	
FCP	**fcmp.d** %sx, {%sy | I}, {%sz | M}	Floating Compare
7E / RR	**fcmp.s** %sx, {%sy | I}, {%sz | M}	
FCM	**fmax.d** %sx, {%sy | I}, {%sz | M}	Floating Compare

3E / RR	**fmax.s** %sx, {%sy | I}, {%sz | M} **fmin.d** %sx, {%sy | I}, {%sz | M} **fmin.s** %sx, {%sy | I}, {%sz | M}	and Select Maximum/Minimum
FAQ 6C / RW	**fadd.q** %sx, {%sy | I}, {%sz | M}	Floating Add Quadruple
FSQ 7C / RW	**fsub.q** %sx, {%sy | I}, {%sz | M}	Floating Subtract Quadruple
FMQ 6D / RW	**fmul.q** %sx, {%sy | I}, {%sz | M}	Floating Multiply Quadruple
FCQ 7D / RW	**fcmp.q** %sx, {%sy | I}, {%sz | M}	Floating Compare Quadruple
FIX 4E / RR	**cvt.w.d**[.*ex*][.*rd*] %sx, {%sy | I} **cvt.w.s**[.*ex*][.*rd*] %sx, {%sy | I}	Convert to Fixed Point
FIXX 4F / RR	**cvt.l.d**[.*rd*] %sx, {%sy | I}	Convert to Fixed Point
FLT 5E / RR	**cvt.d.w** %sx, {%sy | I} **cvt.s.w** %sx, {%sy | I}	Convert to Floating Point
FLTX 5F / RR	**cvt.d.l** %sx, {%sy | I}	Convert to Floating Point
CVD 0F / RW	**cvt.d.s** %sx, {%sy | I} **cvt.d.q** %sx, {%sy | I}	Convert to Double-format
CVS 1F / RW	**cvt.s.d** %sx, {%sy | I} **cvt.s.q** %sx, {%sy | I} **cvt.s.h** %sx, {%sy | I}	Convert to Single-format cvt.s.h available only on ve3 or later.
CVQ 2D / RW	**cvt.q.d** %sx, {%sy | I} **cvt.q.s** %sx, {%sy | I}	Convert to Quadruple-format
CVH 1E / RR	**cvt.h.s** %sx, {%sy | I}	Convert to IEEE fp16-format Available only on ve3 or later.

4.3.2.6 Branch Instructions

Instruction	Assembler Mnemonic Syntax	Description
BC 19 / CF	**b**[*cf*].**l**[.*bp*] [{%sy | I},] AS	Branch on Condition If *cf* is “af” or “at”, {%sy | I} cannot

		be specified. If *cf* is “at”, *cf* can be omitted.
BCS 1B / CF	**b**[*cf*]**.w**[*.bp*] [{%sy | I},] AS	Branch on Condition Single If *cf* is “af” or “at”, {%sy | I} cannot be specified. If *cf* is “at”, *cf* can be omitted.
BCF 1C / CF	**b**[*cf*]**.d**[*.bp*] [{%sy | I},] AS **b**[*cf*]**.s**[*.bp*] [{%sy | I},] AS	Branch on Condition Floating Point If *cf* is “af” or “at”, {%sy | I} cannot be specified. If *cf* is “at”, *cf* can be omitted.
BCR 18 / CF	**br**[*cf*]**.l**[*.bp*] {%sy | I}, {%sz | Z}, AS **br**[*cf*]**.w**[*.bp*] {%sy | I}, {%sz | Z}, AS **br**[*cf*]**.d**[*.bp*] {%sy | I}, {%sz | Z}, AS **br**[*cf*]**.s**[*.bp*] {%sy | I}, {%sz | Z}, AS	Branch on Condition Relative If *cf* is “af” or “at”, both {%sy | I} and {%sz | Z} cannot be specified. If *cf* is “at”, *cf* can be omitted. "AS" is disp only. If "disp" of “AS” is label and suffix is set in "disp" of "AS", suffix is ignored.
BSIC 08 / RM	**bsic** %sx, ASX	Branch and Save IC

4.3.2.7 Vector Transfer Instructions

Instruction	Assembler Mnemonic Syntax	Description
VLD 81 / RVM	**vld**[*.nc*] {%vx | %vix}, {%sy | I}, {%sz | Z} **vdld**[*.nc*] {%vx | %vix}, {%sy | I}, {%sz | Z}	Vector Load vdld available only on ve3 or later. vdld does not detect a missing page or missing

		space exception.
VLDU 82 / RVM	**vldu**[.*nc*] {%vx | %vix}, {%sy | I}, {%sz | Z} **vdldu**[.*nc*] {%vx | %vix}, {%sy | I}, {%sz | Z} **pvldu**[.*nc*] {%vx | %vix}, {%sy | I}, {%sz | Z} **pvdldu**[.*nc*] {%vx | %vix}, {%sy | I}, {%sz | Z}	Vector Load Upper vdldu, pvldu and pvdldu available only on ve3 or later. vdldu and pvdldu do not detect a missing page or missing space exception.
VLDL 83 / RVM	**vldl**[.*ex*][.*nc*] {%vx | %vix}, {%sy | I}, {%sz | Z} **vdldl**[.*ex*][.*nc*] {%vx | %vix}, {%sy | I}, {%sz | Z}	Vector Load Lower vdldl available only on ve3 or later. vdldl does not detect a missing page or missing space exception.
VLD2D C1 / RVM	**vld2d**[.*nc*] {%vx | %vix}, {%sy | I}, {%sz | Z} **vdld2d**[.*nc*] {%vx | %vix}, {%sy | I}, {%sz | Z}	Vector Load 2D vdld2d available only on ve3 or later. vdld2d does not detect a missing page or missing space exception.
VLDU2D C2 / RVM	**vldu2d**[.*nc*] {%vx | %vix}, {%sy | I}, {%sz | Z} **vdldu2d**[.*nc*] {%vx | %vix}, {%sy | I}, {%sz | Z}	Vector Load Upper 2D vdldu2d available only on ve3 or later. vdldu2d does not detect a missing page or missing space exception.
VLDL2D C3 / RVM	**vldl2d**[.*ex*][.*nc*] {%vx | %vix}, {%sy | I}, {%sz | Z} **vdldl2d**[.*ex*][.*nc*] {%vx | %vix}, {%sy | I}, {%sz | Z}	Vector Load Lower 2D vdldl2d available only on ve3 or later. vdldl2d does not detect a missing page or missing space exception.
VST	**vst**[.*nc*][.*ot*] {%vx | %vix}, {%sy | I}, {%sz | Z} [, %vm]	Vector Store

91 / RVM		
VSTU 92 / RVM	**vstu**[.*nc*][.*ot*] {%vx | %vix}, {%sy | I}, {%sz | Z} [, %vm] **pvstu**[.*nc*][.*ot*] {%vx | %vix}, {%sy | I}, {%sz | Z} [, %vm]	Vector Store Upper pvstu available only on ve3 or later.
VSTL 93 / RVM	**vstl**[.*nc*][.*ot*] {%vx | %vix}, {%sy | I}, {%sz | Z} [, %vm]	Vector Store Lower
VST2B 90 / RVM	**vst2bl**[.nc][.ot] {%vx | %vix}, {%sy | I}, {%sz | Z} [, %vm] **vst2bu**[.nc][.ot] {%vx | %vix}, {%sy | I}, {%sz | Z} [, %vm] **pvst2bl**[.nc][.ot] {%vx | %vix}, {%sy | I}, {%sz | Z} [, %vm] **pvst2bu**[.nc][.ot] {%vx | %vix}, {%sy | I}, {%sz | Z} [, %vm]	Vector store 2B Available only on ve3 or later.
VST2D D1 / RVM	**vst2d**[.*nc*][.*ot*] {%vx | %vix}, {%sy | I}, {%sz | Z} [, %vm]	Vector Store 2D
VSTU2D D2 / RVM	**vstu2d**[.*nc*][.*ot*] {%vx | %vix}, {%sy | I}, {%sz | Z} [, %vm]	Vector Store Upper 2D
VSTL2D D3 / RVM	**vstl2d**[.*nc*][.*ot*] {%vx | %vix}, {%sy | I}, {%sz | Z} [, %vm]	Vector Store Lower 2D
VLD2B 80 / RVM	**vld2bl**[.ex][.*nc*] {%vx | %vix}, {%sy | I}, {%sz | Z} **vld2bu**[.*nc*] {%vx | %vix}, {%sy | I}, {%sz | Z} **pvld2bl**[.*ex*][.*nc*] {%vx | %vix}, {%sy | I}, {%sz | Z} **pvld2bu**[.*nc*] {%vx | %vix}, {%sy | I}, {%sz | Z} **vdld2bl**[.*ex*][.*nc*] {%vx | %vix}, {%sy | I}, {%sz | Z} **vdld2bu**[.*nc*] {%vx | %vix}, {%sy | I}, {%sz | Z} **pvdld2bl**[.*ex*][.*nc]* {%vx | %vix}, {%sy | I}, {%sz | Z} **pvdld2bu**[.*nc*] {%vx | %vix}, {%sy | I}, {%sz | Z}	Vector load 2B Available only on ve3 or later. vdld2bl, vdld2bu, pvdld2bl and pvdld2bu do not detect a missing page or missing space exception.
PFCHV 80 / RVM	**pfchv**[.*nc*] {%sy | I}, {%sz | Z}	Pre Fetch Vector
LSV 8E / RR	**lsv** {%vx | %vix}({%sy |N}), {%sz | M}	Load S to V

LVS 9E / RR	**lvs** %sx, {%vx | %vix}({%sy | N})	Load V to S
LVM B7 / RR	**lvm** %vmx, {%sy | N}, {%sz | M}	Load VM N = 0 - 3
SVM A7 / RR	**svm** %sx, %vmz, {%sy | N}	Save VM N = 0 - 3
VBRD 8C / RV	**vbrd**[.*cp*] {%vx | %vix}, {%sy | I} [, %vm] **vbrdl**[.*cp*] {%vx | %vix}, {%sy | I} [, %vm] **vbrdu**[.*cp*] {%vx | %vix}, {%sy | I} [, %vm] **pvbrd**[.*cp*] {%vx | %vix}, {%sy | I} [, %vm]	Vector Broadcast *cp* available only on ve3 or later.
VMV 9C / RV	**vmv** {%vx | %vix}, {%sy | N}, {%vz | %vix} [, %vm]	Vector Move
VEST F9 / RV	**vestl.w** {%vx | %vix}, {%vy | %vix}, {%vz | %vix} [, %vm] **vestl.l** {%vx | %vix}, {%vy | %vix}, {%vz | %vix} [, %vm] **vestr.w** {%vx | %vix}, {%vy | %vix}, {%vz | %vix} [, %vm] **vestr.l** {%vx | %vix}, {%vy | %vix}, {%vz | %vix} [, %vm]	Vector element-wise shift 8B/4B Available only on ve3 or later.

4.3.2.8 Vector Fixed-Point Arithmetic Operation Instructions

Instruction	Assembler Mnemonic Syntax	Description
VADD C8 / RV	**vaddu.***df*[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvaddu.lo**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvaddu.up**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvaddu**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	Vector Add *cp* available only on ve3 or later. If 2nd operand is %sy or I, *cp* is available.
VADS CA / RV	**vadds.w**[.*ex*][.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvadds.lo**[.*ex*][.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvadds.up**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvadds**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy |	Vector Add Single *cp* available only on ve3 or later. If 2nd operand is %sy or I, *cp* is available.

	I}, {%vz | %vix} [, %vm]	
VADX 8B / RV	**vadds.l** {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	Vector Add
VSUB D8 / RV	**vsubu.***df*[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvsubu.lo**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvsubu.up**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvsubu**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	Vector Subtract *cp* available only on ve3 or later. If 2nd operand is %sy or I, *cp* is available.
VSBS DA / RV	**vsubs.w**[.*ex*][.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvsubs.lo**[.*ex*][.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvsubs.up**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvsubs**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	Vector Subtract Single *cp* available only on ve3 or later. If 2nd operand is %sy or I, *cp* is available.
VSBX 9B / RV	**vsubs.l** {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	Vector Subtract
VMPY C9 / RV	**vmulu.***df* {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	Vector Multiply
VMPS CB / RV	**vmuls.w**[.*ex*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	Vector Multiply Single
VMPX DB / RV	**vmuls.l** {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	Vector Multiply
VMPD D9 / RV	**vmuls.l.w** {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [,%vm]	Vector Multiply
VDIV E9 / RV	**vdivu.***df* {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix | %sy | I} [, %vm] **vmodu.***df* {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix | %sy | I} [, %vm]	Vector Divide vmodu available only on ve3 or later.
VDVS EB / RV	**vdivs.w**[.*ex*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix | %sy | I} [,%vm] **vmods.w**[.*ex*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix | %sy | I} [,%vm]	Vector Divide Single vmods.w available only on ve3 or later.
VDVX FB / RV	**vdivs.l** {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix | %sy | I} [, %vm] **vmods.l** {%vx | %vix}, {%vy | %vix | %sy | I},	Vector Divide vmods.l available

	{%vz | %vix | %sy | I} [, %vm]	only on ve3 or later.
VCMP B9 / RV	**vcmpu**.*df*[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvcmpu.lo**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvcmpu.up**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvcmpu**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	Vector Compare *cp* available only on ve3 or later. If 2nd operand is %sy or I, *cp* is available.
VCPS FA / RV	**vcmps.w**[.*ex*][.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvcmps.lo**[.*ex*][.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvcmps.up**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvcmps**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	Vector Compare Single *cp* available only on ve3 or later. If 2nd operand is %sy or I, *cp* is available.
VCPX BA / RV	**vcmps.l** {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	Vector Compare
VCMS 8A / RV	**vmaxs.w**[.*ex*][.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvmaxs.lo**[.*ex*][.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvmaxs.up**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvmaxs**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **vmins.w**[.*ex*][.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvmins.lo**[.*ex*][.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvmins.up**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **pvmins**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	Vector Compare and Select Maximum/Minimum Single *cp* available only on ve3 or later. If 2nd operand is %sy or I, *cp* is available.
VCMX 9A / RV	**vmaxs.l** {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **vmins.l** {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	Vector Compare and Select Maximum/Minimum

4.3.2.9 Vector Logical Arithmetic Operation Instructions

Instruction	Assembler Mnemonic Syntax	Description
VAND C4 / RV	**vand**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | M}, {%vz | %vix} [, %vm] **pvand.lo**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | M}, {%vz | %vix} [, %vm] **pvand.up**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | M}, {%vz | %vix} [, %vm] **pvand**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | M}, {%vz | %vix} [, %vm]	Vector AND *cp* available only on ve3 or later. If 2nd operand is %sy or M, *cp* is available.
VOR C5 / RV	**vor**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | M}, {%vz | %vix} [, %vm] **pvor.lo**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | M}, {%vz | %vix} [, %vm] **pvor.up**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | M}, {%vz | %vix} [, %vm] **pvor**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | M}, {%vz | %vix} [, %vm]	Vector OR *cp* available only on ve3 or later. If 2nd operand is %sy or M, *cp* is available.
VXOR C6 / RV	**vxor**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | M}, {%vz | %vix} [, %vm] **pvxor.lo**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | M}, {%vz | %vix} [, %vm] **pvxor.up**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | M}, {%vz | %vix} [, %vm] **pvxor**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | M}, {%vz | %vix} [, %vm]	Vector Exclusive OR *cp* available only on ve3 or later. If 2nd operand is %sy or M, *cp* is available.
VEQV C7 / RV	**veqv**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | M}, {%vz | %vix} [, %vm] **pveqv.lo**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | M}, {%vz | %vix} [, %vm] **pveqv.up**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | M}, {%vz | %vix} [, %vm] **pveqv**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | M}, {%vz | %vix} [, %vm]	Vector Equivalence *cp* available only on ve3 or later. If 2nd operand is %sy or M, *cp* is available.
VLDZ E7 / RV	**vldz** {%vx | %vix}, {%vz | %vix} [, %vm] **pvldz.lo** {%vx | %vix}, {%vz | %vix} [, %vm] **pvldz.up** {%vx | %vix}, {%vz | %vix} [, %vm] **pvldz** {%vx | %vix}, {%vz | %vix} [, %vm]	Vector Leading Zero Count
VPCNT AC / RV	**vpcnt** {%vx | %vix}, {%vz | %vix} [, %vm] **pvpcnt.lo** {%vx | %vix}, {%vz | %vix} [, %vm]	Vector Population Count

	pvpcnt.up {%vx | %vix}, {%vz | %vix} [, %vm] **pvpcnt** {%vx | %vix}, {%vz | %vix} [, %vm]	
VBRV F7 / RV	**vbrv** {%vx | %vix}, {%vz | %vix} [, %vm] **pvbrv.lo** {%vx | %vix}, {%vz | %vix} [, %vm] **pvbrv.up** {%vx | %vix}, {%vz | %vix} [, %vm] **pvbrv** {%vx | %vix}, {%vz | %vix} [, %vm]	Vector Bit Reverse
VSEQ 99 / RV	**vseq** {%vx | %vix} [, %vm] **pvseq.lo** {%vx | %vix} [, %vm] **pvseq.up** {%vx | %vix} [, %vm] **pvseq** {%vx | %vix} [, %vm]	Vector Sequential Number

4.3.2.10 Vector Shift Instructions

Instruction	Assembler Mnemonic Syntax	Description
VSLL E5 / RV	**vsll**[.*cp*] {%vx | %vix}, {%vz | %vix}, {%vy | %vix | %sy | N} [, %vm] **pvsll.lo**[.*cp*] {%vx | %vix}, {%vz | %vix}, {%vy | %vix | %sy | N} [, %vm] **pvsll.up**[.*cp*] {%vx | %vix}, {%vz | %vix}, {%vy | %vix | %sy | N} [, %vm] **pvsll**[.*cp*] {%vx | %vix}, {%vz | %vix}, {%vy | %vix | %sy | N} [, %vm]	Vector Shift Left Logical *cp* available only on ve3 or later. If 3rd operand is %sy or N, *cp* is available.
VSLD E4 / RV	**vsld** {%vx | %vix}, ({%vy | %vix}, {%vz | %vix}), {%sy | N} [, %vm]	Vector Shift Left Double
VSRL F5 / RV	**vsrl**[.*cp*] {%vx | %vix}, {%vz | %vix}, {%vy | %vix | %sy | N} [, %vm] **pvsrl.lo**[.*cp*] {%vx | %vix}, {%vz | %vix}, {%vy | %vix | %sy | N} [, %vm] **pvsrl.up**[.*cp*] {%vx | %vix}, {%vz | %vix}, {%vy | %vix | %sy | N} [, %vm] **pvsrl**[.*cp*] {%vx | %vix}, {%vz | %vix}, {%vy | %vix | %sy | N} [, %vm]	Vector Shift Right Logical *cp* available only on ve3 or later. If 3rd operand is %sy or N, *cp* is available.
VSRD F4 / RV	**vsrd** {%vx | %vix}, ({%vy | %vix}, {%vz | %vix}), {%sy | N} [, %vm]	Vector Shift Right Double
VSLA E6 / RV	**vsla.w**[.*ex*][.*cp*] {%vx | %vix}, {%vz | %vix}, {%vy | %vix | %sy | N} [, %vm] **pvsla.lo**[.*ex*][.*cp*] {%vx | %vix}, {%vz | %vix}, {%vy | %vix | %sy | N} [, %vm] **pvsla.up**[.*cp*] {%vx | %vix}, {%vz | %vix}, {%vy | %vix | %sy | N} [, %vm]	Vector Shift Left Arighmetic *cp* available only on ve3 or later. If 3rd operand is %sy or N, *cp* is

	pvsla[.*cp*] {%vx | %vix}, {%vz | %vix}, {%vy | %vix | %sy | N} [, %vm]	available.
VSLAX D4 / RV	**vsla.l** {%vx | %vix}, {%vz | %vix}, {%vy | %vix | %sy | N} [, %vm]	Vector Shift Left Arithmetic
VSRA F6 / RV	**vsra.w**[.*ex*][.*cp*] {%vx | %vix}, {%vz | %vix}, {%vy | %vix | %sy | N} [, %vm]	Vector Shift Right Arithmetic
	pvsra.lo[.*ex*][.*cp*] {%vx | %vix}, {%vz | %vix}, {%vy | %vix | %sy | N} [, %vm]	*cp* available only on ve3 or later.
	pvsra.up[.*cp*] {%vx | %vix}, {%vz | %vix}, {%vy | %vix | %sy | N} [, %vm]	If 3rd operand is %sy or N, *cp* is available.
	pvsra[.*cp*] {%vx | %vix}, {%vz | %vix}, {%vy | %vix | %sy | N} [, %vm]	
VSRAX D5 / RV	**vsra.l** {%vx | %vix}, {%vz | %vix}, {%vy | %vix | %sy | N} [, %vm]	Vector Shift Right Arithmetic
VSFA D7 / RV	**vsfa** {%vx | %vix}, {%vz | %vix}, {%sy | N}, {%sz | M} [, %vm]	Vector Shift Left and Add

4.3.2.11 Vector Floating-Point Operation Instructions

Instruction	Assembler Mnemonic Syntax	Description
VFAD CC / RV	**vfadd**.*df*[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	Vector Floating Add
	pvfadd.lo[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	*cp* available only on ve3 or later.
	pvfadd.up[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	If 2nd operand is %sy or I, *cp* is available.
	pvfadd[.cp] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	
VFSB DC / RV	**vfsub**.*df*[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	Vector Floating Subtract
	pvfsub.lo[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	*cp* available only on ve3 or later.
	pvfsub.up[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	If 2nd operand is %sy or I, *cp* is available.
	pvfsub[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	
VFMP CD / RV	**vfmul**.*df*[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	Vector Floating Multiply
	pvfmul.lo[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	*cp* available only on ve3 or later.
	pvfmul.up[.*cp*] {%vx | %vix}, {%vy | %vix	If 2nd operand

	`| %sy | I}, {%vz | %vix} [, %vm]`	is %sy or I, *cp* is available.
	pvfmul[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	
VFDV DD / RV	**vfdiv**.*df* {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix | %sy | I} [, %vm]	Vector Floating Divide
VFSQRT ED / RV	**vfsqrt**.*df* {%vx | %vix}, {%vy | %vix} [, %vm]	Vector Floating Square Root
VFCP FC / RV	**vfcmp**.*df*[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	Vector Floating Compare
	pvfcmp.lo[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	*cp* available only on ve3 or later.
	pvfcmp.up[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	If 2nd operand is %sy or I, *cp* is available.
	pvfcmp[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	
VFCM BD / RV	**vfmax**.*df*[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	Vector Floating Compare and Select Maximum/Minimum
	pvfmax.lo[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	*cp* available only on ve3 or later.
	pvfmax.up[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	If 2nd operand is %sy or I, *cp* is available.
	pvfmax[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	
	vfmin.*df*[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	
	pvfmin.lo[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	
	pvfmin.up[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	
	pvfmin[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	
VFMAD E2 / RV	**vfmad**.*df* {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%vw | %vix} [, %vm]	Vector Floating Fused Multiply Add
	pvfmad.lo {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%vw | %vix} [, %vm]	Calculate as follows (%vix, I and %vm omitted) ・ 2nd operand is %vy, 3rd operand is %vz %vx=%vz*%vw +%vy
	pvfmad.up {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%vw | %vix} [, %vm]	
	pvfmad {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%vw | %vix} [, %vm]	
	vfmad.*df* {%vx | %vix}, {%sy | I}, {%vz | %vix}, {%vw | %vix} [, %vm]	

	pvfmad.lo {%vx | %vix}, {%sy | I}, {%vz | %vix}, {%vw | %vix} [, %vm] **pvfmad.up** {%vx | %vix}, {%sy | I}, {%vz | %vix}, {%vw | %vix} [, %vm] **pvfmad** {%vx | %vix}, {%sy | I}, {%vz | %vix}, {%vw | %vix} [, %vm] **vfmad.***df* {%vx | %vix}, {%vy | %vix}, {%sy | I}, {%vw | %vix} [, %vm] **pvfmad.lo** {%vx | %vix}, {%vy | %vix}, {%sy | I}, {%vw | %vix} [, %vm] **pvfmad.up** {%vx | %vix}, {%vy | %vix}, {%sy | I}, {%vw | %vix} [, %vm] **pvfmad** {%vx | %vix}, {%vy | %vix}, {%sy | I}, {%vw | %vix} [, %vm]	・ 2nd operand is %sy, 3rd operand is %vz %vx=%vz*%vw+%sy ・ 2nd operand is %vy, 3rd operand is %sy %vx=%sy*%vw+%vy
VFMSB F2 / RV	**vfmsb.***df* {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%vw | %vix} [, %vm] **pvfmsb.lo** {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%vw | %vix} [, %vm] **pvfmsb.up** {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%vw | %vix} [, %vm] **pvfmsb** {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%vw | %vix} [, %vm] **vfmsb.***df* {%vx | %vix}, {%sy | I}, {%vz | %vix}, {%vw | %vix} [, %vm] **pvfmsb.lo** {%vx | %vix}, {%sy | I}, {%vz | %vix}, {%vw | %vix} [, %vm] **pvfmsb.up** {%vx | %vix}, {%sy | I}, {%vz | %vix}, {%vw | %vix} [, %vm] **pvfmsb** {%vx | %vix}, {%sy | I}, {%vz | %vix}, {%vw | %vix} [, %vm] **vfmsb.***df* {%vx | %vix}, {%vy | %vix}, {%sy | I}, {%vw | %vix} [, %vm] **pvfmsb.lo** {%vx | %vix}, {%vy | %vix}, {%sy | I}, {%vw | %vix} [, %vm] **pvfmsb.up** {%vx | %vix}, {%vy | %vix}, {%sy | I}, {%vw | %vix} [, %vm] **pvfmsb** {%vx | %vix}, {%vy | %vix}, {%sy | I}, {%vw | %vix} [, %vm]	Vector Floating Fused Multiply Subtract Calculate as follows (%vix, I and %vm omitted) ・ 2nd operand is %vy, 3rd operand is %vz %vx=%vz*%vw-%vy ・ 2nd operand is %sy, 3rd operand is %vz %vx=%vz*%vw-%sy ・ 2nd operand is %vy, 3rd operand is %sy %vx=%sy*%vw-%vy
VFNMAD E3 / RV	**vfnmad.***df* {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%vw | %vix} [, %vm] **pvfnmad.lo** {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%vw | %vix} [, %vm]	Vector Floating Fused Negative Multiply Add Calculate as follows

	pvfnmad.up {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%vw | %vix} [, %vm] **pvfnmad** {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%vw | %vix} [, %vm] **vfnmad.***df* {%vx | %vix}, {%sy | I}, {%vz | %vix}, {%vw | %vix} [, %vm] **pvfnmad.lo** {%vx | %vix}, {%sy | I}, {%vz | %vix}, {%vw | %vix} [, %vm] **pvfnmad.up** {%vx | %vix}, {%sy | I}, {%vz | %vix}, {%vw | %vix} [, %vm] **pvfnmad** {%vx | %vix}, {%sy | I}, {%vz | %vix}, {%vw | %vix} [, %vm] **vfnmad.***df* {%vx | %vix}, {%vy | %vix}, {%sy | I}, {%vw | %vix} [, %vm] **pvfnmad.lo** {%vx | %vix}, {%vy | %vix}, {%sy | I}, {%vw | %vix} [, %vm] **pvfnmad.up** {%vx | %vix}, {%vy | %vix}, {%sy | I}, {%vw | %vix} [, %vm] **pvfnmad** {%vx | %vix}, {%vy | %vix}, {%sy | I}, {%vw | %vix} [, %vm]	(%vix, I and %vm omitted) ・ 2nd operand is %vy, 3rd operand is %vz %vx=-%vz*%vw+%vy ・ 2nd operand is %sy, 3rd operand is %vz %vx=-%vz*%vw+%sy ・ 2nd operand is %vy, 3rd operand is %sy %vx=-%sy*%vw+%vy
VFNMSB F3 / RV	**vfnmsb.***df* {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%vw | %vix} [, %vm] **pvfnmsb.lo** {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%vw | %vix} [, %vm] **pvfnmsb.up** {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%vw | %vix} [, %vm] **pvfnmsb** {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%vw | %vix} [, %vm] **vfnmsb.***df* {%vx | %vix}, {%sy | I}, {%vz | %vix}, {%vw | %vix} [, %vm] **pvfnmsb.lo** {%vx | %vix}, {%sy | I}, {%vz | %vix}, {%vw | %vix} [, %vm] **pvfnmsb.up** {%vx | %vix}, {%sy | I}, {%vz | %vix}, {%vw | %vix} [, %vm] **pvfnmsb** {%vx | %vix}, {%sy | I}, {%vz | %vix}, {%vw | %vix} [, %vm] **vfnmsb.***df* {%vx | %vix}, {%vy | %vix}, {%sy | I}, {%vw | %vix} [, %vm] **pvfnmsb.lo** {%vx | %vix}, {%vy | %vix}, {%sy | I}, {%vw | %vix} [, %vm] **pvfnmsb.up** {%vx | %vix}, {%vy | %vix}, {%sy | I}, {%vw | %vix} [, %vm]	Vector Floating Fused Negative Multiply Subtract Calculate as follows (%vix, I and %vm omitted) ・ 2nd operand is %vy, 3rd operand is %vz %vx=-%vz*%vw-%vy ・ 2nd operand is %sy, 3rd operand is %vz %vx=-%vz*%vw-%sy ・ 2nd operand is %vy, 3rd operand is %sy %vx=-%sy*%vw-%vy

	pvfnmsb {%vx | %vix}, {%vy | %vix}, {%sy | I}, {%vw | %vix} [, %vm]	
VRCP E1 / RV	**vrcp**.*df* {%vx | %vix}, {%vy | %vix} [, %vm] **pvrcp.lo** {%vx | %vix}, {%vy | %vix} [, %vm] **pvrcp.up** {%vx | %vix}, {%vy | %vix} [, %vm] **pvrcp** {%vx | %vix}, {%vy | %vix} [, %vm]	Vector Floating Reciprocal
VRSQRT F1 / RV	**vrsqrt**.*df*[.*nex*] {%vx | %vix}, {%vy | %vix} [, %vm] **pvrsqrt.lo**[.*nex*] {%vx | %vix}, {%vy | %vix} [, %vm] **pvrsqrt.up**[.*nex*] {%vx | %vix}, {%vy | %vix} [, %vm] **pvrsqrt**[.*nex*] {%vx | %vix}, {%vy | %vix} [, %vm]	Vector Floating Reciprocal Square Root
VFIX E8 / RV	**vcvt.w**.*df*[.*ex*][.*rd*] {%vx | %vix}, {%vy | %vix} [, %vm] **pvcvt.w.s.lo**[.*rd*] {%vx | %vix}, {%vy | %vix} [, %vm] **pvcvt.w.s.up**[.*rd*] {%vx | %vix}, {%vy | %vix} [, %vm] **pvcvt.w.s**[.*rd*] {%vx | %vix}, {%vy | %vix} [, %vm]	Vector Convert to Fixed Point
VFIXX A8 / RV	**vcvt.l.d**[.*rd*] {%vx | %vix}, {%vy | %vix} [, %vm]	Vector Convert to Fixed Point
VFLT F8 / RV	**vcvt**.*df*.**w** {%vx | %vix}, {%vy | %vix} [, %vm] **pvcvt.s.w.lo** {%vx | %vix}, {%vy | %vix} [, %vm] **pvcvt.s.w.up** {%vx | %vix}, {%vy | %vix} [, %vm] **pvcvt.s.w** {%vx | %vix}, {%vy | %vix} [, %vm]	Vector Convert to Floating Point
VFLTX B8 / RV	**vcvt.d.l** {%vx | %vix}, {%vy | %vix} [, %vm]	Vector Convert to Floating Point
VCVD 8F / RV	**vcvt.d.s** {%vx | %vix}, {%vy | %vix} [, %vm]	Vector Convert to Single-Format
VCVS 9F / RV	**vcvt.s.d** {%vx | %vix}, {%vy | %vix} [, %vm] **pvcvt.s.h.lo** {%vx | %vix}, {%vy | %vix} [, %vm] **pvcvt.s.h.up** {%vx | %vix}, {%vy | %vix} [, %vm] **pvcvt.s.h** {%vx | %vix}, {%vy | %vix} [, %vm]	Vector Convert to Double-Format `pvcvt` available only on ve3 or later.

VCVH AE / RV	**pvcvt.h.s.lo** {%vx | %vix}, {%vy | %vix} [, %vm] **pvcvt.h.s.up** {%vx | %vix}, {%vy | %vix} [, %vm] **pvcvt.h.s** {%vx | %vix}, {%vy | %vix} [, %vm]	Vector integer compare and select signed 64bit Available only on ve3 or later.

4.3.2.12 Vector Mask Arithmetic Instructions

Instruction	Assembler Mnemonic Syntax	Description
VMRG D6 / RV	**vmrg**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **vmrg.l**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm] **vmrg.w**[.*cp*] {%vx | %vix}, {%vy | %vix | %sy | I}, {%vz | %vix} [, %vm]	Vector Merge *cp* available only on ve3 or later. If 2nd operand is %sy or I, *cp* is available.
VSHF BC / RV	**vshf** {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%sy | N}	Vector Shuffle N = 0 - 15
VCP 8D / RV	**vcp** {%vx | %vix}, {%vz | %vix}[, %vm]	Vector Compress
VEX 9D / RV	**vex** {%vx | %vix}, {%vz | %vix}[, %vm]	Vector Expand
VFMK B4 / RV	**vfmk.l**.*cf* %vmx, {%vz | %vix} [, %vm]	Vector Form Mask If *cf* is “af” or “at”, {%vz | %vix} cannot be specified. If *cf* is “at”, *cf* can be omitted.
VFMS B5 / RV	**vfmk.w**.*cf* %vmx, {%vz | %vix} [, %vm] **pvfmk.w.lo**.*cf* %vmx, {%vz | %vix} [, %vm] **pvfmk.w.up**.*cf* %vmx, {%vz | %vix} [, %vm]	Vector Form Mask Single If *cf* is “af” or “at”, {%vz | %vix} cannot be specified. If *cf* is “at”, *cf* can be omitted.
VFMF B6 / RV	**vfmk**.*df*.*cf* %vmx, {%vz | %vix} [, %vm] **pvfmk.s.lo**.*cf* %vmx, {%vz | %vix} [, %vm] **pvfmk.s.up**.*cf* %vmx, {%vz | %vix} [, %vm]	Vector Form Mask Floating Point If *cf* is “af” or “at”, {%vz | %vix} cannot be specified. If *cf* is “at”, *cf* can be omitted.

4.3.2.13 Vector Recursive Relation Instructions

Instruction	Assembler Mnemonic Syntax	Description
VSUMS EA / RV	**vsum.w**[*.ex*] {%vx | %vix}, {%vy | %vix} [, %vm]	Vector Sum Single
VSUMX AA / RV	**vsum.l** {%vx | %vix}, {%vy | %vix} [, %vm]	Vector Sum
VFSUM EC / RV	**vfsum.***df* {%vx | %vix}, {%vy | %vix} [, %vm]	Vector Floating Sum
VMAXS BB / RV	**vrmaxs.w.***pos*[*.ex*] {%vx | %vix}, {%vy | %vix} [, %vm] **vrmins.w.***pos*[*.ex*] {%vx | %vix}, {%vy | %vix} [, %vm]	Vector Maximum/Minimum Single
VMAXX AB / RV	**vrmaxs.l.***pos* {%vx | %vix}, {%vy | %vix} [, %vm] **vrmins.l.***pos* {%vx | %vix}, {%vy | %vix} [, %vm]	Vector Maximum/Minimum
VFMAX AD / RV	**vfrmax.***df*.*pos* {%vx | %vix}, {%vy | %vix} [, %vm] **vfrmin.***df*.*pos* {%vx | %vix}, {%vy | %vix} [, %vm]	Vector Floating Maximum/Minimum
VRAND 88 / RV	**vrand** {%vx | %vix}, {%vy | %vix} [, %vm]	Vector Reduction AND
VROR 98 / RV	**vror** {%vx | %vix}, {%vy | %vix} [, %vm]	Vector Reduction OR
VRXOR 89 / RV	**vrxor** {%vx | %vix}, {%vy | %vix} [, %vm]	Vector Reduction Exclusive OR
VFIA CE / RV	**vfia.***df* {%vx | %vix}, {%vy | %vix}, {%sy | I}	Vector Floating Iteration Add
VFIS DE / RV	**vfis.***df* {%vx | %vix}, {%vy | %vix}, {%sy | I}	Vector Floating Iteration Subtract
VFIM CF / RV	**vfim.***df* {%vx | %vix}, {%vy | %vix}, {%sy | I}	Vector Floating Iteration Multiply
VFIAM EE / RV	**vfiam.***df* {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%sy | I}	Vector Floating Iteration Add and Multiply
VFISM FE / RV	**vfism.***df* {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%sy | I}	Vector Floating Iteration Subtract

		and Multiply
VFIMA EF / RV	**vfima**.*df* {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%sy | I}	Vector Floating Iteration Multiply and Add
VFIMS FF / RV	**vfims**.*df* {%vx | %vix}, {%vy | %vix}, {%vz | %vix}, {%sy | I}	Vector Floating Iteration Multiply and Subtract

4.3.2.14 Vector Gatering/Scattering Instructions

Instruction	Assembler Mnemonic Syntax	Description
VGT A1 / RVM	**vgt**[.*nc*] {%vx | %vix}, {%vy | %vix | %sw}, {%sy | I}, {%sz | Z} [,%vm]	Vector Gather
VGTU A2 / RVM	**vgtu**[.*nc*] {%vx | %vix}, {%vy | %vix | %sw}, {%sy | I}, {%sz | Z} [,%vm]	Vector Gather Upper
VGTL A3 / RVM	**vgtl**[.*ex*][.*nc*] {%vx | %vix}, {%vy | %vix | %sw}, {%sy | I}, {%sz | Z} [,%vm]	Vector Gather Lower
VSC B1 / RVM	**vsc**[.*nc*][.*ot*] {%vx | %vix}, {%vy | %vix | %sw}, {%sy | I}, {%sz | Z} [, %vm]	Vector Scatter
VLFA B1 / RVM	**vlfa**[.*nc*][.*ot*] {%vx | %vix}, {%vy | %vix | %sw}, {%sy | I}, {%sz | Z} [, %vm]	Vector list floating add Available only on ve3 or later.
VSCU B2 / RVM	**vscu**[.*nc*][.*ot*] {%vx | %vix}, {%vy | %vix | %sw}, {%sy | I}, {%sz | Z} [, %vm]	Vector Scatter Upper
VLFAU B2 / RVM	**vlfau**[.*nc*][.*ot*] {%vx | %vix}, {%vy | %vix | %sw}, {%sy | I}, {%sz | Z} [, %vm]	Vector list floating add upper Available only on ve3 or later.
VSCL B3 / RVM	**vscl**[.*nc*][.*ot*] {%vx | %vix}, {%vy | %vix | %sw}, {%sy | I}, {%sz | Z} [, %vm]	Vector Scatter Lower

4.3.2.15 Vector Mask Register Instructions

Instruction	Assembler Mnemonic Syntax	Description
ANDM 84 / RV	**andm** %vmx, %vmy, %vmz	AND VM
ORM	**orm** %vmx, %vmy, %vmz	OR VM

85 / RV		
XORM 86 / RV	**xorm** %vmx, %vmy, %vmz	Exclusive OR VM
EQVM 87 / RV	**eqvm** %vmx, %vmy, %vmz	Equivalence VM
NNDM 94 / RV	**nndm** %vmx, %vmy, %vmz	Negate AND VM
NEGM 95 / RV	**negm** %vmx, %vmy	Negate VM
PCVM A4 / RV	**pcvm** %sx, %vmy	Population Count of VM
LZVM A5 / RV	**lzvm** %sx, %vmy	Leading Zero of VM
TOVM A6 / RV	**tovm** %sx, %vmy	Trailing One of VM

4.3.2.16 Vector Control Instructions

Instruction	Assembler Mnemonic Syntax	Description
LVL BF / RR	**lvl** {%sy | I} **lpvl** {%sy | I}	Load VL lpvl available only on ve3 or later.
SVL 2F / RR	**svl** %sx **spvl** %sx	Save VL spvl available only on ve3 or later.
SMVL 2E / RR	**smvl** %sx **smpvl** %sx	Save Maximum Vector Length smpvl available only on ve3 or later.
LVIX AF / RR	**lvix** {%sy | N}	Load Vector Data Index N = 0 - 63

4.3.2.17 Control Instructions

Instruction	Assembler Mnemonic Syntax	Description
SIC 28 / RR	**sic** %sx	Save Instruction Counter
LPM	**lpm** %sy	Load Program Mode

3A / RR		Flags
SPM 2A / RR	**spm** %sx	Save Program Mode Flags
LFR 69 / RR	**lfr** {%sy | N}	Load Flag Register N = 0 - 63
SFR 29 / RR	**sfr** %sx	Safe Flag Register
SMIR 22 / RR	**smir**[.*async*] %sx, I **smir**[.*async*] %sx, %usrcc **smir**[.*async*] %sx, %psw **smir**[.*async*] %sx, %sar **smir**[.*async*] %sx, %pmmr **smir**[.*async*] %sx, %pmcrMM **smir**[.*async*] %sx, %pmcNN	Save Miscellaneous Register *async* available only on ve3 or later. I = 0 - 2, 7 - 11, 16 - 36 MM = 0 - 3 NN = 0 - 20
NOP 79 / RR	**nop**	No Operation
MONC 3F / RR	**monc** [N, N, N] **monc.hdb** [N, N, N]	Monitor Call 2nd and 3rd operand : N = 0 - 255
LCR 40 / RR	**lcr** %sx, {%sy|I}, {%sz|Z}	Load Communication Register
SCR 50 / RR	**scr** %sx, {%sy|I}, {%sz|Z}	Store Communication Register
TSCR 41 / RR	**tscr** %sx, {%sy|I}, {%sz|Z}	Test and Set Communication Register
FIDCR 51 / RR	**fidcr** %sx, {%sy|I}, I	Fetch and Increment/Decrement CR 3rd operand : I = 0 - 7
TS1AM 42 / RRM	**ts1am.l** %sx, {%sz | AS}, {%sy|N} **ts1am.w** %sx, {%sz | AS}, {%sy|N}	Test and Set 1 AM
TS2AM 43 / RRM	**ts2am** %sx, {%sz | AS}, {%sy|N}	Test and Set 2 AM

TS3AM 52 / RRM	**ts3am** %sx, {%sz | AS}, {%sy|N}	Test and Set 3 AM N = 0 or 1
ATMAM 53 / RRM	**atmam** %sx, {%sz | AS}, {%sy|N}	Atomic AM N = 0 – 2
CAS 62 / RRM	**cas.l** %sx, {%sz | AS}, {%sy|I} **cas.w** %sx, {%sz | AS}, {%sy|I}	Compare and Swap
FENCE 20 / RR	**fencei** **fencem** I **fencec I**	Fence fencem : I = 1 – 3 Fencec : I = 1 – 7
SVOB 30 / RR	**svob**	Set Vector Out-of-order memory access Boundary
BSWP 2B / RR	**bswp** %sx, {%sz | M}, I	Byte Swap I = 0 or 1

4.3.2.18 Host Memory Access Instructions

Instruction	Assembler Mnemonic Syntax	Description
LHM 21 / RRM	**lhm.b** %sx, HM **lhm.h** %sx, HM **lhm.w** %sx, HM **lhm.l** %sx, HM	Load Host Memory
SHM 31 / RRM	**shm.b** %sx, HM **shm.h** %sx, HM **shm.w** %sx, HM **shm.l** %sx, HM	Store Host Memory

Appendix A History

A.1 History table

Nov. 2018	Rev. 1	Create a new entry
Apr. 2018	Rev.1.1	Revise
Dec. 2018	Rev.1.2	Revise
Sep. 2019	Rev.1.3	Revise
Mar. 2023	Rev1.4	Describe the changes in the ve3 architecture Describe the changes in half-precision floating point number Add calculation method for VFMAD, VFMSB, VFNMAD and VFNMSB Instruction Add option and pseudo-instruction

A.2 Change notes

The following changes are done in this edition.

- Describe the changes in the ve3 architecture
- Describe the changes in half-precision floating point number
- Add calculation method for VFMAD, VFMSB, VFNMAD and VFNMSB Instruction
- Add option and pseudo-instruction