AndeStar Instruction Set Architecture FPU Extension Manual

Document Number UM029-14

Date Issued 2017-02-08

Copyright Notice

Contact Information

Should you have any problems with the information contained herein, please contact Andes Technology Corporation
by email support@andestech.com
or online website https://es.andestech.com/eservice/
for support giving:

- the document title
- the document number
- the page number(s) to which your comments apply
- a concise explanation of the problem

General suggestions for improvements are welcome.

Revision History

Rev.	Revision Date	Revised Content and Revised Chapter/Section
V1.4	2017/2/8	1. Clarified the operations of FNMADDx, FNMSUBx. 2. Clarified the non-trapped IVO result of FMADDx, FMSUBx, FNMADDx, and FNMSUBx. 3. Clarified the non-trapped IVO result of FADDx, FSUBx, FMULx, FDIVx, FSQRTx. 4. Added section 5.1.1. 5. Added N15/D15 FPU latency table. (Sec 5.2)
V1.3	2013/12/2	Fixed typos. (Sec 2.3, 7)
V1.2	2011/12/9	1. Added a dsb instruction requirement for the FMTCSR instruction. 2. Added encoding table and notes. (Sec 4.1)
V1.1	2011/2/11	Typo corrections. (Sec 3.1, 3.2)
V1.0	2010/1/29	Initial Release

Table of Contents

List of Tables

List of Figures

Typographical Convention Index

Document Element	Font	Font Style	Size	Color
Normal text	Georgia	Normal	12	Black
`Command line, source code or file paths`	`Lucida Console`	`Normal`	`11`	`Indigo`
`VARIABLES OR PARAMETERS IN COMMAND LINE, SOURCE CODE OR FILE PATHS`	**`LUCIDA CONSOLE`**	**`BOLD + ALL-CAPS`**	**`11`**	**`INDIGO`**
Note or warning	Georgia	Normal	12	Red
Hyperlink	Georgia	Underlined	12	Blue

1. Introduction

This section gives an overview of the Andes FP SP/DP V1 extension architecture. The SP extension is for single-precision floating-point data format. The DP extension is for double-precision floating-point data format.

1.1. Architecture Registers Defined For FPU ISA Extension

1.1.1. General purpose floating-point registers

The Andes FP SP V1 extension uses additional 32-bit floating-point registers (FSx) to hold single-precision floating-pint values and performs single-precision floating-point operations on them. The minimum number of the 32-bit single-precision floating-point registers is eight and can be extended up to thirty-two.

The Andes FP DP V1 extension uses additional 64-bit floating-point registers (FDx) to hold double-precision floating-point values and performs double-precision floating-point operations on them. The first sixteen 64-bit floating-point registers are overlapped with the thirty-two 32-bit single-precision floating-point registers when both SP and DP extensions are implemented. The minimum number of the 64-bit double-precision floating-point registers is four and can be extended up to thirty-two. The overlapping format of the single-precision registers and the double-precision registers are illustrated as follows:

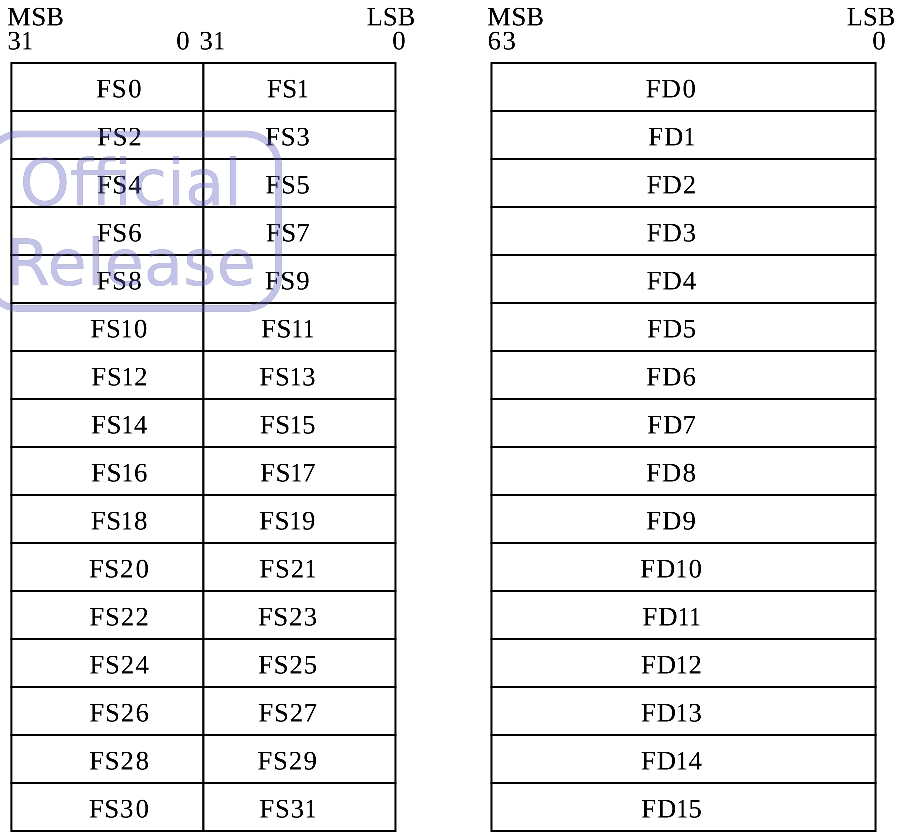

Figure 1. Overlapping of single-precision and double-precision floating-point registers

Four configuration options are defined for the number of registers in an implementation:

Configuration #	**Meaning**
0	8 SP / 4 DP registers
1	16 SP / 8 DP registers
2	32 SP / 16 DP registers
3	32 SP / 32 DP registers

A program compiled for a configuration A can be run on a FPU implementation that implements configuration B without any problem provided that A and B has the following relationships:

Configuration A	Configuration B
0	0, 1, 2, 3
1	1, 2, 3
2	2, 3
3	3

1.1.2. Other registers

Additional registers are defined to hold configuration, control, and status information. They are:

- FPCSR (Floating-Point Control & Status Register): contains rounding modes selection, IEEE floating-point exception trapping enables and cumulative flags. It is a read/write register. It is read using the FMFCSR instruction and is written using the FMTCSR instruction. The updated content of this register and its side effects can be seen by the next floating-point instruction without an intervening DSB instruction. Please see section 4.1for detailed FPCSR definitions.

- FPCFG (Floating-Point Unit Configuration): contains version of FP extension architecture. It is a read only register. It is read using the FMFCFG instruction. Please see section 4.2for detailed FPCFG definitions.

Both registers can be accessed by user mode and superuser mode programs.

1.2. Floating-point Data Format

1.2.1. Floating-point data types

The data types provided by Andes FPU are

- Single precision floating-point data type specified by the IEEE standard in SP extension.
- Double precision floating-point data type specified by the IEEE standard in DP extension.

The 32-bit single-precision and 64-bit double-precision floating-point data formats are shown in the following figures:

32-bit single precision floating-point format:

31	30 … 23	22 … 0
S	Exponent (e)	Significand (f)

64-bit double precision floating-point format

63	62 … 52	51 … 0
S	Exponent (e)	Significand (f)

A normal finite floating-point number of the following form,

$$(-1)^{S} \cdot 2^{e\text{-bias}} \cdot 1.b_1 b_2 \ldots b_{p-1}$$

is represented in the above formats with the following three fields:

1. 1-bit sign S
2. Biased exponent e = E+bias
3. Significand (Fraction) f = $.b_1 b_2 b \ldots b_{p-1}$

The defined values/parameters of these three components for the single precision and double precision format can be summarized in the following table:

Parameter	Single precision value	Double precision
Bits of precision, p	23 + 1	52 + 1
Representation of b_0 integer bit	Hidden 1	Hidden 1
Bits for Significand field (f)	23	52
Significand range	[1 , 2 - 2^{-23}]	[1 , 2 – 2^{-52}]
Bits for Exponent field (e)	8	11
Exponent bias	127	1023
Max exponent, E_max (E = e – bias)	127 (e == 254)	1023 (e = 2046)
Minimum exponent, E_min (E = e - bias)	-126 (e == 1)	-1022 (e == 1)
Reserved Biased Exponent (e)	0, 255	0, 2047

Reserved biased exponent values are used to encode special values defined in the IEEE 754 standard. These special values are described in the following sections.

1.2.2. Denormalized Number

Denormalized number is a nonzero, but small, floating-point number that cannot be represented in the normalized floating-point number representation described in section 1.2.1The uses of denormalized numbers allow the error effect of underflow to be gradual. Thus such a small floating-point number can now be represented more precisely without rounding to zero. The encodings and representations of Denormalized number in the single-precision and double-precision formats are

- Single precision: e = 0, f ≠ 0, value = $(-1)^{S} \cdot 2^{-126} \cdot (0.f)$
- Double precision: e = 0, f ≠ 0, value = $(-1)^{S} \cdot 2^{-1022} \cdot (0.f)$

The FPU hardware supporting of handling Denormalized numbers is implementation-dependent.

1.2.3. Zeros

Two zeros are defined, +0 and -0.

- +0 : S = 0, e = 0, and f = 0
- -0 : S = 1, e = 0, and f = 0

1.2.4. Infinities

Two infinities are defined, +∞ and -∞.

- +∞ : represents positive numbers which are too big to be represented accurately as normalized numbers
  - Single precision: S = 0, e = 255 (0xFF), and f = 0
  - Double precision: S = 0, e = 2047 (0x7FF), and f = 0
- -∞ : represents negative numbers which are too big to be represented accurately as normalized numbers
  - Single precision: S = 1, e = 255 (0xFF), and f = 0
  - Double precision: S = 1, e = 2047 (0x7FF), and f = 0

1.2.5. NaNs

NaN is an abbreviation for “Not a number”. It is a symbolic entity encoded in floating-point format to represent values which can be used neither as numerical values nor as infinities.

IEEE standard defines NaN as being:

- Single precision: e = 255 (0xFF), and f ≠ 0, regardless of S
- Double precision: e = 2047 (0x7FF), and f ≠ 0, regardless of S

Within the NaN category, there are two types of NaNs.

- Signaling NaNs: they signal the invalid operation exception whenever they appear as operands.
- Quiet NaNs: they propagate through almost every arithmetic operation without signaling exceptions.

IEEE Standard requires each format to contain at least one signaling NaN and at least one quite NaN without specifying the exact parameters to distinguish them. Andes FPU architecture further specifies a rule to distinguish them based on the most significant fraction bit of each format.

- Single precision
  - Signaling NaNs: Bit[22] = 0, the remaining fraction bits can take any value except all zeros, the sign bit can take either value.
  - Quiet NaNs: Bit[22] = 1, the remaining fraction bits can take any value except all zeros, the sign bit can take either value.
- Double precision
  - Signaling NaNs: Bit[51] = 0, the remaining fraction bits can take any value except all zeros, the sign bit can take either value.
  - Quiet NaNs: Bit[51] = 1, the remaining fraction bits can take any value except all zeros, the sign bit can take either value.

When an invalid operation that does not involve NaN input operands has happened without an enabled trap, a new quite NaN needs to be delivered as a default result. Andes FPU architecture defines a default QNaN in each format as follows for this purpose.

Floating-point Format	New QNaN value
Single-precision	0xFFFFFFFF
Double-precision	0xFFFFFFFF FFFFFFFF

1.3. IEEE Rounding Modes

IEEE standard defines four rounding modes as follows:

- Round to Nearest (Even): This is the default rounding mode. In this mode, the representable value nearest to the infinitely precise result shall be delivered; if the two nearest representable values are equally near, the one with its least significant bit zero (even) shall be delivered. However, an infinitely precise result with magnitude at least $2^{127}(2-2^{-24})$ for SP, or $2^{1023}(2-2^{-53})$ for DP, shall round to ∞ with no change in sign. We will call it RNE rounding mode in the remaining of this document.
- Round towards +∞: Round the result to the value closest to, but not less than the infinitely precise result. We will call it RPI rounding mode in the remaining of this document.
- Round towards -∞: Round the result to the value closest to, but not greater than the infinitely precise result. We will call it RMI rounding mode in the remaining of this document.
- Round towards 0: Round the result to the value closest to, but not greater in magnitude than the infinitely precise result. We will call it RZE rounding mode in the remaining of this document.

1.4. Floating-point Exceptions

1.4.1. IEEE Standard Required Exceptions

The Andes FP SP/DP V1 extension supports all of the five floating-point exceptions defined in the IEEE 754 standard:

- Invalid Operation exception
- Inexact exception
- Overflow exception
- Underflow exception
- Divide by Zero exception

These IEEE exceptions may or may not generate real Andes exception events based on individual exception enable/disable settings in the FPCSR. When an IEEE exception is enabled, we say that the exception is in trapped mode. When an IEEE exception is disabled, we say that the exception is in non-trapped mode. The default trapping behavior is non-trapped mode.

The non-trapped exceptions set individual cumulative flags in the FPCSR while the trapped exceptions do not set the flags. These cumulative exception status flags can only be cleared by executing a FMTCSR instruction.

A trapped exception will set the corresponding exception type bit in the FPCSR register.

Invalid Operation exception

An Invalid Operation exception is happened if neither a numeric value nor infinity is a sensible result of a floating-point operation, and when an operand of a floating-point operation is a signaling NaN. Specifically, the Invalid Operation exception is generated in the following conditions:

- Any operation on a signaling NaN.
- Addition of opposite-signed infinities.
- Subtraction of same-signed infinities.

- Multiplication of 0*infinity.
- Division of 0/0 or infinity/infinity.
- Square roots, whose operands are negative, including minus infinity but excluding -0.
- Conversion of an infinity or NaN to an integer.

When in non-trapped mode, the result is set to QNaN.

Inexact exception

An Inexact exception is happened from the following two situations:

- If the rounded result of a floating-point operation is different from the exact un-rounded result.
- If the rounded result of a floating-point operation overflows and the overflow exception is in non-trapped mode.

When in non-trapped mode, the result is the rounded or the overflowed result.

Overflow exception

An Overflow exception is happened if the ideally rounded (with unlimited unbiased exponent range) result of a floating point operation is too big for the largest finite number of the destination format (that is, >127 for single precision or >1023 for double precision).

When in non-trapped mode, the result is based on the rounding mode and the sign of the result as follows:

- RNE rounding mode: $+\infty$ or $-\infty$
- RPI rounding mode: $+\infty$ or largest negative number
- RMI rounding mode: largest positive number or $-\infty$
- RZE: largest positive number or largest negative number

Underflow exception

An Underflow exception is detected based on two conditions. One is the generation of a tiny nonzero result between $\pm 2^{-126}$ for single-precision and $\pm 2^{-1022}$ for double-precision. This is called tininess. The other condition is loss of accuracy during the approximation of the tiny numbers by denormalized numbers. This is just called loss of accuracy.

When in trapped mode, the underflow exception should be signaled when tininess is detected regardless of loss of accuracy. When in non-trapped mode, the underflow exception should be signaled when both tininess and loss of accuracy are detected.

The detection of tininess can be either “after rounding” or “before rounding”. The Andes FPU architecture selects “after rounding” as tininess detection. The detection of loss of accuracy can be due to either “a denormalized loss” or “an inexact result”. The Andes FPU architecture selects “inexact” as loss of accuracy detection when an implementation supports arithmetic for denormalized numbers, but selects “denormalized loss” as loss of accuracy detection when there is no support for denormalized numbers.

When there is no support for denormalized numbers, the “inexact” detection of a denormalized result is difficult to perform and the denormalized result will be turned into a result other than the denormalized number itself in non-trapped mode, so selecting “denormalized loss” as loss of accuracy detection is the only choice in this condition.

When in non-trapped mode, the non-trapping result might be zero, denormalized number, or $\pm 2^{-Emin}$. If an implementation supports arithmetic for de-normalized numbers, the non-trapping result should be a properly rounded de-normalized number. If an implementation does not support arithmetic for de-normalized numbers, then the non-trapping result should be as follows depending on the rounding mode and the sign of the result.

Rounding Mode	**When $0 < x < 2^{-Emin}$**	**When $-2^{-Emin} < x < 0$**
RNE	+0	-0
RPI	2^{-Emin}	-0
RMI	+0	-2^{-Emin}
RZE	+0	-0

Divide By Zero exception

A “Divide by Zero” exception occurs if the divisor is zero and the dividend is a finite nonzero number. When no trap happens, the result should be a correctly signed ∞.

1.4.2. Simultaneous Generation of Multiple IEEE Floating-point Exceptions

Among the five IEEE floating-point exceptions, only INEXACT exception can happen simultaneously with two other exceptions, OVERFLOW and UNDERFLOW. When that happens, the INEXACT exception is subordinate to the primary exception case, OVERFLOW or UNDERFLOW.

The following table lists the correct behaviors for the OVERFLOW/INEXACT case under different trap enable conditions.

Trap enable bit		**IEEE exception flag**		**FPU exception type**	
Inexact exception (FPCSR.IEXE)	**Overflow exception (FPCSR.OVFE)**	**IEX**	**OVF**	**IEXT**	**OVFT**
0	0	1	1	x	x
0	1	x	x	0	1
1	0	x	1	1	0
1	1	x	x	0	1

“x” in the above table means “no change”. The grayed out entry does not exist in the OVERFLOW/INEXACT case.

The following table lists the correct behaviors for the UNDERFLOW/INEXACT case under different trap enable conditions.

Trap enable bit		IEEE exception flag		FPU exception type	
Inexact exception (FPCSR.IEXE)	Underflow exception (FPCSR.UDFE)	IEX	UDF	IEXT	UDFT
0	0	1	1	x	x
0	1	1	x	0	1
1	0	x	1	1	0
1	0	x	x	0	1

“x” in the above table means “no change”.

1.4.3. Non-IEEE Standard Required Exceptions

Andes FPU may generate the following FPU-related exceptions:

- Reserved instruction exception (core-generated):
  This exception happens when any FPU instruction is encountered while the FUCOP_EXIST.FPUE register field is 0. This is generated in the Andes core main pipeline.
- FPU disabled exception:
  This exception happens on FPU instructions when the FUCOP_EXIST.FPUE register field is 1 while FUCOP_CTL.FPUE register field is 0. This is generated in the Andes core main pipeline. Software can use the FUCOP_CTL .FPUE bit and this exception to determine if saving and restoring floating-point register on context switch is necessary or not.
- Reserved instruction exception (FPU-generated):
  For Andes cores that implement only one of SP or DP extensions, any non-SP or non-DP FPU instruction will generate this exception. Also, any floating-point register number that is outside the range of [0, Max-1] defined by the FPCFG.FREG field will generate this exception. This exception is generated in the FPU logic.
- Denorm input exception:
  For Andes FPU hardware that does not support denormalized number handling, the Denorm input exception is generated whenever a denormalized operand is encountered and the required denormalized computation is not supported for FPU arithmetic, data format conversion, and compare instructions when the Flush-to-Zero mode in FPCSR is not turned

on. Please see the result table in the descriptions for each relevant instruction for the exact operand value combinations that can generate this exception.

Once this exception is generated, the Denorm input exception handler will then perform the floating-point operation using the denormalized operand(s) and deliver the result in software. The instructions that do not generate Denorm input exception when encountering a denormalized operand are listed in Table 1.

1.5. Non-IEEE Flush-to-Zero Mode

For Andes hardware that does not support denormalized numbers, the denormalized number handling is done in software through denorm and underflow exceptions. This may cause significant performance slowdown if denormalized numbers are frequently encountered. If denormalized numbers can be treated as zeros for both inputs and outputs without causing any result accuracy problem in an application, an Andes hardware supported Flush-to-Zero mode can be used to speedup the handling of denormalized numbers encountered during the computation. However, this is not IEEE compliant and software must be certain that turning Flush-to-Zero mode on cannot spoil the functionality of an application before using it and the performance is certainly degraded by the denormalized number handling.

In Flush-to-Zero mode, the following non-IEEE compliant behaviors will be performed:

- All denormalized input operands of a floating-point arithmetic instruction will be turned into correctly signed zero before the calculation.
- All denormalized result will be turned into correctly signed zero.
- When Flush-to-Zero mode is enabled, the underflow exception trapping enable bit (FPCSR.UDFE) will be ignored and there will be no trapped underflow exception.
- When Flush-to-Zero mode is enabled, the denorm input exception will not happen.
- When a denormalized result is turned into correctly signed zero, both the underflow exception flag and the inexact exception flag will be set. If the inexact exception trapping enable bit (FPCSR.IEXE) is set, a trapped inexact exception will happen. However, most of the time, the inexact exception will be in non-trapped mode and only the inexact exception flag will be set by hardware. Note that flushing a denormalized input operand will not affect the underflow and the inexact exception flags.

The Flush-to-Zero mode does not affect the denormalized number (in/out) of the following instructions in Table 1.

Table 1. Instructions not affected by FTZ mode and not generating denorm input, underflow exceptions

FABSS/FABSD
FCMOVNS/FCMOVND
FCMOVZS/FCMOVZD
FCPYNSS/FCPYNSD
FCPYSS/FCPYSD
FS2SI/FS2SI.z
FS2UI/FS2UI.z
FD2SI/FD2SI.z
FD2UI/FD2UI.z

2. Floating point instruction summary

2.1. Arithmetic Instructions

These instructions generate IEEE 754 exceptions.

Table 2. Single-precision arithmetic instructions

Mnemonic	Instruction	Operation
FADDS fst, fsa, fsb	Add	fst = fsa + fsb
FSUBS fst, fsa, fsb	Subtract	fst = fsa - fsb
FMULS fst, fsa, fsb	Multiply	fst = fsa * fsb
FDIVS fst, fsa, fsb	Divide	fst = fsa / fsb
FSQRTS fst, fsa	Square Root	fst = SquareRoot(fsa)
FMADDS fst, fsa, fsb	Multiply and Add	fst = fst + fsa * fsb
FMSUBS fst, fsa, fsb	Multiply and Subtract	fst = fst - fsa * fsb
FNMADDS fst, fsa, fsb	Negate Multiply and Add	fst = - fst - fsa * fsb
FNMSUBS fst, fsa, fsb	Negate Multiply and Subtract	fst = - fst + fsa * fsb

Table 3. Double-precision arithmetic instructions

Mnemonic	Instruction	Operation
FADDD fdt, fda, fdb	Add	fdt = fda+ fdb
FSUBD fdt, fda, fdb	Subtract	fdt = fda - fdb
FMULD fdt, fda, fdb	Multiply	fdt = fda * fdb
FDIVD fdt, fda, fdb	Divide	fdt = fda / fdb
FSQRTD fdt, fda	Square Root	fdt = SquareRoot(fda)
FMADDD fdt, fda, fdb	Multiply and Add	fdt = fdt + fda * fdb
FMSUBD fdt, fda, fdb	Multiply and Subtract	fdt = fdt - fda * fdb
FNMADDD fdt, fda, fdb	Negate Multiply and Add	fdt = - fdt - fda * fdb
FNMSUBD fdt, fda, fdb	Negate Multiply and Subtract	fdt = - fdt + fda * fdb

2.2. Compare Instructions

Table 4. Single-precision compare instructions

Mnemonic	Instruction	Operation
FCMPEQS fst, fsa, fsb	Compare equal	No exception if fsa/fsb is QNaN fst = (fsa EQ fsb)? 1 : 0
FCMPLTS fst, fsa, fsb	Compare less than	No exception if fsa/fsb is QNaN fst = (fsa LT fsb)? 1 : 0
FCMPLES fst, fsa, fsb	Compare less than or equal	No exception if fsa/fsb is QNaN fst = (fsa LE fsb)? 1 : 0
FCMPUNS fst, fsa, fsb	Compare unordered	No exception if fsa/fsb is QNaN fst = (fsa UN fsb)? 1 : 0
FCMPEQS.e fst, fsa, fsb	Compare equal	Exception if fsa/fsb is QNaN fst = (fsa EQ fsb)? 1 : 0
FCMPLTS.e fst, fsa, fsb	Compare less than	Exception if fsa/fsb is QNaN fst = (fsa LT fsb)? 1 : 0
FCMPLES.e fst, fsa, fsb	Compare less than or equal	Exception if fsa/fsb is QNaN fst = (fsa LE fsb)? 1 : 0
FCMPUNS.e fst, fsa, fsb	Compare unordered	Exception if fsa/fsb is QNaN fst = (fsa UN fsb)? 1 : 0

Table 5. Double-precision compare instructions

Mnemonic	Instruction	Operation
FCMPEQD fst, fda, fdb	Compare equal	No exception if fda/fdb is QNaN fst = (fda EQ fdb)? 1 : 0
FCMPLTD fst, fda, fdb	Compare less than	No exception if fda/fdb is QNaN fst = (fda LT fdb)? 1 : 0
FCMPLED fst, fda, fdb	Compare less than or equal	No exception if fda/fdb is QNaN fst = (fda LE fdb)? 1 : 0

Mnemonic	Instruction	Operation
FCMPUND fst, fda, fdb	Compare unordered	No exception if fda/fdb is QNaN fst = (fda UN fdb)? 1 : 0
FCMPEQD.e fst, fda, fdb	Compare equal	Exception if fda/fdb is QNaN fst = (fda EQ fdb)? 1 : 0
FCMPLTD.e fst, fda, fdb	Compare less than	Exception if fda/fdb is QNaN fst = (fda LT fdb)? 1 : 0
FCMPLED.e fst, fda, fdb	Compare less than or equal	Exception if fda/fdb is QNaN fst = (fda LE fdb)? 1 : 0
FCMPUND.e fst, fda, fdb	Compare unordered	Exception if fda/fdb is QNaN fst = (fda UN fdb)? 1 : 0

2.3. Copy and Move Related Instructions

These instructions do not generate IEEE 754 exceptions.

Table 6. Copy/Move instructions common to both single-precision and double-precision

Mnemonic	Instruction	Operation
FMFCFG rt	Move from FPCFG	rt = FPCFG
FMFCSR rt	Move from FPCSR	rt = FPCSR
FMTCSR rt	Move to FPCSR	FPCSR = ra
FMFSR rt, fsa	Move from single-precision floating-point register	rt = fsa
FMTSR rt, fsa	Move to single-precision floating-point register	fsa = rt

Table 7. Single-precision copy/move instructions

Mnemonic	Instruction	Operation
FABSS fst, fsa	Absolute value	fst = ABS(fsa)
FCPYSS fst, fsa, fsb	Copy sign	fst = CONCAT(fsb(31),fsa(30,0))
FCPYNSS fst, fsa, fsb	Copy negative sign	fst = CONCAT(NOT(fsb(31)),fsa(30,0))
FCMOVZS fst, fsa, fsb	Conditional move on zero	fst = fsa if (fsb == 0)
FCMOVNS fst, fsa, fsb	Conditional move on not zero	fst = fsa if (fsb != 0)

Table 8. Double-precision copy/move instructions

Mnemonic	Instruction	Operation
FABSD fdt, fda	Absolute value	fdt = ABS(fda)
FCPYSD fdt, fda, fdb	Copy sign	fdt = CONCAT(fdb(63),fda(62,0))
FCPYNSD fdt, fda, fdb	Copy negative sign	fdt = CONCAT(NOT(fdb(63)),fda(62,0))
FCMOVZD fdt, fda, fsb	Conditional move on zero	fdt = fda if (fsb == 0)

Mnemonic	Instruction	Operation
FCMOVND fdt, fda, fsb	Conditional move on not zero	fdt = fda if (fsb != 0)

FMFDR rt, fda	Move from double-precision floating-point register	if (PSW.BE == 1) {
		rt.pair.even = fda(63,32);
		rt.pair.odd = fda(31,0);
		} else {
		rt.pair.odd = fda(63,32);
		rt.pair.even = fda(31,0);
		}

FMTDR rt, fda	Move to double-precision floating-point register	if (PSW.BE == 1) {
		fda(63,32) = rt.pair.even;
		fda(31,0) = rt.pair.odd;
		} else {
		fda(63,32) = rt.pair.odd;
		fda(31,0) = rt.pair.even;
		}

2.4. Load and Store Instructions

Table 9. Load/Store instructions common to both Single-precision and Double-precision

Mnemonic	Instruction	Operation

FLS fst, [ra + (rb << sv)]	Load single-precision data	address = ra + (rb << sv)
		fst = Word-memory(address)

FLS.bi fst, [ra], (rb << sv)	Load single-precision data with base update	address = ra
		fst = Word-memory(address)
		ra = ra + (rb << sv)

FLSI fst, [ra + (imm12s << 2)]	Load single-precision data immediate	address = ra + (imm12s << 2)
		fst = Word-memory(address)

FLSI.bi fst, [ra], (imm12s << 2)	Load single-precision data immediate with base update	address = ra
		fst = Word-memory(address)
		ra = ra + (imm12s << 2)

FSS fst, [ra + (rb << sv)]	Store single-precision data	address = ra + (rb << sv) Word-memory(address) = fst

FSS.bi fst, [ra], (rb << sv)	Store single-precision data with base update	address = ra
		Word-memory(address) = fst
		ra = ra + (rb << sv)

FSSI fst, [ra + (imm12s << 2)]	Store single-precision data immediate	address = ra + (imm12s << 2) Word-memory(address) = fst

FSSI.bi fst, [ra], (imm12s << 2)	Store single-precision data immediate with base update	address = ra
		Word-memory(address) = fst
		ra = ra + (imm12s << 2)

Table 10. Double-precision load/store instructions

Mnemonic	Instruction	Operation

FLD fdt, [ra + (rb << sv)]	Load double-precision data	address = ra + (rb << sv)
		fdt = DWord-memory(address)

FLD.bi fdt, [ra] + (rb << sv)	Load double -precision data with base update	address = ra
		fdt = DWord-memory(address)
		ra = ra + (rb << sv)

FLDI fdt, [ra + (imm12s << 2)]	Load double-precision data immediate	address = ra + (imm12s << 2)
		fdt = DWord-memory(address)

FLDI.bi fdt, [ra], (imm12s << 2)	Load double-precision data immediate with base update	address = ra
		fdt = DWord-memory(address)
		ra = ra + (imm12s << 2)

FSD fdt, [ra + (rb << sv)]	Store double -precision data	address = ra + (rb << sv) DWord-memory(address) = fdt

FSD.bi fdt, [ra] + (rb << sv)	Store double -precision data with base update	address = ra
		DWord-memory(address) = fdt
		ra = ra + (rb << sv)

FSDI fdt, [ra + (imm12s << 2)]	Store double -precision data immediate	address = ra + (imm12s << 2) DWord-memory(address) = fdt

FSDI.bi fdt, [ra], (imm12s << 2)	Store double -precision data immediate with base update	address = ra
		DWord-memory(address) = fdt
		ra = ra + (imm12s << 2)

2.5. Data Format Conversion Instructions

Table 11. Single-precision data format conversion instructions

Mnemonic	Instruction	Operation
FUI2S fst, fsa	Convert unsigned integer to single-precision	
FSI2S fst, fsa	Convert signed integer to single-precision	
FS2UI fst, fsa	Convert single-precision to unsigned integer with FPCSR rounding mode	
FS2SI fst, fsa	Convert single-precision to signed integer with FPCSR rounding mode	
FS2UI.z fst, fsa	Convert single-precision to unsigned integer with "→ 0" rounding mode	
FS2SI.z fst, fsa	Convert single-precision to signed integer with "→ 0" rounding mode	
FS2D fdt, fsa	Convert single-precision to double-precision	

Table 12. Double-precision data format conversion instructions

Mnemonic	Instruction	Operation
FUI2D fdt, fsa	Convert unsigned integer to double-precision	
FSI2D fdt, fsa	Convert signed integer to double-precision	
FD2UI fst, fda	Convert double-precision to unsigned integer with FPCSR rounding mode	

Mnemonic	Instruction	Operation
FD2SI fst, fda	Convert double-precision to signed integer with FPCSR rounding mode	
FD2UI.z fst, fda	Convert double-precision to unsigned integer with "→ 0" rounding mode	
FD2SI.z fst, fda	Convert double-precision to signed integer with "→ 0" rounding mode	
FD2S fst, fda	Convert double-precision to double-precision	

3. Floating point detailed instruction description

- SI(FRa) : the data in floating-point register FRa interpreted as Signed Integer.
- UI(FRa) : the data in floating-point register FRa interpreted as Unsigned Integer.

3.1. Instructions Common to Both SP and DP extensions

These instructions will be present if either SP or DP extension is implemented.

Instruction	Description
FMFCSR	Move from FPCSR
FMTCSR	Move to FPCSR
FMFCFG	Move from FPCFG
FMFSR	Move from single-precision floating-point register
FMTSR	Move to single-precision floating-point register
FLS(.bi)	Load SP floating-point register (with base update)
FLSI(.bi)	Load SP floating-point register immediate (with base update)
FSS(.bi)	Store SP floating-point register (with base update)
FSSI(.bi)	Store SP floating-point register immediate (with base update)

These instructions are in this category to simplify floating-point SP and DP ABIs into a unified ABI.

Instruction	Description
FMFDR	Move from double-precision floating-point register
FMTDR	Move to double-precision floating-point register
FLD(.bi)	Load DP floating-point register (with base update)
FLDI(.bi)	Load DP floating-point register immediate (with base update)
FSD(.bi)	Store DP floating-point register (with base update)
FSDI(.bi)	Store DP floating-point register immediate (with base update)
FCPYSD	DP floating-point copy sign

FCPYSD (Floating-point Copy Sign Double-precision)

Type: 32-Bit floating-point SP or DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FDt	FDa	FDb	0011 FCPYSD	00 CP0	1000 FD1

Syntax: FCPYSD FDt, FDa, FDb

Purpose: Generate a floating-point value by copying the sign of a floating-point value in one register to a value in another register.

Description: The sign of the floating-point value in FDb is copied to the floating point value in FDa. The floating point result is written to FDt.

No checking of NaN operands is performed for this instruction.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
FDt = CONCAT(FDb(63),FDa(62,0));
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (Register out-of-range)

Floating Point Exceptions: None

Privilege level: All

Note:

- Move between floating-point registers can be performed with "FCPYSD FDt, FDa, FDa".

FLD (Floating-point Load Double-precision Data)

Type: 32-Bit Floating-point SP or DP extension

Format:

FLD

31	30 25	24 20	19 15	14 10	9 8	7	6	5 4	3 0
0	COP 110101	FDt	Ra	Rb	sv	0 bi	0	00 CP0	0011 FLD

FLD.bi

31	30 25	24 20	19 15	14 10	9 8	7	6	5 4	3 0
0	COP 110101	FDt	Ra	Rb	sv	1 bi	0	00 CP0	0011 FLD

Syntax:

FLD	FDt, [Ra + (Rb << sv)]
FLD.bi	FDt, [Ra], (Rb << sv)

Purpose: To load a 64-bit double-precision floating-point data from memory into a double-precision floating-point register.

Description: This instruction loads a double-precision floating-point data from the memory into the floating-point register FDt. Two different forms are used to specify the memory address. The regular form uses Ra + (Rb << sv) as its memory address while the .bi form uses Ra. For the .bi form, the Ra register will be updated with the Ra + (Rb << sv) value after the memory load operation.

The memory address has to be double-word-aligned. Otherwise, a Data Alignment Check exception will be generated.

The floating-point register number is in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
Addr = Ra + (Rb << sv);
If (.bi form) {
   Vaddr = Ra;
} else {
   Vaddr = Addr;
}
if (!Word_Aligned(Vaddr)) {
   Generate_Exception(Data_alignment_check);
}
(PAddr, Attributes) = Address_Translation(Vaddr, PSW.DT);
Excep_status = Page_Exception(Attributes, PSW.POM, LOAD);
If (Excep_status == NO_EXCEPTION) {
   Wdata(63,0) = Load_Memory(PAddr, DOUBLEWORD, Attributes);
   FDt = Wdata(63,0);
   If (.bi form) { Ra = Addr; }
} else {
   Generate_Exception(Excep_status);
}
```

Exceptions: TLB fill, Non-leaf PTE not present, Leaf PTE not present, Read protection, Page modified, Access bit, TLB VLPT miss, Machine error, Data alignment check,
Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (Register out-of-range)

Privilege level: All

Note:

FLDI (Floating-point Load Double-precision Data Immediate)

Type: 32-Bit Floating-point SP or DP extension

Format:

FLDI

31	30 25	24 20	19 15	14 13	12	11 0
0	LDC 011010	FDt	Ra	00 CP0	bi 0	imm12s

FLDI.bi

31	30 25	24 20	19 15	14 13	12	11 0
0	LDC 011010	FDt	Ra	00 CP0	bi 1	imm12s

Syntax: FLDI FDt, [Ra + (imm12s << 2)]
FLDI.bi FDt, [Ra], (imm12s << 2)

Purpose: To load a 64-bit double-precision floating-point data from memory into a double-precision floating-point register.

Description: This instruction loads a double-precision floating-point data from the memory into the floating-point register FDt. Two different forms are used to specify the memory address. The regular form uses Ra + (imm12s << 2) as its memory address while the .bi form uses Ra. For the .bi form, the Ra register will be updated with the Ra + (imm12s << 2) value after the memory load operation.

The memory address has to be double-word-aligned. Otherwise, a Data Alignment Check exception will be generated.

The floating-point register number is in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
Addr = Ra + Sign_Extend(imm12s << 2);
If (.bi form) {
   Vaddr = Ra;
} else {
   Vaddr = Addr;
}
if (!Word_Aligned(Vaddr)) {
   Generate_Exception(Data_alignment_check);
}
(PAddr, Attributes) = Address_Translation(Vaddr, PSW.DT);
Excep_status = Page_Exception(Attributes, PSW.POM, LOAD);
If (Excep_status == NO_EXCEPTION) {
   Wdata(63,0) = Load_Memory(PAddr, DOUBLEWORD, Attributes);
   FDt = Wdata(63,0);
   If (.bi form) { Ra = Addr; }
} else {
   Generate_Exception(Excep_status);
}
```

Exceptions: TLB fill, Non-leaf PTE not present, Leaf PTE not present, Read protection, Page modified, Access bit, TLB VLPT miss, Machine error, Data alignment check,
Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (Register out-of-range)

Privilege level: All

Note:

Implementation Note: From the coprocessor's point of view, this instruction is equivalent to FLD instruction. So these two instructions can be sent to the coprocessor using the same command encoding.

FLS (Floating-point Load Single-precision Data)

Type: 32-Bit floating-point SP or DP V1 extension

Format:

FLS

31	30 25	24 20	19 15	14 10	9 8	7	6	5 4	3 0
0	COP 110101	FSt	Ra	Rb	sv	0 bi	0	00 CP0	0010 FLS

FLS.bi

31	30 25	24 20	19 15	14 10	9 8	7	6	5 4	3 0
0	COP 110101	FSt	Ra	Rb	sv	1 bi	0	00 CP0	0010 FLS

Syntax:

```
FLS       FSt, [Ra + (Rb << sv)]
FLS.bi    FSt, [Ra], (Rb << sv)
```

Purpose: To load a 32-bit single-precision floating-point data from memory into a floating-point register.

Description: This instruction loads a single-precision floating-point data from the memory into the floating-point register FSt. Two different forms are used to specify the memory address. The regular form uses Ra + (Rb << sv) as its memory address while the .bi form uses Ra. For the .bi form, the Ra register will be updated with the Ra + (Rb << sv) value after the memory load operation.

The memory address has to be word-aligned. Otherwise, a Data Alignment Check exception will be generated.

The floating-point register number is in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
Addr = Ra + (Rb << sv);
If (.bi form) {
   Vaddr = Ra;
} else {
   Vaddr = Addr;
}
if (!Word_Aligned(Vaddr)) {
   Generate_Exception(Data_alignment_check);
}
(PAddr, Attributes) = Address_Translation(Vaddr, PSW.DT);
Excep_status = Page_Exception(Attributes, PSW.POM, LOAD);
If (Excep_status == NO_EXCEPTION) {
   Wdata(31,0) = Load_Memory(PAddr, WORD, Attributes);
   FSt = Wdata(31,0);
   If (.bi form) { Ra = Addr; }
} else {
   Generate_Exception(Excep_status);
}
```

Exceptions: TLB fill, Non-leaf PTE not present, Leaf PTE not present, Read protection, Page modified, Access bit, TLB VLPT miss, Machine error, Data alignment check,
Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (Register out-of-range)

Privilege level: All

Note:

FLSI (Floating-point Load Single-precision Data Immediate)

Type: 32-Bit floating-point SP or DP V1 extension

Format:

FLSI

31	30 25	24 20	19 15	14 13	12	11 0
0	LWC 011000	FSt	Ra	00 CP0	bi 0	imm12s

FLSI.bi

31	30 25	24 20	19 15	14 13	12	11 0
0	LWC 011000	FSt	Ra	00 CP0	bi 1	imm12s

Syntax: FLSI FSt, [Ra + (imm12s << 2)]
FLSI.bi FSt, [Ra], (imm12s << 2)

Purpose: To load a 32-bit single-precision floating-point data from memory into a floating-point register.

Description: This instruction loads a single-precision floating-point data from the memory into the floating-point register FSt. Two different forms are used to specify the memory address. The regular form uses Ra + (imm12s << 2) as its memory address while the .bi form uses Ra. For the .bi form, the Ra register will be updated with the Ra + (imm12s << 2) value after the memory load operation.

The memory address has to be word-aligned. Otherwise, a Data Alignment Check exception will be generated.

The floating-point register number is in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
Addr = Ra + Sign_Extend(imm12s << 2);
If (.bi form) {
   Vaddr = Ra;
} else {
   Vaddr = Addr;
}
if (!Word_Aligned(Vaddr)) {
   Generate_Exception(Data_alignment_check);
}
(PAddr, Attributes) = Address_Translation(Vaddr, PSW.DT);
Excep_status = Page_Exception(Attributes, PSW.POM, LOAD);
If (Excep_status == NO_EXCEPTION) {
   Wdata(31,0) = Load_Memory(PAddr, WORD, Attributes);
   FSt = Wdata(31,0);
   If (.bi form) { Ra = Addr; }
} else {
   Generate_Exception(Excep_status);
}
```

Exceptions: TLB fill, Non-leaf PTE not present, Leaf PTE not present, Read protection, Page modified, Access bit, TLB VLPT miss, Machine error, Data alignment check,
Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (Register out-of-range)

Privilege level: All

Note:

Implementation Note: From the coprocessor's point of view, this instruction is equivalent to FLS instruction. So these two instructions can be sent to the coprocessor using the same command encoding.

FMFCFG (Floating-point Move From Configuration register)

Type: 32-Bit Floating point SP or DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	Rt	00000	00000 CFG	1100 XR	00 CP0	0001 MFCP

Syntax: FMFCFG Rt

Purpose: Move the FPCFG register to a general purpose register.

Description: Transfer the content of FPCFG register to a general purpose register Rt.

Operations:

```
Rt = FPCFG;
```

Exceptions: Reserved instruction (When FPU extension is not implemented)
FPU disabled (When the use of FPU is not enabled)

Floating Point Exceptions: None

Privilege level: All

Note:

FMFCSR (Floating-point Move From FPCSR)

Type: 32-Bit Floating point SP or DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	Rt	00000	00001 CSR	1100 XR	00 CP0	0001 MFCP

Syntax: FMFCSR   Rt

Purpose: Move the FPCSR to a general purpose register.

Description: Transfer the content of FPCSR to a general register Rt.

Operations:

```
Rt = FPCSR;
```

Exceptions: Reserved instruction (When FPU extension is not implemented)
FPU disabled (When the use of FPU is not enabled)

Floating Point Exceptions: None

Privilege level: All

Note:

FMFDR (Floating-point Move From Double-precision FR)

Type: 32-Bit Floating point SP or DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	Rt	FDa	00000	0001 DR	00 CP0	0001 MFCP

Syntax: FMFDR Rt, FDa

Purpose: Move the content of a 64-bit double-precision floating-point register to two 32-bit even/odd pair of general purpose registers.

Description: Transfer the content of the 64-bit double-precision floating-point register FDa to one even/odd pair of general registers containing Rt. Rt(4,1) determines the even/odd pair group of the two registers. When the data endian is *big*, the even register of the pair contains the high 32-bit of the FDa content and the odd register of the pair contains the low 32-bit of the FDa content. When the data endian is *little*, the odd register of the pair contains the high 32-bit of the FDa content and the even register of the pair contains the low 32-bit of the FDa content.

The floating-point register number is in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
If (PSW.BE == 1) {
  [Rt(4,1).0(0)](31,0) = FDa(63,32); // even register
  [Rt(4,1).1(0)](31,0) = FDa(31,0);  // odd register
} else {
  [Rt(4,1).1(0)](31,0) = FDa(63,32); // odd register
  [Rt(4,1).0(0)](31,0) = FDa(31,0);  // even register
}
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (Register out-of-range)

Floating Point Exceptions: None

Privilege level: All

Note:

FMFSR (Floating-point Move From Single-precision FR)

Type: 32-Bit Floating point SP or DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	Rt	FSa	00000	0000 SR	00 CP0	0001 MFCP

Syntax: FMFSR Rt, Fsa

Purpose: Move the content of a 32-bit single-precision floating-point register to a 32-bit general purpose register.

Description: Transfer the content of the 32-bit single-precision floating-point register FSa to a general register Rt.

The floating-point register number is in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
    Rt = FSa;
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (Register out-of-range)

Floating Point Exceptions: None

Privilege level: All

Note:

FMTCSR (Floating-point Move To FPCSR)

Type: 32-Bit Floating point SP or DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	Rt	00000	00001 CSR	1100 XR	00 CP0	1001 MTCP

Syntax: FMTCSR Rt

Purpose: Move to the FPCSR from a general purpose register.

Description: Transfer the content of Ra to the FPCSR. A DSB instruction is needed after the FMTCSR instruction in order for the following floating-point instruction to see the updated FPCSR content and its side effects.

Operations:

```
FPCSR = Rt;
```

Exceptions: Reserved instruction (When FPU extension is not implemented)
FPU disabled (When the use of FPU is not enabled)

Floating Point Exceptions: None

Privilege level: All

Note:

FMTDR (Floating-point Move To Double-precision FR)

Type: 32-Bit Floating point SP or DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP	Rt	FDa	00000	0001	00	1001
	110101				DR	CP0	MTCP

Syntax: FMTDR Rt, Fda

Purpose: Move to a 64-bit double-precision floating-point register from two 32-bit even/odd pair of general purpose registers.

Description: Transfer the contents of one even/odd pair of general purpose registers containing Rt to the 64-bit double-precision floating-point register FDa. Rt(4,1) determines the even/odd pair group of the two registers. When the data endian is *big*, the high 32-bit of FDa is from the even register and the low 32-bit of FDa is from the odd register. When the data endian is *little*, the high 32-bit of FDa is from the odd register and the low 32-bit of FDa is from the even register.

The floating-point register number is in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
If (PSW.BE == 1) {
  FDa(63,32) = [Rt(4,1).0(0)](31,0); // even register
  FDa(31,0)  = [Rt(4,1).1(0)](31,0); // odd register
} else {
  FDa(63,32) = [Rt(4,1).1(0)](31,0); // odd register
  FDa(31,0)  = [Rt(4,1).0(0)](31,0); // even register
}
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (Register out-of-range)

Floating Point Exceptions: None

Privilege level: All

Note:

FMTSR (Floating-point Move To Single-precision FR)

Type: 32-Bit Floating point SP or DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	Rt	FSa	00000	0000 SR	00 CP0	1001 MTCP

Syntax: FMTSR Rt, Fsa

Purpose: Move to a 32-bit single-precision floating-point register from a 32-bit general purpose register.

Description: Transfer the content of Rt to the 32-bit single-precision floating-point register FSa.

The floating-point register number is in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
FSa = Rt;
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (Register out-of-range)

Floating Point Exceptions: None

Privilege level: All

Note:

FSD (Floating-point Store Double-precision Data)

Type: 32-Bit Floating-point SP or DP extension

Format:

FSD

31	30 25	24 20	19 15	14 10	9 8	7	6	5 4	3 0
0	COP 110101	FDt	Ra	Rb	sv	0 bi	0	00 CP0	1011 FSD

FSD.bi

31	30 25	24 20	19 15	14 10	9 8	7	6	5 4	3 0
0	COP 110101	FDt	Ra	Rb	sv	1 bi	0	00 CP0	1011 FSD

Syntax:		
	FSD	FDt, [Ra + (Rb << sv)]
	FSD.bi	FDt, [Ra], (Rb << sv)

Purpose: To store a 64-bit double-precision floating-point data from a double-precision floating-point register into memory.

Description: This instruction stores a double-precision floating-point data from the floating-point register FDt into the memory. Two different forms are used to specify the memory address. The regular form uses Ra + (Rb << sv) as its memory address while the .bi form uses Ra. For the .bi form, the Ra register will be updated with the Ra + (Rb << sv) value after the memory store operation.

The memory address has to be double-word-aligned. Otherwise, a Data Alignment Check exception will be generated.

The floating-point register number is in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
Addr = Ra + (Rb << sv);
If (.bi form) {
   Vaddr = Ra;
} else {
   Vaddr = Addr;
}
if (!Word_Aligned(Vaddr)) {
   Generate_Exception(Data_alignment_check);
}
(PAddr, Attributes) = Address_Translation(Vaddr, PSW.DT);
Excep_status = Page_Exception(Attributes, UserMode, STORE);
If (Excep_status == NO_EXCEPTION) {
   Ddata = FDt;
   Store_Memory(PAddr, DOUBLEWORD, Attributes, Ddata);
   If (.bi form) { Ra = Addr; }
} else {
   Generate_Exception(Excep_status);
}
```

Exceptions: TLB fill, Non-leaf PTE not present, Leaf PTE not present, Read protection, Page modified, Access bit, TLB VLPT miss, Machine error, Data alignment check,
Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (Register out-of-range)

Privilege level: All

Note:

FSDI (Floating-point Store Double-precision Data Immediate)

Type: 32-Bit Floating-point SP or DP extension

Format:

FSD

31	30 25	24 20	19 15	14 13	12	11 0
0	SDC 011011	FDt	Ra	00 CP0	bi 0	imm12s

FSD.bi

31	30 25	24 20	19 15	14 13	12	11 0
0	SDC 011011	FDt	Ra	00 CP0	bi 1	imm12s

Syntax:

```
FSDI      FDt, [Ra + (imm12s << 2)]
FSDI.bi   FDt, [Ra], (imm12s << 2)
```

Purpose: To store a 64-bit double-precision floating-point data from a double-precision floating-point register into memory.

Description: This instruction stores a double-precision floating-point data from the floating-point register FDt into the memory. Two different forms are used to specify the memory address. The regular form uses Ra + (imm12s << 2) as its memory address while the .bi form uses Ra. For the .bi form, the Ra register will be updated with the Ra + (imm12s << 2) value after the memory store operation.

The memory address has to be double-word-aligned. Otherwise, a Data Alignment Check exception will be generated.

The floating-point register number is in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
Addr = Ra + Sign_Extend(imm12s << 2);
If (.bi form) {
   Vaddr = Ra;
} else {
   Vaddr = Addr;
}
if (!Word_Aligned(Vaddr)) {
   Generate_Exception(Data_alignment_check);
}
(PAddr, Attributes) = Address_Translation(Vaddr, PSW.DT);
Excep_status = Page_Exception(Attributes, UserMode, STORE);
If (Excep_status == NO_EXCEPTION) {
   Ddata = FDt;
   Store_Memory(PAddr, DOUBLEWORD, Attributes, Ddata);
   If (.bi form) { Ra = Addr; }
} else {
   Generate_Exception(Excep_status);
}
```

Exceptions: TLB fill, Non-leaf PTE not present, Leaf PTE not present, Read protection, Page modified, Access bit, TLB VLPT miss, Machine error, Data alignment check,
Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (Register out-of-range)

Privilege level: All

Note:

Implementation Note: From the coprocessor's point of view, this instruction is equivalent to FSD instruction. So these two instructions can be sent to the coprocessor using the same command encoding.

FSS (Floating-point Store Single-precision Data)

Type: 32-Bit floating-point SP or DP V1 extension

Format:

FSS

31	30 25	24 20	19 15	14 10	9 8	7	6	5 4	3 0
0	COP 110101	FSt	Ra	Rb	sv	0 bi	0	00 CP0	1010 FSS

FSS.bi

31	30 25	24 20	19 15	14 10	9 8	7	6	5 4	3 0
0	COP 110101	FSt	Ra	Rb	sv	1 bi	0	00 CP0	1010 FSS

Syntax:		
	FSS	FSt, [Ra + (Rb << sv)]
	FSS.bi	FSt, [Ra], (Rb << sv)

Purpose: To store a 32-bit single-precision floating-point data from a floating-point register into memory.

Description: This instruction stores a single-precision floating-point data from the floating-point register FSt into the memory. Two different forms are used to specify the memory address. The regular form uses Ra + (Rb << sv) as its memory address while the .bi form uses Ra. For the .bi form, the Ra register will be updated with the Ra + (Rb << sv) value after the memory store operation.

The memory address has to be word-aligned. Otherwise, a Data Alignment Check exception will be generated.

The floating-point register number is in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
Addr = Ra + (Rb << sv);
If (.bi form) {
   Vaddr = Ra;
} else {
   Vaddr = Addr;
}
if (!Word_Aligned(Vaddr)) {
   Generate_Exception(Data_alignment_check);
}
(PAddr, Attributes) = Address_Translation(Vaddr, PSW.DT);
Excep_status = Page_Exception(Attributes, UserMode, STORE);
If (Excep_status == NO_EXCEPTION) {
   Store_Memory(PAddr, WORD, Attributes, FSt);
   If (.bi form) { Ra = Addr; }
} else {
   Generate_Exception(Excep_status);
}
```

Exceptions: TLB fill, Non-leaf PTE not present, Leaf PTE not present, Read protection, Page modified, Access bit, TLB VLPT miss, Machine error, Data alignment check,
Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (Register out-of-range)

Privilege level: All

Note:

FSSI (Floating-point Store Single-precision Data Immediate)

Type: 32-Bit floating-point SP or DP V1 extension

Format:

FSSI

31	30 25	24 20	19 15	14 13	12	11 0
0	SWC 011001	FSt	Ra	00 CP0	bi 0	imm12s

FSSI.bi

31	30 25	24 20	19 15	14 13	12	11 0
0	SWC 011001	FSt	Ra	00 CP0	bi 1	imm12s

Syntax: FSSI FSt, [Ra + (imm12s << 2)]
FSSI.bi FSt, [Ra], (imm12s << 2)

Purpose: To store a 32-bit single-precision floating-point data from a floating-point register into memory.

Description: This instruction stores a single-precision floating-point data from the floating-point register FSt into the memory. Two different forms are used to specify the memory address. The regular form uses Ra + (imm12s << 2) as its memory address while the .bi form uses Ra. For the .bi form, the Ra register will be updated with the Ra + (imm12s << 2) value after the memory store operation.

The memory address has to be word-aligned. Otherwise, a Data Alignment Check exception will be generated.

The floating-point register number is in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
Addr = Ra + Sign_Extend(imm12s << 2);
If (.bi form) {
   Vaddr = Ra;
} else {
   Vaddr = Addr;
}
if (!Word_Aligned(Vaddr)) {
   Generate_Exception(Data_alignment_check);
}
(PAddr, Attributes) = Address_Translation(Vaddr, PSW.DT);
Excep_status = Page_Exception(Attributes, UserMode, STORE);
If (Excep_status == NO_EXCEPTION) {
   Store_Memory(PAddr, WORD, Attributes, FSt);
   If (.bi form) { Ra = Addr; }
} else {
   Generate_Exception(Excep_status);
}
```

Exceptions: TLB fill, Non-leaf PTE not present, Leaf PTE not present, Read protection, Page modified, Access bit, TLB VLPT miss, Machine error, Data alignment check,
Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (Register out-of-range)

Privilege level: All

Note:

Implementation Note: From the coprocessor's point of view, this instruction is equivalent to FSS instruction. So these two instructions can be sent to the coprocessor using the same command encoding.

3.2. Instructions When Both SP and DP extensions Exist

These instructions will be present if both SP and DP extensions are implemented.

Instruction	Description
FS2D	Convert single-precision to double-precision
FD2S	Convert double-precision to single-precision

FS2D (Floating Point Convert From SP To DP)

Type: 32-Bit floating-point SP and DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FDt	FSa	00000 FS2D	1111 F2OP	00 CP0	0000 FS1

Syntax: FS2D FDt, Fsa

Purpose: Convert a single-precision floating-point value to a double-precision floating-point value.

Description: The single-precision floating-point data in FSa is converted to a double-precision floating-point value, rounded based on the rounding mode in the FPCSR register. The result is written to FDt register. This operation is always exact.

The floating-point register numbers (FSa and FDt) are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
    FDt = ConvertSP2DP(FSa);
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When both SP and DP extensions are not implemented, Register out-of-range)
Denorm input (for non-denorm-support FPU)

Floating Point Exceptions: Invalid operation

Privilege level: All

Note:

- An exception handler can convert a single-precision denormal value into the corresponding double-precision value by adding 896 to the exponent and normalizing.

FD2S (Floating Point Convert From DP To SP)

Type: 32-Bit floating-point SP and DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FDa	00000 FD2S	1111 F2OP	00 CP0	1000 FD1

Syntax: FD2S FSt, Fda

Purpose: Convert a double-precision floating-point value to a single-precision floating-point value.

Description: The double-precision floating-point data in FDa register is converted to a single-precision floating-point value, rounded based on the rounding mode in the FPCSR register. The result is written to FSt.

The floating-point register numbers (FSt and FDa) are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
FSt = ConvertDP2SP(FDa);
```

Exceptions: Reserved instruction (When FPU is not implemented)

FPU disabled (When the use of FPU is not enabled)

Reserved instruction (When both SP and DP extension is not implemented, Register out-of-range)

Denorm input (for non-denorm-support FPU)

Floating Point Exceptions: Invalid operation, Overflow, Underflow, Inexact

Privilege level: All

Note:

3.3. Single Precision Extension Instructions

These instructions will be present if SP extension is implemented.

Instruction	Description
FADDS	SP floating-point addition
FSUBS	SP floating-point subtraction
FMULS	SP floating-point multiplication
FDIVS	SP floating-point division
FABSS	SP floating-point absolute value
FSQRTS	SP floating-point square root
FCPYSS	SP floating-point copy sign
FCPYNSS	SP floating-point copy negative sign
FCMPxxS	SP floating-point compare (no IVO exception on QNAN)
FCMPxxS.e	SP floating-point compare (IVO exception on QNAN)
FCMOVZS	SP floating-point conditional move on zero
FCMOVNS	SP floating-point conditional move on not zero
FLS(I)(.bi)	Load SP floating-point register (immediate) (with base update)
FSS(I)(.bi)	Store SP floating-point register (immediate) (with base update)
FMFSR	Move from SP floating-point register to a general purpose register
FMTSR	Move to SP floating-point register from a general purpose register
FUI2S	Convert unsigned integer to SP floating-point
FSI2S	Convert signed integer to SP floating-point
FS2UI	Convert SP floating-point to unsigned integer (FPCSR rounding mode)
FS2UI.z	Convert SP floating-point to unsigned integer (toward zero rounding mode)
FS2SI	Convert SP floating-point to signed integer (FPCSR rounding mode)

Instruction	Description
FS2SI.z	Convert SP floating-point to signed integer (toward zero rounding mode)

These instructions will be present if SP extension is implemented and FMA option is supported.

Instruction	Description
FMADDS	SP floating-point addition multiply and add
FMSUBS	SP floating-point subtraction multiply and subtract
FNMADDS	SP floating-point negate of “multiply and add”
FNMSUBS	SP floating-point negate of “multiply and subtract”

FABSS (Floating-point Absolute Single-precision)

Type: 32-Bit floating-point SP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	00101 FABSS	1111 F2OP	00 CP0	0000 FS1

Syntax: FABSS FSt, Fsa

Purpose: Compute the absolute value of a single-precision floating-point data.

Description: The absolute value of the single-precision floating-point data in FSa is calculated and written to FSt.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

This instruction is non-arithmetic and it does not generate any IEEE 754 exceptions.

Operations:

```
if (FSa <= -0) {
   FSt = -FSa;
} else {
   FSt = FSa;
}
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When SP extension is not implemented, Register out-of-range)

Floating-point Exceptions: None

Privilege level: All

Note:

FADDS (Floating-point Addition Single-precision)

Type: 32-Bit floating-point SP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	FSb	0000 FADDS	00 CP0	0000 FS1

Syntax: FADDS FSt, FSa, FSb

Purpose: Add the single-precision floating-point values in two registers.

Description: The single-precision floating-point value in FSa is added with the single-precision floating-point value in FSb. And the floating-point result is written to FSt.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

The following table shows the results obtained when adding various types of numbers.

Table 13. FADDS results

		FSa								
		-∞	**-N**	**-DN**	**-0**	**+0**	**+DN**	**+N**	**+∞**	**NaN**
	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞	IVO	NaNa
	-N	-∞	-F	*	FSb	FSb	*	±F	+∞	NaNa
FSb	**-DN**	-∞	*	*	*	*	*	*	+∞	NaNa
	-0	-∞	FSa	*	-0	±0	*	FSa	+∞	NaNa
	+0	-∞	FSa	*	±0	+0	*	FSa	+∞	NaNa
	+DN	-∞	*	*	*	*	*	*	+∞	NaNa

	FSa									
	+N	-∞	±F	*	FSb	FSb	*	+F	+∞	NaNa
	+∞	IVO	+∞	+∞	+∞	+∞	+∞	+∞	+∞	NaNa
	NaN	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNc

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
F	Means a finite floating-point value (N, DN) which may involve underflow, overflow handling to become (N, DN, ∞).
IVO	Means invalid operation exception. If trapping is disabled, deliver the default QNaN as the result.
NaNa	Means a QNaN from FSa or a QNaN converted from a SNaN from FSa by changing its most significant fraction bit from 0 to 1.
NaNb	Means a QNaN from FSb or a QNaN converted from a SNaN from FSb by changing its most significant fraction bit from 0 to 1.
NaNc	Means a QNaN converted from ● A SNaN from FSa or from FSb if FSa is a QNaN ● A QNaN from FSa if FSb is also a QNaN
*	Indicates F (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)

Note that IEEE-754 standard states that

"*When the sum of two operands with opposite signs is exactly zero, the sign of that sum shall be + in all rounding modes except round toward -∞, in which mode that sign shall be –. However, x + x retains the same sign as x even when x is zero.*"

Operations:

```
FSt = FSa + FSb;
```

Exceptions: Reserved instruction (When FPU is not implemented)

FPU disabled (When the use of FPU is not enabled)

Reserved instruction (When SP extension is not implemented, Register out-of-range)

Denorm input (for non-denorm-support FPU)

Floating-point Exceptions: Invalid operation, Inexact, Overflow, Underflow

Privilege level: All

Note:

FCMOVNS (Floating-point Conditional Move on Not Zero Single-precision)

Type: 32-Bit floating-point SP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	FSb	0110 FCMOVNS	00 CP0	0000 FS1

Syntax: FCMOVNS FSt, FSa, FSb

Purpose: Move the content of a single-precision floating-point register based on a not zero condition stored in a single-precision floating-point register.

Description: If FSb is not integer-zero (Note “integer-zero” means all 32 bits are 0), then move the single-precision floating-point value in FSa to FSt.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

This instruction is non-arithmetic and it does not generate any IEEE 754 exceptions.

Operations:

```
If (FSb != 0(31,0)) {
   FSt = FSa;
}
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When SP extension is not implemented, Register out-of-range)

Floating-point Exceptions: None

Privilege level: All

Note:

FCMOVZS (Floating-point Conditional Move on Zero Single-precision)

Type: 32-Bit floating-point SP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	FSb	0111 FCMOVZS	00 CP0	0000 FS1

Syntax: FCMOVZS FSt, FSa, FSb

Purpose: Move the content of a single-precision floating-point register based on a zero condition stored in a single-precision floating-point register.

Description: If FSb is integer-zero (Note "integer-zero" means all 32 bits are 0), then move the single-precision floating-point value in FSa to FSt.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

This instruction is non-arithmetic and it does not generate any IEEE 754 exceptions.

Operations:

```
If (FSb == 0(31,0)) {
   FSt = FSa;
}
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When SP extension is not implemented, Register out-of-range)

Floating-point Exceptions: None

Privilege level: All

Note:

FCMPxxS (Floating Point Compare Single-precision)

FCMPxxS (no Invalid operation exception for QNaN)

FCMPxxS.e (with Invalid operation exception for QNaN)

xx = EQ, LT, LE, UN

Type: 32-Bit floating-point SP V1 extension

Format:

FCMPxxS

31	30 25	24 20	19 15	14 10	9 7	6	5 4	3 0
0	COP 110101	FSt	FSa	FSb	TYP	0 e	00 CP0	0100 FS2

FCMPxxS.e

31	30 25	24 20	19 15	14 10	9 7	6	5 4	3 0
0	COP 110101	FSt	FSa	FSb	TYP	1 e	00 CP0	0100 FS2

TYP	Mnemonic
000	EQ
001	LT
010	LE
011	UN
100-111	Reserved

Syntax: FCMPxxS FSt, FSa, FSb
FCMPxxS.e FSt, FSa, FSb

Purpose: Compare two single-precision floating-point numbers for various relationships.

Description: The single-precision floating-point value in FSa is compared with the single-precision floating-point value in FSb. If the specified relationship (EQ, LT, LE, and UN) is true, a value of integer 1 is written to FSt, otherwise a value of integer 0 is written to FSt.

EQ represents "equal". LT represents "less than". LE represents "less than or equal". UN represents "un-ordered" relationship. The unordered relationship is true if one or both operands are NaN. Every NaN shall compare un-ordered with everything, including itself.

Comparisons are exact and never overflow nor underflow. Comparisons ignore the sign of zero, so +0 = -0. Comparisons with plus and minus infinity (±∞) execute normally and do not take an Invalid Operation exception.

If one of the operands is a SNaN or when the ".e" flavor of compare instruction is used and one of the operand is a QNaN, an Invalid Operation condition is happened and the Invalid Operation flag in the FPCSR will be set to record this. If the Invalid Operation enable bit in the FPCSR is set, an Invalid Operation exception will be taken and no result will be written to Rt. If the enable bit is not set, then a true value (i.e. 1) will be written for the UN relationship or a false value (i.e. 0) will be written for the EQ/LT/LE relationships.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

The following tables show the results obtained when comparing various types of numbers based on each relationship.

Table 14. FCMPEQS results

EQ	**FSa**									
FSb		**-∞**	**-N**	**-DN**	**-0**	**+0**	**+DN**	**+N**	**+∞**	**NaN**
	-∞	1	0	0	0	0	0	0	0	0*
	-N	0	Calc	0	0	0	0	0	0	0*
	-DN	0	0	*	0	0	0	0	0	0*

EQ	FSa									
	-0	0	0	0	1	1	0	0	0	0*
	+0	0	0	0	1	1	0	0	0	0*
	+DN	0	0	0	0	0	*	0	0	0*
	+N	0	0	0	0	0	0	Calc	0	0*
	+∞	0	0	0	0	0	0	0	1	0*
	NaN	0*	0*	0*	0*	0*	0*	0*	0*	0*

Table 15. FCMPLTS results

LT	FSa									
		-∞	-N	-DN	-0	+0	+DN	+N	+∞	NaN
	-∞	0	0	0	0	0	0	0	0	0*
	-N	1	Calc	0	0	0	0	0	0	0*
	-DN	1	1	*	0	0	0	0	0	0*
	-0	1	1	1	0	0	0	0	0	0*
FSb	+0	1	1	1	0	0	0	0	0	0*
	+DN	1	1	1	1	1	*	0	0	0*
	+N	1	1	1	1	1	1	Calc	0	0*
	+∞	1	1	1	1	1	1	1	0	0*
	NaN	0*	0*	0*	0*	0*	0*	0*	0*	0*

Table 16. FCMPLES results

LE	FSa									
		-∞	-N	-DN	-0	+0	+DN	+N	+∞	NaN
	-∞	1	0	0	0	0	0	0	0	0*
FSb	-N	1	Calc	0	0	0	0	0	0	0*
	-DN	1	1	*	0	0	0	0	0	0*

LE		FSa								
	-0	1	1	1	1	1	0	0	0	0*
	+0	1	1	1	1	1	0	0	0	0*
	+DN	1	1	1	1	1	*	0	0	0*
	+N	1	1	1	1	1	1	Calc	0	0*
	+∞	1	1	1	1	1	1	1	1	0*
	NaN	0*	0*	0*	0*	0*	0*	0*	0*	0*

Table 17. FCMPUNS results

UN		FSa								
		-∞	-N	-DN	-0	+0	+DN	+N	+∞	NaN
FSb	-∞	0	0	0	0	0	0	0	0	1*
	-N	0	0	0	0	0	0	0	0	1*
	-DN	0	0	0	0	0	0	0	0	1*
	-0	0	0	0	0	0	0	0	0	1*
	+0	0	0	0	0	0	0	0	0	1*
	+DN	0	0	0	0	0	0	0	0	1*
	+N	0	0	0	0	0	0	0	0	1*
	+∞	0	0	0	0	0	0	0	0	1*
	NaN	1*	1*	1*	1*	1*	1*	1*	1*	1*

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
Calc	Means either 0 or 1 based on comparison calculation.
*	Indicates 0 or 1 (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)
0*/1*	Indicates 0 or 1 when an Invalid Operation exception is not happened or when an Invalid Operation exception is happened but the exception enable bit is not set.

Operations:

```
if ((FSa == NaN) || (FSb == NaN)) {
   if ((FSa == SNaN) || (FSb == SNaN) ||
        ((".e" form) &&
          ((FSa == QNaN) || (FSb == QNaN)))) {
      if (FPCSR.IVOE == 1) {
         Generate_Exception(FP_IVO);
      } else {
         FPCSR.IO = 1;
      }
   }
}
if (FSa relation FSb) { // pass IVO exception checking
   FSt = 1;
} else {
   FSt = 0;
}
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When SP extension is not implemented, Register out-of-range)
Denorm input (for non-denorm-support FPU)

Floating Point Exceptions: Invalid operation

Privilege level: All

Note:

- "Compare Less Than A,B" is the same as "Compare Greater Than B,A"; and "Compare Less Than or Equal A,B" is the same as "Compare Greater Than or Equal B,A". Therefore, only the less-than operations are provided.
- Andes FPU extension provides hardware support for the IEEE Standard required six predicates (=,≠,<,≦,≧,>) and the optional un-ordered predicate. The other 19 optional predicates can be constructed from sequences of two comparisons and two branches.

FCPYNSS (Floating-point Copy Negative Sign Single-precision)

Type: 32-Bit floating-point SP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	FSb	0010 FCPYNSS	00 CP0	0000 FS1

Syntax: FCPYNSS FSt, FSa, FSb

Purpose: Generate a floating-point value by negating and copying the sign of a floating-point value in one register to a value in another register.

Description: The sign of the floating-point value in FSb is negated and then copied to the floating-point value in FSa. The floating-point result is written to FSt.

No checking of NaN operands is performed for this instruction.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
FSt = CONCAT(NOT(FSb(31)),FSa(30,0));
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When SP extension is not implemented, Register out-of-range)

Floating Point Exceptions: None

Privilege level: All

Note:

- –*x* is performed with “FCPYNSS FSt, FSa, FSa”.

FCPYSS (Floating-point Copy Sign Single-precision)

Type: 32-Bit floating-point SP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	FSb	0011 FCPYSS	00 CP0	0000 FS1

Syntax: FCPYSS FSt, FSa, FSb

Purpose: Generate a floating-point value by copying the sign of a floating-point value in one register to a value in another register.

Description: The sign of the floating-point value in FSb is copied to the floating-point value in FSa. The floating-point result is written to FSt.

No checking of NaN operands is performed for this instruction.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
    FSt = CONCAT(FSb(31),FSa(30,0));
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When SP extension is not implemented, Register out-of-range)

Floating Point Exceptions: None

Privilege level: All

Note:

- Move between floating-point registers can be performed with “FCPYSS FSt, FSa, FSa”.

FDIVS (Floating-point Divide Single-precision)

Type: 32-Bit floating-point SP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	FSb	1101 FDIVS	00 CP0	0000 FS1

Syntax: FDIVS FSt, FSa, FSb

Purpose: Divide the single-precision floating-point values in two registers.

Description: The single-precision floating-point value in FSa is divided by the single-precision floating-point value in FSb. And the floating-point result is written to FSt.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

A "Divide by Zero" exception occurs if the divisor (FSb) is zero and the dividend (FSa) is a finite nonzero number. When no trap happens, the result should be a correctly signed ∞.

The following table shows the results obtained when dividing various types of numbers.

Table 18. FDIVS results

		FSa								
		-∞	**-N**	**-DN**	**-0**	**+0**	**+DN**	**+N**	**+∞**	**NaN**
FSb	**-∞**	IVO	+0	+0	+0	-0	-0	-0	IVO	NaNa
	-N	+∞	+F	*	+0	-0	*	-F	-∞	NaNa
	-DN	+∞	*	*	+0	-0	*	*	-∞	NaNa
	-0	+∞	DBZ	DBZ	IVO	IVO	DBZ	DBZ	-∞	NaNa

		FSa								
	+0	-∞	DBZ	DBZ	IVO	IVO	DBZ	DBZ	+∞	NaNa
	+DN	-∞	*	*	-0	+0	*	*	+∞	NaNa
	+N	-∞	-F	*	-0	+0	*	+F	+∞	NaNa
	+∞	IVO	-0	-0	-0	+0	+0	+0	IVO	NaNa
	NaN	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNc

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
F	Means a finite floating-point value (N, DN) which may involve underflow, overflow handling to become (N, DN, ∞).
IVO	Means invalid operation exception. If trapping is disabled, deliver the default QNaN as the result.
DBZ	Means divide-by-zero exception
NaNa	Means a QNaN from FSa or a QNaN converted from a SNaN from FSa by changing its most significant fraction bit from 0 to 1.
NaNb	Means a QNaN from FSb or a QNaN converted from a SNaN from FSb by changing its most significant fraction bit from 0 to 1.
NaNc	Means a QNaN converted from ● A SNaN from FSa or from FSb if FSa is a QNaN ● A QNaN from FSa if FSb is also a QNaN
*	Indicates F (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)

Operations:

```
FSt = FSa / FSb;
```

Exceptions: Reserved instruction (When FPU is not implemented)

FPU disabled (When the use of FPU is not enabled)

Reserved instruction (When SP extension is not implemented, Register out-of-range)

Denorm input (for non-denorm-support FPU)

Floating-point Exceptions: Invalid operation, Inexact, Overflow, Underflow, Divide by Zero

Privilege level: All

Note:

FLS (Floating-point Load Single-precision Data)

This instruction is common to both SP and DP extensions. Please see FLS instruction description in page 32 for details.

FLSI (Floating-point Load Single-precision Data Immediate)

This instruction is common to both SP and DP extensions. Please see FLSI instruction description in page 34 for details.

FMADDS (Floating-point Multiply and Add Single-precision)

Type: 32-Bit floating-point SP V1 extension (FMA optional)

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	FSb	0100 FMADDS	00 CP0	0000 FS1

Syntax: FMADDS FSt, FSa, FSb

Purpose: Multiply the single-precision floating-point values of two registers and then accumulate the result to the third register.

Description: The single-precision floating-point value in FSa is multiplied with the single-precision floating-point value in FSb. And the multiplication result with unbounded range and precision is added with the floating-point value in FSt. The rounded addition result is then written back to FSt. This instruction is implemented if FPCFG.FMA register field is equal to 1.

No underflow, overflow, or inexact exception can occur due to the multiplication. These exceptions can be generated due to the addition. So a FMADDS instruction differs from a FMULS instruction followed by a FADDS instruction.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

If (FSt, FSa, FSb) contains the following content as (c, 0, infinity) or (c, infinity, 0), an Invalid operation exception will be raised even if c is a QNaN.

The following table shows the intermediate results obtained when multiplying various types of numbers from FSa and FSb.

Table 19. FMADDS multiplication intermediate results

		FSa								
		-∞	**-N**	**-DN**	**-0**	**+0**	**+DN**	**+N**	**+∞**	**NaN**
FSb	**-∞**	+∞	+∞	+∞	IVO	IVO	-∞	-∞	-∞	NaN1
	-N	+∞	+Fu	*’	+0	-0	*’	-Fu	-∞	NaN1
	-DN	+∞	*’	*’	+0	-0	*’	*’	-∞	NaN1
	-0	IVO	+0	+0	+0	-0	-0	-0	IVO	NaN1
	+0	IVO	-0	-0	-0	+0	+0	+0	IVO	NaN1
	+DN	-∞	*’	*’	-0	+0	*’	*’	+∞	NaN1
	+N	-∞	-Fu	*’	-0	+0	*’	+Fu	+∞	NaN1
	+∞	-∞	-∞	-∞	IVO	IVO	+∞	+∞	+∞	NaN1
	NaN	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN3

The following table shows the final results obtained when adding various types of numbers from FSt and the intermediate result from multiplication.

Table 20. FMADDS results

		Intermediate result from multiplication (MIR)								
		-∞	**-Fu**	**IVO**	**-0**	**+0**	**DIE**	**+Fu**	**+∞**	**NaN**
FSt	**-∞**	-∞	-∞	IVO1	-∞	-∞	-∞	-∞	IVO1	NaNm
	-N	-∞	-F	IVO1	FSt	FSt	DIE	±F	+∞	NaNm
	-DN	-∞	*	IVO1	*	*	DIE	*	+∞	NaNm
	-0	-∞	MIR’	IVO1	-0	±0	DIE	MIR’	+∞	NaNm
	+0	-∞	MIR’	IVO1	±0	+0	DIE	MIR’	+∞	NaNm
	+DN	-∞	*	IVO1	*	*	DIE	*	+∞	NaNm
	+N	-∞	±F	IVO1	FSt	FSt	DIE	+F	+∞	NaNm
	+∞	IVO1	+∞	IVO1	+∞	+∞	+∞	+∞	+∞	NaNm
	NaN	NaNt	NaNt	IVO2	NaNt	NaNt	NaNt	NaNt	NaNt	NaNn

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
Fu	Means a finite floating-point value (N, DN) with unbounded range and precision.
F	Means a finite floating-point value (N, DN) which may involve underflow, overflow handling to become (N, DN, ∞).
IVO	Means invalid operation exception
IVO1	Means invalid operation exception. If trapping is disabled, deliver the default QNaN as the result.
IVO2	Means invalid operation exception. If trapping is disabled, deliver NaNt (see below) as the result.
DIE	Means denorm input exception
MIR’	Means the properly rounded MIR
NaN1	Means the NaN from FSa (it can be either SNaN or QNaN).
NaN2	Means the NaN from FSb (it can be either SNaN or QNaN).
NaN3	Means a NaN from ● the SNaN from FSa or the SNaN from FSb if FSa is a QNaN ● the QNaN from FSa if FSb is also a QNaN
NaNt	Means a QNaN from FSt or a QNaN converted from a SNaN from FSt by changing its most significant fraction bit from 0 to 1.
NaNm	Means a QNaN from MIR or a QNaN converted from a SNaN from MIR by changing its most significant fraction bit from 0 to 1.
NaNn	Means a QNaN converted from ● the SNaN from FSt or the SNaN from MIR if FSt is a QNaN ● the QNaN from FSt if MIR is also a QNaN
*’	Indicates Fu (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)
*	Indicates F (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)

When (*a* x *b*) + *c* is exactly zero, the sign of the result shall be determined by the rules described by the FADDS description. When the exact result of (*a* x *b*) + *c* is non-zero yet the result is zero because of rounding, the zero results takes the sign of the exact result.

Operations:

```
    FSt = FSt + (FSa * FSb);
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When SP extension is not implemented, Register out-of-range)
Denorm input (for non-denorm-support FPU)

Floating-point Exceptions: Invalid operation, Inexact, Overflow, Underflow

Privilege level: All

Note:

FMULS (Floating-point Multiplication Single-precision)

Type: 32-Bit floating-point SP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	FSb	1100 FMULS	00 CP0	0000 FS1

Syntax: FMULS FSt, FSa, FSb

Purpose: Multiply the single-precision floating-point values in two registers.

Description: The single-precision floating-point value in FSa is multiplied with the single-precision floating-point value in FSb. And the floating-point result is written to FSt.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

The following table shows the results obtained when multiplying various types of numbers.

Table 21. FMULS results

		FSa								
		-∞	**-N**	**-DN**	**-0**	**+0**	**+DN**	**+N**	**+∞**	**NaN**
FSb	**-∞**	+∞	+∞	+∞	IVO	IVO	-∞	-∞	-∞	NaNa
	-N	+∞	+F	*	+0	-0	*	-F	-∞	NaNa
	-DN	+∞	*	*	+0	-0	*	*	-∞	NaNa
	-0	IVO	+0	+0	+0	-0	-0	-0	IVO	NaNa
	+0	IVO	-0	-0	-0	+0	+0	+0	IVO	NaNa
	+DN	-∞	*	*	-0	+0	*	*	+∞	NaNa
	+N	-∞	-F	*	-0	+0	*	+F	+∞	NaNa
	+∞	-∞	-∞	-∞	IVO	IVO	+∞	+∞	+∞	NaNa
	NaN	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNc

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
F	Means a finite floating-point value (N, DN) which may involve underflow, overflow handling to become (N, DN, ∞).
IVO	Means invalid operation exception. If trapping is disabled, deliver the default QNaN as the result.
NaNa	Means a QNaN from FSa or a QNaN converted from a SNaN from FSa by changing its most significant fraction bit from 0 to 1.
NaNb	Means a QNaN from FSb or a QNaN converted from a SNaN from FSb by changing its most significant fraction bit from 0 to 1.
NaNc	Means a QNaN converted from ● A SNaN from FSa or from FSb if FSa is a QNaN ● A QNaN from FSa if FSb is also a QNaN
*	Indicates F (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)

Operations:

```
    FSt = FSa * FSb;
```

Exceptions: Reserved instruction (When FPU is not implemented)

FPU disabled (When the use of FPU is not enabled)

Reserved instruction (When SP extension is not implemented, Register out-of-range)

Denorm input (for non-denorm-support FPU)

Floating-point Exceptions: Invalid operation, Inexact, Overflow, Underflow

Privilege level: All

Note:

FMFSR (Floating-point Move From Single-precision FR)

This instruction is common to both SP and DP extensions. Please see FLS instruction description in page 40 for details.

FMSUBS (Floating-point Multiply and Subtraction Single-precision)

Type: 32-Bit floating-point SP V1 extension (FMA optional)

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	FSb	0101 FMSUBS	00 CP0	0000 FS1

Syntax: FMSUBS FSt, FSa, FSb

Purpose: Multiply the single-precision floating-point values of two registers and then subtract the result from the third register.

Description: The single-precision floating-point value in FSa is multiplied with the single-precision floating-point value in FSb. And the multiplication result with unbounded range and precision is subtracted from the floating-point value in FSt. The rounded subtraction result is then written back to FSt. This instruction is implemented if FPCFG.FMA register field is equal to 1.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.
If (FSt, FSa, FSb) contains the following content as (c, 0, infinity) or (c, infinity, 0), an Invalid operation exception will be raised even if c is a QNaN.

The following table shows the intermediate results obtained when multiplying various types of numbers from FSa and FSb.

Table 22. FMSUBS multiplication intermediate results

		FSb								
		-∞	**-N**	**-DN**	**-0**	**+0**	**+DN**	**+N**	**+∞**	**NaN**
	-∞	+∞	+∞	+∞	IVO	IVO	-∞	-∞	-∞	NaN1
	-N	+∞	+Fu	*'	+0	-0	*'	-Fu	-∞	NaN1
	-DN	+∞	*'	*'	+0	-0	*'	*'	-∞	NaN1
FSa	**-0**	IVO	+0	+0	+0	-0	-0	-0	IVO	NaN1
	+0	IVO	-0	-0	-0	+0	+0	+0	IVO	NaN1
	+DN	-∞	*'	*'	-0	+0	*'	*'	+∞	NaN1
	+N	-∞	-Fu	*'	-0	+0	*'	+Fu	+∞	NaN1
	+∞	-∞	-∞	-∞	IVO	IVO	+∞	+∞	+∞	NaN1
	NaN	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN3

The following table shows the final results obtained when subtracting various types of numbers from FSt and the intermediate result from multiplication.

Table 23. FMSUBS results

		Intermediate result from multiplication (MIR)								
		-∞	**-Fu**	**IVO**	**-0**	**+0**	**DIE**	**+Fu**	**+∞**	**NaN**
	-∞	IVO1	-∞	IVO1	-∞	-∞	-∞	-∞	-∞	NaNm
	-N	+∞	±F	IVO1	FSt	FSt	DIE	-F	-∞	NaNm
	-DN	+∞	*	IVO1	*	*	DIE	*	-∞	NaNm
FSt	**-0**	+∞	MIR'	IVO1	±0	-0	DIE	MIR'	-∞	NaNm
	+0	+∞	MIR'	IVO1	+0	±0	DIE	MIR'	-∞	NaNm
	+DN	+∞	*	IVO1	*	*	DIE	*	-∞	NaNm
	+N	+∞	+F	IVO1	FSt	FSt	DIE	±F	-∞	NaNm
	+∞	+∞	+∞	IVO1	+∞	+∞	+∞	+∞	IVO1	NaNm
	NaN	NaNt	NaNt	IVO2	NaNt	NaNt	NaNt	NaNt	NaNt	NaNn

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
Fu	Means a finite floating-point value (N, DN) with unbounded range and precision.
F	Means a finite floating-point value (N, DN) which may involve underflow, overflow handling to become (N, DN, ∞).
IVO	Means invalid operation exception
IVO1	Means invalid operation exception. If trapping is disabled, deliver the default QNaN as the result.
IVO2	Means invalid operation exception. If trapping is disabled, deliver NaNt (see below) as the result.
DIE	Means denorm input exception
MIR’	Means the properly rounded MIR
NaN1	Means the NaN from FSa (it can be either SNaN or QNaN).
NaN2	Means the NaN from FSb (it can be either SNaN or QNaN).
NaN3	Means a NaN from ● the SNaN from FSa or the SNaN from FSb if FSa is a QNaN ● the QNaN from FSa if FSb is also a QNaN
NaNt	Means a QNaN from FSt or a QNaN converted from a SNaN from FSt by changing its most significant fraction bit from 0 to 1.
NaNm	Means a QNaN from MIR or a QNaN converted from a SNaN from MIR by changing its most significant fraction bit from 0 to 1.
NaNn	Means a QNaN converted from ● the SNaN from FSt or the SNaN from MIR if FSt is a QNaN ● the QNaN from FSt if MIR is also a QNaN
*’	Indicates Fu (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)
*	Indicates F (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)

When c - (a x b) is exactly zero, the sign of the result shall be determined by the rules described by the FSUBS description. When the exact result of c - (a x b) is non-zero yet the result is zero because of rounding, the zero results takes the sign of the exact result.

Operations:

```
    FSt = FSt - (FSa * FSb);
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When SP extension is not implemented, Register out-of-range)
Denorm input (for non-denorm-support FPU)

Floating-point Exceptions: Invalid operation, Inexact, Overflow, Underflow

Privilege level: All

Note:

FMTSR (Floating-point Move To Single-precision FR)

This instruction is common to both SP and DP extensions. Please see FLS instruction description in page 44 for details.

FNMADDS (Floating-point Negate Multiply and Add Single-precision)

Type: 32-Bit floating-point SP V1 extension (FMA optional)

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	FSb	1000 FNMADDS	00 CP0	0000 FS1

Syntax: FNMADDS FSt, FSa, FSb

Purpose: Multiply the single-precision floating-point values of two registers and then accumulate and negate the result to the third register.

Description: The single-precision floating-point value in FSa is multiplied with the single-precision floating-point value in FSb. And the multiplication result with unbounded range and precision is added with the floating-point value in FSt. The rounded addition result is negated and then written back to FSt. This instruction is implemented if FPCFG.FMA register field is equal to 1.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

If (FSt, FSa, FSb) contains the following content as (c, 0, infinity) or (c, infinity, 0), an Invalid operation exception will be raised even if c is a QNaN.

The following table shows the intermediate results obtained when multiplying various types of numbers from FSa and FSb.

Table 24 FNMADDS. multiplication intermediate results

		FSa								
		-∞	**-N**	**-DN**	**-0**	**+0**	**+DN**	**+N**	**+∞**	**NaN**
	-∞	+∞	+∞	+∞	IVO	IVO	-∞	-∞	-∞	NaN1
	-N	+∞	+Fu	*’	+0	-0	*’	-Fu	-∞	NaN1
	-DN	+∞	*’	*’	+0	-0	*’	*’	-∞	NaN1
FSb	**-0**	IVO	+0	+0	+0	-0	-0	-0	IVO	NaN1
	+0	IVO	-0	-0	-0	+0	+0	+0	IVO	NaN1
	+DN	-∞	*’	*’	-0	+0	*’	*’	+∞	NaN1
	+N	-∞	-Fu	*’	-0	+0	*’	+Fu	+∞	NaN1
	+∞	-∞	-∞	-∞	IVO	IVO	+∞	+∞	+∞	NaN1
	NaN	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN3

The following table shows the final results obtained when adding and negating various types of numbers from FSt and the intermediate result from multiplication.

Table 25. FNMADDS results

		Intermediate result from multiplication (MIR)								
		-∞	**-Fu**	**IVO**	**-0**	**+0**	**DIE**	**+Fu**	**+∞**	**NaN**
	-∞	+∞	+∞	IVO1	+∞	+∞	+∞	+∞	IVO1	NaNm
	-N	+∞	+F	IVO1	-FSt	-FSt	DIE	∓F	-∞	NaNm
	-DN	+∞	*	IVO1	*	*	DIE	*	-∞	NaNm
FSt	**-0**	+∞	-MIR’	IVO1	+0	∓0	DIE	-MIR’	-∞	NaNm
	+0	+∞	-MIR’	IVO1	∓0	-0	DIE	-MIR’	-∞	NaNm
	+DN	+∞	*	IVO1	*	*	DIE	*	-∞	NaNm
	+N	+∞	∓F	IVO1	-FSt	-FSt	DIE	-F	-∞	NaNm
	+∞	IVO1	-∞	IVO1	-∞	-∞	-∞	-∞	-∞	NaNm
	NaN	NaNt	NaNt	IVO2	NaNt	NaNt	NaNt	NaNt	NaNt	NaNn

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
Fu	Means a finite floating-point value (N, DN) with unbounded range and precision.
F	Means a finite floating-point value (N, DN) which may involve underflow, overflow handling to become (N, DN, ∞).
IVO	Means invalid operation exception
IVO1	Means invalid operation exception. If trapping is disabled, deliver the default QNaN as the result.
IVO2	Means invalid operation exception. If trapping is disabled, deliver NaNt (see below) as the result.
DIE	Means denorm input exception
MIR’	Means the properly rounded MIR
NaN1	Means the NaN from FSa (it can be either SNaN or QNaN).
NaN2	Means the NaN from FSb (it can be either SNaN or QNaN).
NaN3	Means a NaN from ● the SNaN from FSa or the SNaN from FSb if FSa is a QNaN ● the QNaN from FSa if FSb is also a QNaN
NaNt	Means a QNaN from FSt or a QNaN converted from a SNaN from FSt by changing its most significant fraction bit from 0 to 1.
NaNm	Means a QNaN from MIR or a QNaN converted from a SNaN from MIR by changing its most significant fraction bit from 0 to 1.
NaNn	Means a QNaN converted from ● the SNaN from FSt or the SNaN from MIR if FSt is a QNaN ● the QNaN from FSt if MIR is also a QNaN
*’	Indicates Fu (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)
*	Indicates F (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)

When $(a \times b) + c$ is exactly zero, the sign of the result shall be determined by negating the rules described by the FADDS description. When the exact result of $(a \times b) + c$ is non-zero yet the result is zero because of rounding, the zero results takes the negating sign of the exact result.

Operations:

```
    FSt = - (FSt + (FSa * FSb));
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When SP extension is not implemented, Register out-of-range)
Denorm input (for non-denorm-support FPU)

Floating-point Exceptions: Invalid operation, Inexact, Overflow, Underflow

Privilege level: All

Note:

FNMSUBS (Floating-point Negate Multiply and Subtraction Single-precision)

Type: 32-Bit floating-point SP V1 extension (FMA optional)

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	FSb	1001 FNMSUBS	00 CP0	0000 FS1

Syntax: FNMSUBS FSt, FSa, FSb

Purpose: Multiply the single-precision floating-point values of two registers and then subtract the value of the third register from the multiplication result.

Description: The single-precision floating-point value in FSa is multiplied with the single-precision floating-point value in FSb. And the multiplication result with unbounded range and precision is being subtracted by the floating-point value in FSt. The rounded subtraction result is then written back to FSt. It is equivalent to a negation of the result generated by a FMSUBS instruction. This instruction is implemented if FPCFG.FMA register field is equal to 1.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

If (FSt, FSa, FSb) contains the following content as (c, 0, infinity) or (c, infinity, 0), an Invalid operation exception will be raised even if c is a QNaN.

The following table shows the intermediate results obtained when multiplying various types of numbers from FSa and FSb.

Table 26. FNMSUBS multiplication intermediate results

		FSb								
		-∞	**-N**	**-DN**	**-0**	**+0**	**+DN**	**+N**	**+∞**	**NaN**
	-∞	+∞	+∞	+∞	IVO	IVO	-∞	-∞	-∞	NaN1
	-N	+∞	+Fu	*’	+0	-0	*’	-Fu	-∞	NaN1
	-DN	+∞	*’	*’	+0	-0	*’	*’	-∞	NaN1
	-0	IVO	+0	+0	+0	-0	-0	-0	IVO	NaN1
FSa	**+0**	IVO	-0	-0	-0	+0	+0	+0	IVO	NaN1
	+DN	-∞	*’	*’	-0	+0	*’	*’	+∞	NaN1
	+N	-∞	-Fu	*’	-0	+0	*’	+Fu	+∞	NaN1
	+∞	-∞	-∞	-∞	IVO	IVO	+∞	+∞	+∞	NaN1
	NaN	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN3

The following table shows the final results obtained when subtracting various types of numbers from FSt and the intermediate result from multiplication.

Table 27. FNMSUBS results

		Intermediate result from multiplication (MIR)								
		-∞	**-Fu**	**IVO**	**-0**	**+0**	**DIE**	**+Fu**	**+∞**	**NaN**
	-∞	IVO1	+∞	IVO1	+∞	+∞	+∞	+∞	+∞	NaNm
	-N	-∞	∓F	IVO1	-FSt	-FSt	DIE	+F	+∞	NaNm
	-DN	-∞	*	IVO1	*	*	DIE	*	+∞	NaNm
	-0	-∞	MIR’	IVO1	∓0	+0	DIE	MIR’	+∞	NaNm
FSt	**+0**	-∞	MIR’	IVO1	-0	∓0	DIE	MIR’	+∞	NaNm
	+DN	-∞	*	IVO1	*	*	DIE	*	+∞	NaNm
	+N	-∞	-F	IVO1	-FSt	-FSt	DIE	∓F	+∞	NaNm
	+∞	-∞	-∞	IVO1	-∞	-∞	-∞	-∞	IVO1	NaNm
	NaN	NaNt	NaNt	IVO2	NaNt	NaNt	NaNt	NaNt	NaNt	NaNn

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
Fu	Means a finite floating-point value (N, DN) with unbounded range and precision.
F	Means a finite floating-point value (N, DN) which may involve underflow, overflow handling to become (N, DN, ∞).
IVO	Means invalid operation exception
IVO1	Means invalid operation exception. If trapping is disabled, deliver the default QNaN as the result.
IVO2	Means invalid operation exception. If trapping is disabled, deliver NaNt (see below) as the result.
DIE	Means denorm input exception
MIR'	Means the properly rounded MIR
NaN1	Means the NaN from FSa (it can be either SNaN or QNaN).
NaN2	Means the NaN from FSb (it can be either SNaN or QNaN).
NaN3	Means a NaN from ● the SNaN from FSa or the SNaN from FSb if FSa is a QNaN ● the QNaN from FSa if FSb is also a QNaN
NaNt	Means a QNaN from FSt or a QNaN converted from a SNaN from FSt by changing its most significant fraction bit from 0 to 1.
NaNm	Means a QNaN from MIR or a QNaN converted from a SNaN from MIR by changing its most significant fraction bit from 0 to 1.
NaNn	Means a QNaN converted from ● the SNaN from FSt or the SNaN from MIR if FSt is a QNaN ● the QNaN from FSt if MIR is also a QNaN
*'	Indicates Fu (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)
*	Indicates F (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)

When *c* - (*a x b*) is exactly zero, the sign of the result shall be determined by negating the rules described by the FSUBS description. When the exact result of *c* - (*a x b*) is non-zero yet the result is zero because of rounding, the zero results takes the negating sign of the exact result.

Operations:

```
    FSt = - (FSt - (FSa * FSb));
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When SP extension is not implemented, Register out-of-range)
Denorm input (for non-denorm-support FPU)

Floating-point Exceptions: Invalid operation, Inexact, Overflow, Underflow

Privilege level: All

Note:

FS2SI (Floating Point Convert To Signed Integer from Single)

Type: 32-Bit floating-point SP V1 extension

Format:

FS2SI

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	11000 FS2SI	1111 F2OP	00 CP0	0000 FS1

FS2SI.z

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	11100 FS2SI.z	1111 F2OP	00 CP0	0000 FS1

Syntax:

FS2SI	FSt, FSa
FS2SI.z	FSt, Fsa

Purpose: Convert a single-precision floating-point value to a 32-bit signed integer.

Description: The single-precision floating-point value in FSa is converted to a signed integer value, rounded based on the rounding mode in the FPCSR register for the base form and rounded towards zero for the ".z" form. The result is written to FSt. If FSa contains Infinity, NaN, or the rounded result is outside the range of [2^{31}-1, -2^{31}], an Invalid Operation exception (IVO) is raised. If the IVO enable bit in the FPCSR is set, an FP Invalid Operation exception is taken (trapped). If the IVO enable bit is not set, then the IVO status flag in the FPCSR will be set and the following result will be written into FSt:

- +∞, or result > 2^{31}-1 : Rt = 0x7FFFFFFF
- -∞, or result < -2^{31} : Rt = 0x80000000
- NaN : Rt = 0xFFFFFFFF

This instruction will convert a denormalized number into a correctly-rounded integer.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
    FSt = ConvertSP2SI(FSa);
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When SP extension is not implemented, Register out-of-range)

Floating Point Exceptions: Invalid operation, Inexact

Privilege level: All

Note:

- The ".z" form is used for C, C++ languages.

FS2UI (Floating Point Convert To Unsigned Integer from Single)

Type: 32-Bit floating-point SP V1 extension

Format:

FS2UI

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	10000 FS2UI	1111 F2OP	00 CP0	0000 FS1

FS2UI.z

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	10100 FS2UI.z	1111 F2OP	00 CP0	0000 FS1

Syntax:	FS2UI	FSt, FSa
	FS2UI.z	FSt, Fsa

Purpose: Convert a single-precision floating-point value to a 32-bit unsigned integer.

Description: The single-precision floating-point value in FSa is converted to an unsigned integer value, rounded based on the rounding mode in the FPCSR register for the base form and rounded towards zero for the ".z" form. The result is written to FSt. If FSa contains Infinity, NaN, or the rounded result is outside the range of [2^{32}-1, 0] (Note: -0 is treated as 0), an Invalid Operation exception (IVO) is raised. If the IVO enable bit in the FPCSR is set, an FP Invalid Operation exception is taken (trapped). If the IVO enable bit is not set, then the IVO status flag in the FPCSR will be set and the following result will be written into FSt:

- +∞, or result > 2^{32}-1 : FSt = 0xFFFFFFFF
- -∞, result < 0 : FSt = 0x00000000
- NaN : FSt = 0xFFFFFFFF

This instruction will convert a denormalized number into a correctly-rounded integer.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
    FSt = ConvertSP2UI(FSa);
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When SP extension is not implemented, Register out-of-range)

Floating Point Exceptions: Invalid operation, Inexact

Privilege level: All

Note:

- The ".z" form is used for C, C++ languages.

FSI2S (Floating-point Convert From Signed Integer Single-precision)

Type: 32-Bit floating-point SP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	01100 FSI2S	1111 F2OP	00 CP0	0000 FS1

Syntax: FSI2S FSt, Fsa

Purpose: Convert a 32-bit signed integer to a single-precision floating-point value.

Description: The signed integer value in FSa is converted to a single-precision floating-point value. The floating-point result is rounded based on the rounding mode in FPCSR and the rounded result is written to FSt. An integer zero is converted to +0, not -0.

Operations:

```
FSt = ConvertSI2SP(SI(FSa));
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When SP extension is not implemented)

Floating Point Exceptions: Inexact

Privilege level: All

Note:

FSQRTS (Floating-point Square Root Single-precision)

Type: 32-Bit floating-point SP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	00001 FSQRTS	1111 F2OP	00 CP0	0000 FS1

Syntax: FSQRTS FSt, Fsa

Purpose: Compute square root from a single-precision floating-point value.

Description: The square root of the single-precision floating-point value in FSa is computed by this instruction. The floating-point result is rounded based on the rounding mode in FPCSR and the rounded result is written to FSt. If the floating-point value in FSa is -0, the result is -0. If the floating-point value in FSa is less than 0, an Invalid Operation exception is generated.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

The following table shows the results obtained when taking the square root of various types of numbers.

Table 28. FSQRTS results

FSa	FSt
-∞	IVO
-N	IVO
-DN	IVO
-0	-0
+0	+0

FSa	**FSt**
+DN	*
+N	+F
+∞	+∞
NaN	NaN'

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
F	Means a finite floating-point value (N, DN) which may involve underflow, overflow handling to become (N, DN, ∞).
IVO	Means invalid operation exception. If trapping is disabled, deliver the default QNaN as the result.
NaN'	Means a QNaN from FSa or a QNaN converted from a SNaN from FSa by changing its most significant fraction bit from 0 to 1.
*	Indicates F (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)

Note that IEEE-754 standard states that

"*Except that SquareRoot(-0) shall be −0, every valid square root shall have a positive sign.*"

Operations:

```
    FSt = SQRT(FSa);
```

Exceptions: Reserved instruction (When FPU is not implemented)

FPU disabled (When the use of FPU is not enabled)

Reserved instruction (When SP extension is not implemented, Register out-of-range)

Denorm input (for non-denorm-support FPU)

Floating Point Exceptions: Invalid operation, Inexact

Privilege level: All

Note:

FSS (Floating-point Store Single-precision Data)

This instruction is common to both SP and DP extensions. Please see FLS instruction description in page 51 for details.

FSSI (Floating-point Store Single-precision Data Immediate)

This instruction is common to both SP and DP extensions. Please see FLS instruction description in page 51 for details.

FSUBS (Floating-point Subtraction Single-precision)

Type: 32-Bit Floating point SP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	FSb	0001 FSUBS	00 CP0	0000 FS1

Syntax: FSUBS FSt, FSa, FSb

Purpose: Subtract the single-precision floating-point values in two registers.

Description: The single-precision floating-point value in FSb is subtracted from the single-precision floating-point value in FSa. And the floating-point result is written to FSt.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

The following table shows the results obtained when subtracting various types of numbers.

Table 29. FSUBS results

		FSa								
		-∞	-N	-DN	-0	+0	+DN	+N	+∞	NaN
FSb	-∞	IVO	+∞	+∞	+∞	+∞	+∞	+∞	+∞	NaNa
	-N	-∞	±F	*	-FSb	-FSb	*	+F	+∞	NaNa
	-DN	-∞	*	*	*	*	*	*	+∞	NaNa
	-0	-∞	FSa	*	±0	+0	*	FSa	+∞	NaNa
	+0	-∞	FSa	*	-0	±0	*	FSa	+∞	NaNa
	+DN	-∞	*	*	*	*	*	*	+∞	NaNa
	+N	-∞	-F	*	-FSb	-FSb	*	±F	+∞	NaNa

	FSa									
	+∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞	IVO	NaNa
	NaN	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNc

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
F	Means a finite floating-point value (N, DN) which may involve underflow, overflow handling to become (N, DN, ∞).
IVO	Means invalid operation exception. If trapping is disabled, deliver the default QNaN as the result.
NaNa	Means a QNaN from FSa or a QNaN converted from a SNaN from FSa by changing its most significant fraction bit from 0 to 1.
NaNb	Means a QNaN from FSb or a QNaN converted from a SNaN from FSb by changing its most significant fraction bit from 0 to 1.
NaNc	Means a QNaN converted from ● A SNaN from FSa or from FSb if FSa is a QNaN ● A QNaN from FSa if FSb is also a QNaN
*	Indicates F (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)

Note that IEEE-754 standard states that

"*When the difference of two operands with like signs is exactly zero, the sign of that difference shall be + in all rounding modes except round toward -∞, in which mode that sign shall be –. However, x – (– x) retains the same sign as x even when x is zero.*"

Operations:

```
FSt = FSa - FSb;
```

Exceptions: Reserved instruction (When FPU is not implemented)

FPU disabled (When the use of FPU is not enabled)

Reserved instruction (When SP extension is not implemented, Register out-of-range)

Denorm input (for non-denorm-support FPU)

Floating-point Exceptions: Invalid operation, Inexact, Overflow, Underflow

Privilege level: All

Note:

FUI2S (Floating-point Convert From Unsigned Integer Single-precision)

Type: 32-Bit floating-point SP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FSa	01000 FUI2S	1111 F2OP	00 CP0	0000 FS1

Syntax: FUI2S FSt, Fsa

Purpose: Convert a 32-bit unsigned integer to a single-precision floating-point value.

Description: The unsigned integer value in FSa is converted to a single-precision floating-point value. The floating-point result is rounded based on the rounding mode in FPCSR and the rounded result is written to FSt. An integer zero is converted to +0, not -0.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
    FSt = ConvertSI2SP(UI(FSa));
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When SP extension is not implemented, Register out-of-range)

Floating Point Exceptions: Inexact

Privilege level: All

Note:

3.4. Double Precision Extension Instructions

These instructions will be present if DP extension is implemented.

Instruction	Description
FADDD	DP floating-point addition
FSUBD	DP floating-point subtraction
FMULD	DP floating-point multiplication
FDIVD	DP floating-point division
FABSD	DP floating-point absolute value
FSQRTD	DP floating-point square root
FCPYSD	Move to SP/DP common section
FCPYNSD	DP floating-point copy negative sign
FCMPxxD	DP floating-point compare (no IVO exception on QNAN)
FCMPxxD.e	DP floating-point compare (IVO exception on QNAN)
FCMOVZD	DP floating-point conditional move on zero
FCMOVND	DP floating-point conditional move on not zero
FLD(I)(.bi)	Load DP floating-point register (immediate) (with base update)
FSD(I)(.bi)	Store DP floating-point register (immediate) (with base update)
FMFDR	Move from DP floating-point register to two general purpose registers
FMTDR	Move to DP floating-point register from two general purpose registers
FUI2D	Convert unsigned integer to DP floating-point
FSI2D	Convert signed integer to DP floating-point
FD2UI	Convert DP floating-point to unsigned integer (FPCSR rounding mode)

Instruction	Description
FD2UI.z	Convert DP floating-point to unsigned integer (toward zero rounding mode)
FD2SI	Convert DP floating-point to signed integer (FPCSR rounding mode)
FD2SI.z	Convert DP floating-point to signed integer (toward zero rounding mode)

These instructions will be present if DP extension is implemented and FMA option is supported.

Instruction	Description
FMADDD	DP floating-point addition multiply and add
FMSUBD	DP floating-point subtraction multiply and subtract
FNMADDD	DP floating-point negate of "multiply and add"
FNMSUBD	DP floating-point negate of "multiply and subtract"

FABSD (Floating-point Absolute Double-precision)

Type: 32-Bit floating-point DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FDt	FDa	00101 FABSD	1111 F2OP	00 CP0	1000 FD1

Syntax: FABSD FDt, Fda

Purpose: Compute the absolute value of a double-precision floating-point data.

Description: The absolute value of the double-precision floating-point data in FDa is calculated and written to FDt.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
if (FDa) <= -0) {
   FDt = -FDa;
} else {
   FDt = FDa;
}
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When DP extension is not implemented, Register out-of-range)

Floating-point Exceptions: None

Privilege level: All

Note:

FADDD (Floating-point Addition Double-precision)

Type: 32-Bit floating-point DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FDt	FDa	FDb	0000 FADDD	00 CP0	1000 FD1

Syntax: FADDD FDt, FDa, FDb

Purpose: Add the double-precision floating-point values in two registers.

Description: The double-precision floating-point value in FDa is added with the double-precision floating-point value in FDb. And the floating-point result is written to FDt.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

The following table shows the results obtained when adding various types of numbers.

Table 30. FADDD results

	FDa									
		-∞	**-N**	**-DN**	**-0**	**+0**	**+DN**	**+N**	**+∞**	**NaN**
FDb	**-∞**	-∞	-∞	-∞	-∞	-∞	-∞	-∞	IVO	NaNa
	-N	-∞	-F	*	FDb	FDb	*	±F	+∞	NaNa
	-DN	-∞	*	*	*	*	*	*	+∞	NaNa
	-0	-∞	FDa	*	-0	±0	*	FDa	+∞	NaNa
	+0	-∞	FDa	*	±0	+0	*	FDa	+∞	NaNa
	+DN	-∞	*	*	*	*	*	*	+∞	NaNa

	FDa									
	+N	-∞	±F	*	FDb	FDb	*	+F	+∞	NaNa
	+∞	IVO	+∞	+∞	+∞	+∞	+∞	+∞	+∞	NaNa
	NaN	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNc

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
F	Means a finite floating-point value (N, DN) which may involve underflow, overflow handling to become (N, DN, ∞).
IVO	Means invalid operation exception. If trapping is disabled, deliver the default QNaN as the result.
NaNa	Means a QNaN from FDa or a QNaN converted from a SNaN from FDa by changing its most significant fraction bit from 0 to 1.
NaNb	Means a QNaN from FDb or a QNaN converted from a SNaN from FDb by changing its most significant fraction bit from 0 to 1.
NaNc	Means a QNaN converted from ● A SNaN from FDa or from FDb if FDa is a QNaN ● A QNaN from FDa if FDb is also a QNaN
*	Indicates F (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)

Note that IEEE-754 standard states that

"*When the sum of two operands with opposite signs is exactly zero, the sign of that sum shall be + in all rounding modes except round toward -∞, in which mode that sign shall be –. However, x + x retains the same sign as x even when x is zero.*"

Operations:

```
        FDt = FDa + FDb;
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When DP extension is not implemented, Register out-of-range)
Denorm input (for non-denorm-support FPU)

Floating Point Exceptions: Invalid operation, Inexact, Overflow, Underflow

Privilege level: All

Note:

FCMOVND (Floating-point Conditional Move on Not Zero Double-precision)

Type: 32-Bit floating-point DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FDt	FDa	FSb	0110 FCMOVND	00 CP0	1000 FD1

Syntax: FCMOVND FDt, FDa, FSb

Purpose: Move the content of a double-precision floating-point register based on a not zero condition stored in a single-precision floating-point register.

Description: If FSb is not integer-zero (Note "integer-zero" means all 32 bits are 0), then move the double-precision floating-point value in FDa register to FDt register.

This instruction is non-arithmetic and it does not generate any IEEE 754 exceptions.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
If (FSb != 0(31,0)) {
   FDt = FDa;
}
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When DP extension is not implemented, Register out-of-range)

Floating-point Exceptions: None

Privilege level: All

Note:

FCMOVZD (Floating-point Conditional Move on Zero Double-precision)

Type: 32-Bit floating-point DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FDt	FDa	FSb	0111 FCMOVZD	00 CP0	1000 FD1

Syntax: FCMOVZD FDt, FDa, FSb

Purpose: Move the content of a double-precision floating-point register based on a zero condition stored in a single-precision floating-point register.

Description: If FSb is integer-zero (Note "integer-zero" means all 32 bits are 0), then move the double-precision floating-point value in FDa register to FDt register.

This instruction is non-arithmetic and it does not generate any IEEE 754 exceptions.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
If (FSb == 0(31,0)) {
   FDt = FDa;
}
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When DP extension is not implemented, Register out-of-range)

Floating-point Exceptions: None

Privilege level: All

Note:

FCMPxxD (Floating Point Compare Double-precision)

FCMPxxD (no Invalid operation exception for QNaN)

FCMPxxD.e (with Invalid operation exception for QNaN)

xx = EQ, LT, LE, UN

Type: 32-Bit floating-point DP V1 extension

Format:

FCMPxxD

31	30 25	24 20	19 15	14 10	9 7	6	5 4	3 0
0	COP 110101	FSt	FDa	FDb	TYP	0 e	00 CP0	1100 FD2

FCMPxxD.e

31	30 25	24 20	19 15	14 10	9 7	6	5 4	3 0
0	COP 110101	FSt	FDa	FDb	TYP	1 e	00 CP0	1100 FD2

xx	**Mnemonic**
000	EQ
001	LT
010	LE
011	UN
100-111	Reserved

Syntax: FCMPxxD FSt, FDa, FDb
FCMPxxD.e FSt, FDa, FDb

Purpose: Compare two double-precision floating-point numbers for various relationships.

Description: The double-precision floating-point value in FDa is compared with the double-precision floating-point value in FDb. If the specified relationship (EQ, LT, LE, and UN) is true, a value of integer 1 is written to FSt, otherwise a value of integer 0 is written to FSt.

EQ represents “equal”. LT represents “less than”. LE represents “less than or equal”. UN represents “un-ordered” relationship. The unordered relationship is true if one or both operands are NaN. Every NaN shall compare un-ordered with everything, including itself.

Comparisons are exact and never overflow nor underflow. Comparisons ignore the sign of zero, so +0 = -0. Comparisons with plus and minus infinity (±∞) execute normally and do not take an Invalid Operation exception.

If one of the operands is a SNaN or when the “.e” flavor of compare instruction is used and one of the operand is a QNaN, an Invalid Operation condition is happened and the Invalid Operation flag in the FPCSR will be set to record this. If the Invalid Operation enable bit in the FPCSR is set, an Invalid Operation exception will be taken and no result will be written to Rt. If the enable bit is not set, then a true value (i.e. 1) will be written for the UN relationship or a false value (i.e. 0) will be written for the EQ/LT/LE relationships.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

The following tables show the results obtained when comparing various types of numbers based on each relationship.

Table 31. FCMPEQD results

EQ		**FDa**								
FDb		**-∞**	**-N**	**-DN**	**-0**	**+0**	**+DN**	**+N**	**+∞**	**NaN**
	-∞	1	0	0	0	0	0	0	0	0*
	-N	0	Calc	0	0	0	0	0	0	0*
	-DN	0	0	*	0	0	0	0	0	0*

EQ	FDa									
	-0	0	0	0	1	1	0	0	0	0*
	+0	0	0	0	1	1	0	0	0	0*
	+DN	0	0	0	0	0	*	0	0	0*
	+N	0	0	0	0	0	0	Calc	0	0*
	+∞	0	0	0	0	0	0	0	1	0*
	NaN	0*	0*	0*	0*	0*	0*	0*	0*	0*

Table 32. FCMPLTD results

LT	FDa									
FDb		-∞	-N	-DN	-0	+0	+DN	+N	+∞	NaN
	-∞	0	0	0	0	0	0	0	0	0*
	-N	1	Calc	0	0	0	0	0	0	0*
	-DN	1	1	*	0	0	0	0	0	0*
	-0	1	1	1	0	0	0	0	0	0*
	+0	1	1	1	0	0	0	0	0	0*
	+DN	1	1	1	1	1	*	0	0	0*
	+N	1	1	1	1	1	1	Calc	0	0*
	+∞	1	1	1	1	1	1	1	0	0*
	NaN	0*	0*	0*	0*	0*	0*	0*	0*	0*

Table 33. FCMPLED results

LE	FDa									
FDb		-∞	-N	-DN	-0	+0	+DN	+N	+∞	NaN
	-∞	1	0	0	0	0	0	0	0	0*
	-N	1	Calc	0	0	0	0	0	0	0*
	-DN	1	1	*	0	0	0	0	0	0*

LE	FDa									
	-0	1	1	1	1	1	0	0	0	0*
	+0	1	1	1	1	1	0	0	0	0*
	+DN	1	1	1	1	1	*	0	0	0*
	+N	1	1	1	1	1	1	Calc	0	0*
	+∞	1	1	1	1	1	1	1	1	0*
	NaN	0*	0*	0*	0*	0*	0*	0*	0*	0*

Table 34. FCMPUND results

UN	**FDa**									
		-∞	**-N**	**-DN**	**-0**	**+0**	**+DN**	**+N**	**+∞**	**NaN**
FDb	**-∞**	0	0	0	0	0	0	0	0	1*
	-N	0	0	0	0	0	0	0	0	1*
	-DN	0	0	0	0	0	0	0	0	1*
	-0	0	0	0	0	0	0	0	0	1*
	+0	0	0	0	0	0	0	0	0	1*
	+DN	0	0	0	0	0	0	0	0	1*
	+N	0	0	0	0	0	0	0	0	1*
	+∞	0	0	0	0	0	0	0	0	1*
	NaN	1*	1*	1*	1*	1*	1*	1*	1*	1*

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
Calc	Means either 0 or 1 based on comparison calculation.
*	Indicates 0 or 1 (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)
0*/1*	Indicates 0 or 1 when an Invalid Operation exception is not happened or when an Invalid Operation exception is happened but the exception enable bit is not set.

Operations:

```
if ((FDa == NaN) || (FDb == NaN)) {
   if ((FDa == SNaN) || (FDb == SNaN) ||
        ((".e" form) &&
         ((FDa == QNaN) || (FDb == QNaN)))) {
      if (FPCSR.IVOE == 1) {
         Generate_Exception(FP_IVO);
      } else {
         FPCSR.IO = 1;
      }
   }
}
if (FDa relation FDb) {  // pass IVO exception checking
   FSt = 1;
} else {
   FSt = 0;
}
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When DP extension is not implemented, Register out-of-range)
Denorm input (for non-denorm-support FPU)

Floating Point Exceptions: Invalid operation

Privilege level: All

Note:

1. "Compare Less Than A,B" is the same as "Compare Greater Than B,A"; and "Compare Less Than or Equal A,B" is the same as "Compare Greater Than or Equal B,A". Therefore, only the less-than operations are provided.
2. Andes FPU extension provides hardware support for the IEEE Standard required six predicates (=,≠,<,≦,≧,>) and the optional un-ordered predicate. The other 19 optional predicates can be constructed from sequences of two comparisons and two branches.

FCPYNSD (Floating-point Copy Negative Sign Double-precision)

Type: 32-Bit floating-point DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FDt	FDa	FDb	0010 FCPYNSD	00 CP0	1000 FD1

Syntax: FCPYNSD FDt, FDa, FDb

Purpose: Generate a floating-point value by negating and copying the sign of a floating-point value in one register to a value in another register.

Description: The sign of the floating-point value in FDb is negated and then copied to the floating-point value in FDa. The floating-point result is written to FDt.

No checking of NaN operands is performed for this instruction.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
    FDt = CONCAT(NOT(FDb(63)),FDa(62,0));
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When DP extension is not implemented, Register out-of-range)

Floating Point Exceptions: None

Privilege level: All

Note:

- $-x$ is performed with “FCPYNSD FDt, FDa, FDa”.

FD2SI (Floating Point Convert To Signed Integer from Double)

Type: 32-Bit floating-point DP V1 extension

Format:

FD2SI

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FDa	11000 FD2SI	1111 F2OP	00 CP0	1000 FD1

FD2SI.z

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FDa	11100 FD2SI.z	1111 F2OP	00 CP0	1000 FD1

Syntax: FD2SI FSt, FDa
FD2SI.z FSt, Fda

Purpose: Convert a double-precision floating-point value to a 32-bit signed integer.

Description: The double-precision floating-point value in FDa register is converted to a signed integer value, rounded based on the rounding mode in the FPCSR register for the base form and rounded towards zero for the ".z" form. The result is written to FSt. If the double-precision value is Infinity, NaN, or the rounded result is outside the range of [2^{31}-1, -2^{31}], an Invalid Operation exception (IVO) is raised. If the IVO enable bit in the FPCSR is set, an FP Invalid Operation exception is taken (trapped). If the IVO enable bit is not set, then the IVO status flag in the FPCSR will be set and the following result will be written into FSt:

- +∞, or result > 2^{31}-1 : FSt = 0x7FFFFFFF
- -∞, or result < -2^{31} : FSt = 0x80000000
- NaN : FSt = 0xFFFFFFFF

This instruction will convert a denormalized number into a correctly-rounded integer.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
    FSt = ConvertDP2SI(FDa);
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When DP extension is not implemented, Register out-of-range)

Floating Point Exceptions: Invalid operation, Inexact

Privilege level: All

Note:

- The ".z" form is used for C, C++ languages.

FD2UI (Floating Point Convert To Unsigned Integer from Double)

Type: 32-Bit floating-point DP V1 extension

Format:

FD2UI

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FDa	10000 FD2UI	1111 F2OP	00 CP0	1000 FD1

FD2UI.z

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FSt	FDa	10100 FD2UI.z	1111 F2OP	00 CP0	1000 FD1

Syntax: FD2UI FSt, FDa
FD2UI.z FSt, FDa

Purpose: Convert a double-precision floating-point value to a 32-bit unsigned integer.

Description: The double-precision floating-point value in FDa register is converted to a 32-bit unsigned integer value, rounded based on the rounding mode in the FPCSR register for the base form and rounded towards zero for the ".z" form. The result is written to FSt. If the double-precision value is Infinity, NaN, or the rounded result is outside the range of [2^{32}-1, 0] (Note: -0 is treated as 0), an Invalid Operation exception (IVO) is raised. If the IVO enable bit in the FPCSR is set, an FP Invalid Operation exception is taken (trapped). If the IVO enable bit is not set, then the IVO status flag in the FPCSR will be set and the following result will be written into FSt:

- +∞, or result > 2^{32}-1 : FSt = 0xFFFFFFFF
- -∞, or result < 0 : FSt = 0x00000000
- NaN : FSt = 0xFFFFFFFF

This instruction will convert a denormalized number into a correctly-rounded integer.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
    FSt = ConvertDP2UI(FDa);
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When DP extension is not implemented, Register out-of-range)

Floating Point Exceptions: Invalid operation, Inexact

Privilege level: All

Note:

- The ".z" form is used for C, C++ languages.

FDIVD (Floating-point Divide Double-precision)

Type: 32-Bit floating-point DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FDt	FDa	FDb	1101 FDIVD	00 CP0	1000 FD1

Syntax: FDIVD FDt, FDa, FDb

Purpose: Divide the double-precision floating-point values in two registers.

Description: The double-precision floating-point value in FDa is divided by the double-precision floating-point value in FDb. And the floating-point result is written to FDt.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

The following table shows the results obtained when dividing various types of numbers.

Table 35. FDIVD results

		FDa								
FDb		**-∞**	**-N**	**-DN**	**-0**	**+0**	**+DN**	**+N**	**+∞**	**NaN**
	-∞	IVO	+0	+0	+0	-0	-0	-0	IVO	NaNa
	-N	+∞	+F	*	+0	-0	*	-F	-∞	NaNa
	-DN	+∞	*	*	+0	-0	*	*	-∞	NaNa
	-0	+∞	DBZ	DBZ	IVO	IVO	DBZ	DBZ	-∞	NaNa
	+0	-∞	DBZ	DBZ	IVO	IVO	DBZ	DBZ	+∞	NaNa
	+DN	-∞	*	*	-0	+0	*	*	+∞	NaNa
	+N	-∞	-F	*	-0	+0	*	+F	+∞	NaNa

		FDa								
	+∞	IVO	-0	-0	-0	+0	+0	+0	IVO	NaNa
	NaN	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNc

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
F	Means finite floating-point value (N, DN) which may involve underflow, overflow handling to become (N, DN, ∞).
IVO	Means invalid operation exception. If trapping is disabled, deliver the default QNaN as the result.
DBZ	Means divide-by-zero exception
NaNa	Means a QNaN from FDa or a QNaN converted from a SNaN from FDa by changing its most significant fraction bit from 0 to 1.
NaNb	Means a QNaN from FDb or a QNaN converted from a SNaN from FDb by changing its most significant fraction bit from 0 to 1.
NaNc	Means a QNaN converted from ● A SNaN from FDa or from FDb if FDa is a QNaN ● A QNaN from FDa if FDb is also a QNaN
*	Indicates F (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)

Operations:

```
FDt = FDa / FDb;
```

Exceptions: Reserved instruction (When FPU is not implemented)

FPU disabled (When the use of FPU is not enabled)

Reserved instruction (When DP extension is not implemented, Register out-of-range)

Denorm input (for non-denorm-support FPU)

Floating Point Exceptions: Invalid operation, Inexact, Overflow, Underflow, Divide by Zero

Privilege level: All

Note:

FMADDD (Floating-point Multiply and Add Double-precision)

Type: 32-Bit floating-point DP V1 extension (FMA optional)

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FDt	FDa	FDb	0100 FMADDD	00 CP0	1000 FD1

Syntax: FMADDD FDt, FDa, FDb

Purpose: Multiply the double-precision floating-point values of two registers and then accumulate the result to the third register.

Description: The double-precision floating-point value in FDa is multiplied with the double-precision floating-point value in FDb. And the multiplication result with unbounded range and precision is added with the floating-point value in FDt. The rounded addition result is then written back to FDt. This instruction is implemented if FPCFG.FMA register field is equal to 1.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

If (FDt, FDa, FDb) contains the following content as (c, 0, infinity) or (c, infinity, 0), an Invalid operation exception will be raised even if c is a QNaN.

The following table shows the intermediate results obtained when multiplying various types of numbers from FDa and FDb.

Table 36. FMADDD multiplication intermediate results

		FDa								
		-∞	-N	-DN	-0	+0	+DN	+N	+∞	NaN
FDb	-∞	+∞	+∞	+∞	IVO	IVO	-∞	-∞	-∞	NaN1
	-N	+∞	+Fu	*'	+0	-0	*'	-Fu	-∞	NaN1
	-DN	+∞	*'	*'	+0	-0	*'	*'	-∞	NaN1
	-0	IVO	+0	+0	+0	-0	-0	-0	IVO	NaN1
	+0	IVO	-0	-0	-0	+0	+0	+0	IVO	NaN1
	+DN	-∞	*'	*'	-0	+0	*'	*'	+∞	NaN1
	+N	-∞	-Fu	*'	-0	+0	*'	+Fu	+∞	NaN1
	+∞	-∞	-∞	-∞	IVO	IVO	+∞	+∞	+∞	NaN1
	NaN	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN3

The following table shows the final results obtained when adding various types of numbers from FDt and the intermediate result from multiplication.

Table 37. FMADDD results

		Intermediate result from multiplication (MIR)								
		-∞	-Fu	IVO	-0	+0	DIE	+Fu	+∞	NaN
FDt	-∞	-∞	-∞	IVO1	-∞	-∞	-∞	-∞	IVO1	NaNm
	-N	-∞	-F	IVO1	FDt	FDt	DIE	±F	+∞	NaNm
	-DN	-∞	*	IVO1	*	*	DIE	*	+∞	NaNm
	-0	-∞	MIR'	IVO1	-0	±0	DIE	MIR'	+∞	NaNm
	+0	-∞	MIR'	IVO1	±0	+0	DIE	MIR'	+∞	NaNm
	+DN	-∞	*	IVO1	*	*	DIE	*	+∞	NaNm
	+N	-∞	±F	IVO1	FDt	FDt	DIE	+F	+∞	NaNm
	+∞	IVO1	+∞	IVO1	+∞	+∞	+∞	+∞	+∞	NaNm
	NaN	NaNt	NaNt	IVO2	NaNt	NaNt	NaNt	NaNt	NaNt	NaNn

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
Fu	Means a finite floating-point value (N, DN) with unbounded range and precision.
F	Means a finite floating-point value (N, DN) which may involve underflow, overflow handling to become (N, DN, ∞).
IVO	Means invalid operation exception
IVO1	Means invalid operation exception. If trapping is disabled, deliver the default QNaN as the result.
IVO2	Means invalid operation exception. If trapping is disabled, deliver NaNt (see below) as the result.
DIE	Means denorm input exception
MIR'	Means the properly rounded MIR
NaN1	Means the NaN from FDa (it can be either SNaN or QNaN).
NaN2	Means the NaN from FDb (it can be either SNaN or QNaN).
NaN3	Means a NaN from ● the SNaN from FDa or the SNaN from FDb if FDa is a QNaN ● the QNaN from FDa if FDb is also a QNaN
NaNt	Means a QNaN from FDt or a QNaN converted from a SNaN from FDt by changing its most significant fraction bit from 0 to 1.
NaNm	Means a QNaN from MIR or a QNaN converted from a SNaN from MIR by changing its most significant fraction bit from 0 to 1.
NaNn	Means a QNaN converted from ● the SNaN from FDt or the SNaN from MIR if FDt is a QNaN ● the QNaN from FDt if MIR is also a QNaN
*'	Indicates Fu (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)
*	Indicates F (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)

When (a x b) + c is exactly zero, the sign of the result shall be determined by the rules described by the FADDD description. When the exact result of (a x b) + c is non-zero yet the result is zero because of rounding, the zero results takes the sign of the exact result.

Operations:

```
FDt = FDt + (FDa * FDb);
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When DP extension is not implemented, Register out-of-range)
Denorm input (for non-denorm-support FPU)

Floating-point Exceptions: Invalid operation, Inexact, Overflow, Underflow

Privilege level: All

Note:

FMSUBD (Floating-point Multiply and Subtraction Double-precision)

Type: 32-Bit floating-point DP V1 extension (FMA optional)

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FDt	FDa	FDb	0101 FMSUBD	00 CP0	1000 FD1

Syntax: FMSUBD FDt, FDa, FDb

Purpose: Multiply the double-precision floating-point values of two registers and then subtract the result from the third register.

Description: The double-precision floating-point value in FDa is multiplied with the double-precision floating-point value in FDb. And the multiplication result with unbounded range and precision is subtracted from the floating-point value in FDt. The rounded subtraction result is then written back to FDt. This instruction is implemented if FPCFG.FMA register field is equal to 1.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

If (FDt, FDa, FDb) contains the following content as (c, 0, infinity) or (c, infinity, 0), an Invalid operation exception will be raised even if c is a QNaN.

The following table shows the intermediate results obtained when multiplying various types of numbers from FDa and FDb.

Table 38. FMSUBD multiplication intermediate results

		FDb								
		-∞	-N	-DN	-0	+0	+DN	+N	+∞	NaN
FDa	-∞	+∞	+∞	+∞	IVO	IVO	-∞	-∞	-∞	NaN1
	-N	+∞	+Fu	*’	+0	-0	*’	-Fu	-∞	NaN1
	-DN	+∞	*’	*’	+0	-0	*’	*’	-∞	NaN1
	-0	IVO	+0	+0	+0	-0	-0	-0	IVO	NaN1
	+0	IVO	-0	-0	-0	+0	+0	+0	IVO	NaN1
	+DN	-∞	*’	*’	-0	+0	*’	*’	+∞	NaN1
	+N	-∞	-Fu	*’	-0	+0	*’	+Fu	+∞	NaN1
	+∞	-∞	-∞	-∞	IVO	IVO	+∞	+∞	+∞	NaN1
	NaN	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN3

The following table shows the final results obtained when subtracting various types of numbers from FDt and the intermediate result from multiplication.

Table 39. FMSUBD results

		Intermediate result from multiplication (MIR)								
		-∞	-Fu	IVO	-0	+0	DIE	+Fu	+∞	NaN
FDt	-∞	IVO1	-∞	IVO1	-∞	-∞	-∞	-∞	-∞	NaNm
	-N	+∞	±F	IVO1	FDt	FDt	DIE	-F	-∞	NaNm
	-DN	+∞	*	IVO1	*	*	DIE	*	-∞	NaNm
	-0	+∞	MIR’	IVO1	±0	-0	DIE	MIR’	-∞	NaNm
	+0	+∞	MIR’	IVO1	+0	±0	DIE	MIR’	-∞	NaNm
	+DN	+∞	*	IVO1	*	*	DIE	*	-∞	NaNm
	+N	+∞	+F	IVO1	FDt	FDt	DIE	±F	-∞	NaNm
	+∞	+∞	+∞	IVO1	+∞	+∞	+∞	+∞	IVO1	NaNm
	NaN	NaNt	NaNt	IVO2	NaNt	NaNt	NaNt	NaNt	NaNt	NaNn

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
Fu	Means a finite floating-point value (N, DN) with unbounded range and precision.
F	Means a finite floating-point value (N, DN) which may involve underflow, overflow handling to become (N, DN, ∞).
IVO	Means invalid operation exception
IVO1	Means invalid operation exception. If trapping is disabled, deliver the default QNaN as the result.
IVO2	Means invalid operation exception. If trapping is disabled, deliver NaNt (see below) as the result.
DIE	Means denorm input exception
MIR’	Means the properly rounded MIR
NaN1	Means the NaN from FDa (it can be either SNaN or QNaN).
NaN2	Means the NaN from FDb (it can be either SNaN or QNaN).
NaN3	Means a NaN from ● the SNaN from FDa or the SNaN from FDb if FDa is a QNaN ● the QNaN from FDa if FDb is also a QNaN
NaNt	Means a QNaN from FDt or a QNaN converted from a SNaN from FDt by changing its most significant fraction bit from 0 to 1.
NaNm	Means a QNaN from MIR or a QNaN converted from a SNaN from MIR by changing its most significant fraction bit from 0 to 1.
NaNn	Means a QNaN converted from ● the SNaN from FDt or the SNaN from MIR if FDt is a QNaN ● the QNaN from FDt if MIR is also a QNaN
*’	Indicates Fu (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)
*	Indicates F (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)

When *c* - (*a x b*) is exactly zero, the sign of the result shall be determined by the rules described by the FSUBD description. When the exact result of *c* - (*a x b*) is non-zero yet the result is zero because of rounding, the zero results takes the sign of the exact result.

Operations:

```
        FDt = FDt - (FDa * FDb);
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When DP extension is not implemented, Register out-of-range)
Denorm input (for non-denorm-support FPU)

Floating-point Exceptions: Invalid operation, Inexact, Overflow, Underflow

Privilege level: All

Note:

FMULD (Floating-point Multiplication Double-precision)

Type: 32-Bit floating-point DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FDt	FDa	FDb	1100 FMULD	00 CP0	1000 FD1

Syntax: FMULD FDt, FDa, FDb

Purpose: Multiply the double-precision floating-point values in two registers.

Description: The double-precision floating-point value in FDa is multiplied with the double-precision floating-point value in FDb. And the floating-point result is written to FDt.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

The following table shows the results obtained when multiplying various types of numbers.

Table 40. FMULD results

	FDa									
		-∞	**-N**	**-DN**	**-0**	**+0**	**+DN**	**+N**	**+∞**	**NaN**
FDb	**-∞**	+∞	+∞	+∞	IVO	IVO	-∞	-∞	-∞	NaNa
	-N	+∞	+F	*	+0	-0	*	-F	-∞	NaNa
	-DN	+∞	*	*	+0	-0	*	*	-∞	NaNa
	-0	IVO	+0	+0	+0	-0	-0	-0	IVO	NaNa
	+0	IVO	-0	-0	-0	+0	+0	+0	IVO	NaNa
	+DN	-∞	*	*	-0	+0	*	*	+∞	NaNa
	+N	-∞	-F	*	-0	+0	*	+F	+∞	NaNa

	FDa									
	+∞	-∞	-∞	-∞	IVO	IVO	+∞	+∞	+∞	NaNa
	NaN	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNc

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
F	Means a finite floating-point value (N, DN) which may involve underflow, overflow handling to become (N, DN, ∞).
IVO	Means invalid operation exception. If trapping is disabled, deliver the default QNaN as the result.
NaNa	Means a QNaN from FDa or a QNaN converted from a SNaN from FDa by changing its most significant fraction bit from 0 to 1.
NaNb	Means a QNaN from FDb or a QNaN converted from a SNaN from FDb by changing its most significant fraction bit from 0 to 1.
NaNc	Means a QNaN converted from ● A SNaN from FDa or from FDb if FDa is a QNaN ● A QNaN from FDa if FDb is also a QNaN
*	Indicates F (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)

Operations:

```
FDt = FDa * FDb;
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When DP extension is not implemented, Register out-of-range)
Denorm input (for non-denorm-support FPU)

Floating Point Exceptions: Invalid operation, Inexact, Overflow, Underflow

Privilege level: All

Note:

FNMADDD (Floating-point Negate Multiply and Add Double-precision)

Type: 32-Bit floating-point DP V1 extension (FMA optional)

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FDt	FDa	FDb	1000 FNMADDD	00 CP0	1000 FD1

Syntax: FNMADDD FDt, FDa, FDb

Purpose: Multiply the double-precision floating-point values of two registers and then accumulate and negate the result to the third register.

Description: The double-precision floating-point value in FDa is multiplied with the double-precision floating-point value in FDb. And the multiplication result with unbounded range and precision is added with the floating-point value in FDt. The rounded addition result is negated and then written back to FDt. This instruction is implemented if FPCFG.FMA register field is equal to 1.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

If (FDt, FDa, FDb) contains the following content as (c, 0, infinity) or (c, infinity, 0), an Invalid operation exception will be raised even if c is a QNaN.

The following table shows the intermediate results obtained when multiplying various types of numbers from FDa and FDb.

Table 41. FNMADDD multiplication intermediate results

		FDa								
		-∞	**-N**	**-DN**	**-0**	**+0**	**+DN**	**+N**	**+∞**	**NaN**
FDb	**-∞**	+∞	+∞	+∞	IVO	IVO	-∞	-∞	-∞	NaN1
	-N	+∞	+Fu	*'	+0	-0	*'	-Fu	-∞	NaN1
	-DN	+∞	*'	*'	+0	-0	*'	*'	-∞	NaN1
	-0	IVO	+0	+0	+0	-0	-0	-0	IVO	NaN1
	+0	IVO	-0	-0	-0	+0	+0	+0	IVO	NaN1
	+DN	-∞	*'	*'	-0	+0	*'	*'	+∞	NaN1
	+N	-∞	-Fu	*'	-0	+0	*'	+Fu	+∞	NaN1
	+∞	-∞	-∞	-∞	IVO	IVO	+∞	+∞	+∞	NaN1
	NaN	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN3

The following table shows the final results obtained when adding and negating various types of numbers from FDt and the intermediate result from multiplication.

Table 42. FNMADDD results

		Intermediate result from multiplication (MIR)								
		-∞	**-Fu**	**IVO**	**-0**	**+0**	**DIE**	**+Fu**	**+∞**	**NaN**
FDt	**-∞**	+∞	+∞	IVO1	+∞	+∞	+∞	+∞	IVO1	NaNm
	-N	+∞	+F	IVO1	-FDt	-FDt	DIE	∓F	-∞	NaNm
	-DN	+∞	*	IVO1	*	*	DIE	*	-∞	NaNm
	-0	+∞	-MIR'	IVO1	+0	∓0	DIE	-MIR'	-∞	NaNm
	+0	+∞	-MIR'	IVO1	∓0	-0	DIE	-MIR'	-∞	NaNm
	+DN	+∞	*	IVO1	*	*	DIE	*	-∞	NaNm
	+N	+∞	∓F	IVO1	-FDt	-FDt	DIE	-F	-∞	NaNm
	+∞	IVO1	-∞	IVO1	-∞	-∞	-∞	-∞	-∞	NaNm
	NaN	NaNt	NaNt	IVO2	NaNt	NaNt	NaNt	NaNt	NaNt	NaNn

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
Fu	Means a finite floating-point value (N, DN) with unbounded range and precision.
F	Means a finite floating-point value (N, DN) which may involve underflow, overflow handling to become (N, DN, ∞).
IVO	Means invalid operation exception
IVO1	Means invalid operation exception. If trapping is disabled, deliver the default QNaN as the result.
IVO2	Means invalid operation exception. If trapping is disabled, deliver NaNt (see below) as the result.
DIE	Means denorm input exception
MIR'	Means the properly rounded MIR
NaN1	Means the NaN from FDa (it can be either SNaN or QNaN).
NaN2	Means the NaN from FDb (it can be either SNaN or QNaN).
NaN3	Means a NaN from ● the SNaN from FDa or the SNaN from FDb if FDa is a QNaN ● the QNaN from FDa if FDb is also a QNaN
NaNt	Means a QNaN from FDt or a QNaN converted from a SNaN from FDt by changing its most significant fraction bit from 0 to 1.
NaNm	Means a QNaN from MIR or a QNaN converted from a SNaN from MIR by changing its most significant fraction bit from 0 to 1.
NaNn	Means a QNaN converted from ● the SNaN from FDt or the SNaN from MIR if FDt is a QNaN ● the QNaN from FDt if MIR is also a QNaN
*'	Indicates Fu (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)
*	Indicates F (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)

When $(a \times b) + c$ is exactly zero, the sign of the result shall be determined by negating the rules described by the FADDD description. When the exact result of $(a \times b) + c$ is non-zero yet the result is zero because of rounding, the zero results takes the negating sign of the exact result.

Operations:

```
    FDt = - (FDt + (FDa * FDb));
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When DP extension is not implemented, Register out-of-range)
Denorm input (for non-denorm-support FPU)

Floating-point Exceptions: Invalid operation, Inexact, Overflow, Underflow

Privilege level: All

Note:

FNMSUBD (Floating-point Negate Multiply and Subtraction Double-precision)

Type: 32-Bit floating-point DP V1 extension (FMA optional)

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FDt	FDa	FDb	1001 FNMSUBD	00 CP0	1000 FD1

Syntax: FNMSUBD FDt, FDa, FDb

Purpose: Multiply the double-precision floating-point values of two registers and then subtract the value of the third register from the multiplication result.

Description: The double-precision floating-point value in FDa is multiplied with the double-precision floating-point value in FDb. And the multiplication result with unbounded range and precision is being subtracted by the floating-point value in FDt. The rounded subtraction result is then written back to FDt. It is equivalent to a negation of the result generated by a FMSUBD instruction. This instruction is implemented if FPCFG.FMA register field is equal to 1.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

If (FDt, FDa, FDb) contains the following content as (c, 0, infinity) or (c, infinity, 0), an Invalid operation exception will be raised even if c is a QNaN.

The following table shows the intermediate results obtained when multiplying various types of numbers from FDa and FDb.

Table 43. FNMSUBD multiplication intermediate results

		FDb								
		-∞	-N	-DN	-0	+0	+DN	+N	+∞	NaN
FDa	-∞	+∞	+∞	+∞	IVO	IVO	-∞	-∞	-∞	NaN1
	-N	+∞	+Fu	*’	+0	-0	*’	-Fu	-∞	NaN1
	-DN	+∞	*’	*’	+0	-0	*’	*’	-∞	NaN1
	-0	IVO	+0	+0	+0	-0	-0	-0	IVO	NaN1
	+0	IVO	-0	-0	-0	+0	+0	+0	IVO	NaN1
	+DN	-∞	*’	*’	-0	+0	*’	*’	+∞	NaN1
	+N	-∞	-Fu	*’	-0	+0	*’	+Fu	+∞	NaN1
	+∞	-∞	-∞	-∞	IVO	IVO	+∞	+∞	+∞	NaN1
	NaN	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN2	NaN3

The following table shows the final results obtained when subtracting various types of numbers from FDt and the intermediate result from multiplication.

Table 44. FNMSUBD results

		Intermediate result from multiplication (MIR)								
		-∞	-Fu	IVO	-0	+0	DIE	+Fu	+∞	NaN
FDt	-∞	IVO1	+∞	IVO1	+∞	+∞	+∞	+∞	+∞	NaNm
	-N	-∞	∓F	IVO1	-FDt	-FDt	DIE	+F	+∞	NaNm
	-DN	-∞	*	IVO1	*	*	DIE	*	+∞	NaNm
	-0	-∞	MIR’	IVO1	∓0	+0	DIE	MIR’	+∞	NaNm
	+0	-∞	MIR’	IVO1	-0	∓0	DIE	MIR’	+∞	NaNm
	+DN	-∞	*	IVO1	*	*	DIE	*	+∞	NaNm
	+N	-∞	-F	IVO1	-FDt	-FDt	DIE	∓F	+∞	NaNm
	+∞	-∞	-∞	IVO1	-∞	-∞	-∞	-∞	IVO1	NaNm
	NaN	NaNt	NaNt	IVO2	NaNt	NaNt	NaNt	NaNt	NaNt	NaNn

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
Fu	Means a finite floating-point value (N, DN) with unbounded range and precision.
F	Means a finite floating-point value (N, DN) which may involve underflow, overflow handling to become (N, DN, ∞).
IVO	Means invalid operation exception
IVO1	Means invalid operation exception. If trapping is disabled, deliver the default QNaN as the result.
IVO2	Means invalid operation exception. If trapping is disabled, deliver NaNt (see below) as the result.
DIE	Means denorm input exception
MIR’	Means the properly rounded MIR
NaN1	Means the NaN from FDa (it can be either SNaN or QNaN).
NaN2	Means the NaN from FDb (it can be either SNaN or QNaN).
NaN3	Means a NaN from ● the SNaN from FDa or the SNaN from FDb if FDa is a QNaN ● the QNaN from FDa if FDb is also a QNaN
NaNt	Means a QNaN from FDt or a QNaN converted from a SNaN from FDt by changing its most significant fraction bit from 0 to 1.
NaNm	Means a QNaN from MIR or a QNaN converted from a SNaN from MIR by changing its most significant fraction bit from 0 to 1.
NaNn	Means a QNaN converted from ● the SNaN from FDt or the SNaN from MIR if FDt is a QNaN ● the QNaN from FDt if MIR is also a QNaN
*’	Indicates Fu (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)
*	Indicates F (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)

When *c* - (*a x b*) is exactly zero, the sign of the result shall be determined by negating the rules described by the FSUBD description. When the exact result of *c* - (*a x b*) is non-zero yet the result is zero because of rounding, the zero results takes the negating sign of the exact result.

Operations:

```
      FDt = - (FDt - (FDa * FDb));
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When DP extension is not implemented, Register out-of-range)
Denorm input (for non-denorm-support FPU)

Floating-point Exceptions: Invalid operation, Inexact, Overflow, Underflow

Privilege level: All

Note:

FSI2D (Floating Point Convert From Signed Integer Double-precision)

Type: 32-Bit floating-point DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FDt	FSa	01100 FSI2D	1111 F2OP	00 CP0	1000 FD1

Syntax: FSI2D FDt, Fsa

Purpose: Convert a 32-bit signed integer to a double-precision floating point value.

Description: The signed integer value in FSa is converted to a double-precision floating-point value. The floating-point result is rounded based on the rounding mode in FPCSR and the rounded result is written to FDt. An integer zero is converted to +0, not -0.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
    FDt = ConvertSI2D(SI(FSa));
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When DP extension is not implemented, Register out-of-range)

Floating Point Exceptions:

Privilege level: All

Note:

FSQRTD (Floating-point Square Root Double-precision)

Type: 32-Bit floating-point DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FDt	FDa	00001 FSQRTD	1111 F2OP	00 CP0	1000 FD1

Syntax: FSQRTD FDt, Fda

Purpose: Compute square root from a double-precision floating-point value.

Description: The square root of the double-precision floating-point value in FDa is computed by this instruction. The floating-point result is rounded based on the rounding mode in FPCSR and the rounded result is written to FDt. If the floating-point value in FDa is -0, the result is -0. If the floating-point value in FDa is less than 0, an Invalid Operation exception is generated.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

The following table shows the results obtained when taking the square root of various types of numbers.

Table 45. FSQRTD results

FDa	FDt
-∞	IVO
-N	IVO
-DN	IVO
-0	-0

FDa	FDt
+0	+0
+DN	*
+N	+F
+∞	+∞
NaN	NaN’

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
F	Means a finite floating-point value (N, DN) which may involve underflow, overflow handling to become (N, DN, ∞).
IVO	Means invalid operation exception. If trapping is disabled, deliver the default QNaN as the result.
NaN’	Means a QNaN from FDa or a QNaN converted from a SNaN from FDa by changing its most significant fraction bit from 0 to 1.
*	Indicates F (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)

Note that IEEE-754 standard states that

“*Except that SquareRoot(-0) shall be −0, every valid square root shall have a positive sign.*”

Operations:

```
    FDt = SQRT(FDa);
```

Exceptions: Reserved instruction (When FPU is not implemented)

FPU disabled (When the use of FPU is not enabled)

Reserved instruction (When SP extension is not implemented, Register out-of-range)

Denorm input (for non-denorm-support FPU)

Floating Point Exceptions: Invalid operation, Inexact

Privilege level: All

Note:

FSUBD (Floating-point Subtraction Double-precision)

Type: 32-Bit floating-point DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FDt	FDa	FDb	0001 FSUBD	00 CP0	1000 FD1

Syntax: FSUBD FDt, FDa, FDb

Purpose: Subtract the double-precision floating-point values in two registers.

Description: The double-precision floating-point value in FDb is subtracted from the double-precision floating-point value in FDa. And the floating-point result is written to FDt.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

The following table shows the results obtained when subtracting various types of numbers.

Table 46. FSUBD results

		FDa								
		-∞	**-N**	**-DN**	**-0**	**+0**	**+DN**	**+N**	**+∞**	**NaN**
FDb	**-∞**	IVO	+∞	+∞	+∞	+∞	+∞	+∞	+∞	NaNa
	-N	-∞	±F	*	-FDb	-FDb	*	+F	+∞	NaNa
	-DN	-∞	*	*	*	*	*	*	+∞	NaNa
	-0	-∞	FDa	*	±0	+0	*	FDa	+∞	NaNa
	+0	-∞	FDa	*	-0	±0	*	FDa	+∞	NaNa
	+DN	-∞	*	*	*	*	*	*	+∞	NaNa

	FDa								
+N	-∞	-F	*	-FDb	-FDb	*	±F	+∞	NaNa
+∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞	IVO	NaNa
NaN	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNb	NaNc

Notes:

N	Means normalized finite floating-point value.
DN	Means denormalized finite floating-point value.
F	Means a finite floating-point value (N, DN) which may involve underflow, overflow handling to become (N, DN, ∞).
IVO	Means invalid operation exception. If trapping is disabled, deliver the default QNaN as the result.
NaNa	Means a QNaN from FDa or a QNaN converted from a SNaN from FDa by changing its most significant fraction bit from 0 to 1.
NaNb	Means a QNaN from FDb or a QNaN converted from a SNaN from FDb by changing its most significant fraction bit from 0 to 1.
NaNc	Means a QNaN converted from ● A SNaN from FDa or from FDb if FDa is a QNaN ● A QNaN from FDa if FDb is also a QNaN
*	Indicates F (if denormal calculation is supported) or denorm input exception (if denormal calculation is not supported and Flush-to-zero mode is disabled) or the same result as if the DN operand is a correctly signed zero (if denormal calculation is not supported and Flush-to-zero mode is enabled)

Note that IEEE-754 standard states that

“*When the difference of two operands with like signs is exactly zero, the sign of that difference shall be + in all rounding modes except round toward* -∞, *in which mode that sign shall be –. However, x – (– x) retains the same sign as x even when x is zero.*”

Operations:

```
FDt = FDa - FDb;
```

Exceptions: Reserved instruction (When FPU is not implemented)

FPU disabled (When the use of FPU is not enabled)

Reserved instruction (When DP extension is not implemented, Register out-of-range)

Denorm input (for non-denorm-support FPU)

Floating Point Exceptions: Invalid operation, Inexact, Overflow, Underflow

Privilege level: All

Note:

FUI2D (Floating Point Convert From Unsigned Integer Double-precision)

Type: 32-Bit floating-point DP V1 extension

Format:

31	30 25	24 20	19 15	14 10	9 6	5 4	3 0
0	COP 110101	FDt	FSa	01000 FUI2D	1111 F2OP	00 CP0	1000 FD1

Syntax: FUI2D FDt, Fsa

Purpose: Convert a 32-bit unsigned integer to a double-precision floating point value.

Description: The unsigned integer value in FSa is converted to a double-precision floating-point value. The floating-point result is rounded based on the rounding mode in FPCSR and the rounded result is written to FDt. An integer zero is converted to +0, not -0.

The floating-point register numbers are in the range of [0, Max-1] defined by the FPCFG.FREG field. Any register number outside this range will generate a Reserved Instruction exception.

Operations:

```
FDt = ConvertSI2D(UI(FSa));
```

Exceptions: Reserved instruction (When FPU is not implemented)
FPU disabled (When the use of FPU is not enabled)
Reserved instruction (When DP extension is not implemented, Register out-of-range)

Floating Point Exceptions:

Privilege level: All

Note:

4. Related Register definitions

4.1. Floating-Point Control Status Register

Type: FPU internal register

Mnemonic Name: FPCSR

12	11	10	9	8	7	6	5	4	3	2	1 0
DNZ	IEXE	UDFE	OVFE	DBZE	IVOE	IEX	UDF	OVF	DBZ	IVO	RM

31 20	19	18	17	16	15	14	13
Reserved	RIT	DNIT	IEXT	UDFT	OVFT	DBZT	IVOT

The Floating-Point Control Status register contains several fields which controls the rounding mode behavior, the IEEE exception trapping behavior, the demoralized number handling mode, and records the IEEE exception status.

This register can be read by using the FMFCSR instruction and can be written by using the FMTCSR instruction in both user and superuser mode. The IEEE cumulative exception flags will be set by hardware when an untrapped IEEE floating-point exception has happened. The floating-point exception type fields will be updated by hardware when a trapped floating-point exception has been taken.

A DSB instruction is needed after the FMTCSR instruction in order for the following floating-point instruction to see the updated FPCSR content and its side effects.

Field Name	Bits	Description	Type	Reset
RM	2 (1,0)	Rounding modes. Value: Meaning 0: Round to Nearest Even 1: Round towards Plus Infinity 2: Round towards Minus Infinity 3: Round towards Zero	RW	0
IVO	1 (2)	IEEE Invalid Operation (IVO) cumulative exception flag. It is set when an untrapped IVO exception has happened.	RW	0
DBZ	1 (3)	IEEE Divide by Zero (DBZ) cumulative exception flag. It is set when an untrapped DBZ exception has happened.	RW	0
OVF	1 (4)	IEEE Overflow (OVF) cumulative exception flag. It is set when an untrapped OVF exception has happened.	RW	0
UDF	1 (5)	IEEE Underflow (UDF) cumulative exception flag. It is set when an untrapped UDF exception has happened.	RW	0
IEX	1 (6)	IEEE Inexact (IEX) cumulative exception flag. It is set when an untrapped IEX exception has happened.	RW	0
IVOE	1 (7)	IEEE Invalid Operation (IVO) exception trapping enable. Value: Meaning 0: IVO exception trapping is disabled. 1: IVO exception trapping is enabled.	RW	0

Field Name	Bits	Description	Type	Reset
DBZE	1 (8)	IEEE Divide by Zero (DBZ) exception trapping enable. Value / Meaning: 0 — DBZ exception trapping is disabled. 1 — DBZ exception trapping is enabled.	RW	0
OVFE	1 (9)	IEEE Overflow (OVF) exception trapping enable. Value / Meaning: 0 — OVF exception trapping is disabled. 1 — OVF exception trapping is enabled.	RW	0
UDFE	1 (10)	IEEE Underflow (UDF) exception trapping enable. Value / Meaning: 0 — UDF exception trapping is disabled. 1 — UDF exception trapping is enabled.	RW	0

Field Name	Bits	Description	Type	Reset
IEXE	1 (11)	IEEE Ineaxct (IEX) exception trapping enable. Value: 0 — IEX exception trapping is disabled. Value: 1 — IEX exception trapping is enabled.	RW	0
DNZ	1 (12)	Denormalized flush-to-Zero mode. Value: 0 — Denormalized number Flush-to-Zero mode is off. Denormalized inputs will generate a Denorm input arithmetic exception. Denormalized result will generate an underflow exception. Value: 1 — Denormalized number Flush-to-Zero mode is on. Denormalized inputs and result will be turned into 0 and no Denorm input and underflow arithmetic exception is generated.	RW	0

Field Name	Bits	Description	Type	Reset
IVOT	1 (13)	Invalid Operation exception type. This bit is set when Andes core decides to take a FPU exception and the exception type is a trappable invalid operation exception. This bit is cleared when Andes core decides to take a different type of FPU exception.	RO	0
DBZT	1 (14)	Divide-by-zero exception type. This bit is set when Andes core decides to take a FPU exception and the exception type is a trappable divide-by-zero exception. This bit is cleared when Andes core decides to take a different type of FPU exception.	RO	0
OVFT	1 (15)	Overflow exception type. This bit is set when Andes core decides to take a FPU exception and the exception type is a trappable overflow exception. This bit is cleared when Andes core decides to take a different type of FPU exception.	RO	0
UDFT	1 (16)	Underflow exception type. This bit is set when Andes core decides to take a FPU exception and the exception type is a trappable underflow exception. This bit is cleared when Andes core decides to take a different type of FPU exception.	RO	0
IEXT	1 (17)	Inexact exception type. This bit is set when Andes core decides to take a FPU exception and the exception type is a trappable inexact exception. This bit is cleared when Andes core decides to take a different type of FPU exception.	RO	0

Field Name	Bits	Description	Type	Reset
DNIT	1 (18)	Denormalized-input exception type. This bit is set when Andes core decides to take a FPU exception and the exception type is a denormalized-input exception. This bit is cleared when Andes core decides to take a different type of FPU exception.	RO	0
RIT	1 (19)	Reserved instruction exception type. This bit is set when Andes core decides to take a FPU exception (i.e. CP0 exception) and the exception type is a Reserved instruction exception. This bit is cleared when Andes core decides to take a different type of FPU exception.	RO	0
Reserved	12 (31,20)		RAZWI	0

4.2. Floating-Point Configuration Register

Type: FPU internal register

Mnemonic Name: FPCFG

The Floating-point Configuration register contains floating-point extension implementation information such as single-precision, double-precision capabilities, ISA version number, and the number of floating-point registers.

This register can be read by using FMFCFG instruction in both user and superuser mode. And this register cannot be written.

31 27	26 22	21 5	4	3 2	1	0
AVER	IMVER		FMA	FREG	DP	SP

Field Name	Bits	Description	Type	Reset
SP	1 (0)	Indicates if SP extension exists?	RO	IM
DP	1 (1)	Indicates if DP extension exists?	RO	IM
FREG	2 (3,2)	The number of single/double-precision floating-point registers implemented.	RO	IM

Value	Meaning
0	8 SP / 4 DP registers
1	16 SP / 8 DP registers
2	32 SP / 16 DP registers
3	32 SP / 32 DP registers

Field Name	Bits	Description	Type	Reset
FMA	1 (4)	Indicates if fused-multiply-add instructions are supported in FPU or not. Value | Meaning 0 | No fused-multiply-add instructions. 1 | Has fused-multiply-add instructions. The involved instructions are listed as follows: FMADDS / FMADDD FMSUBS / FMSUBD FNMADDS / FNMADDD FNSUBS / FNMSUBD	RO	IM
Reserved	17 (21,5)	RAZWI		0
IMVER	5 (26,22)	Indicates the FPU implementation and revision number.	RO	IM
AVER	5 (31,27)	Indicates the FPU ISA extension architecture version number.	RO	IM

4.3. Additional definitions for CPU_VER register

To indicate if there is any coprocessor (or FPU) extension support in a CPU core, a new definition for the CFGID field (bit 3) is added as follows:

31 24	23 16	15 0
CPUID	REV	CFGID

Field name	Bits	Description	Type	Reset
Config ID (CFGID)	16 (15,0)	Used to distinguish any other minor configuration differences. The current definition is as follows:	RO	IM

Bit (n)	Meaning
Bit (0)	Performance extension exists?
Bit (1)	16-bit extension exists?
Bit (2)	Performance extension 2 exists?
Bit (3)	COP/FPU extension exists?
Bit (4)	String extension exists?

4.4. Additional definitions for ITYPE register

For the arithmetic exception in the general exception vector entry point, the following fields are defined as follows:

31	30 22	21 20	19 16	15 5	4	3 0
0	Reserved	CPID	Sub_Type	Reserved	Inst	Exc Type

Field Name	Bits	Description	Type	Reset
SUB_TYPE	4 (19,16)	1. Indicates arithmetic exception type when an arithmetic exception has happened. The encoding is as follows: (see table below) 2. Indicates coprocessor exception type when a coprocessor exception has happened. The encoding is as follows: (see table below)	RO	IM

1. Indicates arithmetic exception type when an arithmetic exception has happened. The encoding is as follows:

Value	Meaning
0	Reserved
1	INT Divide by Zero (for DIV, DIVS)
2	Integer Overflow (for DIVS)
3-15	-

2. Indicates coprocessor exception type when a coprocessor exception has happened. The encoding is as follows:

Value	Meaning
0	Reserved
1	Coprocessor disabled exception
2	Coprocessor exception. Please check coprocessor exception status for details.
3-15	-

Field Name	Bits	Description	Type	Reset
CPID	2 (21,20)	Indicates the ID number of the coprocessor which generates the exception.		

4.5. FPU and Coprocessor Existence Configuration Register

Type: AndesCore configuration system register

Mnemonic Name: cr6 (FUCOP_EXIST)

IM Requirement: Required

Access Mode: Superuser

SR Value {Major, Minor, Extension}: {0, 5, 0}

This system register indicates the existence status of the coprocessors and the floating-point unit. Note that the floating-point unit existence is a combination of "CP0EX" and "CP0ISFPU" (i.e. CR6.CP0EX is 1 and CR6.CP0ISFPU is 1).

31	30 … 4	3	2	1	0
CP0ISFPU		CP3EX	CP2EX	CP1EX	CP0EX

Field Name	Bits	Description	Type	Reset
CP0EX	1 (0)	Coprocessor #0 existence status bit. <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Coprocessor #0 does not exist. Any encountering of Coprocessor #0 instruction will cause "Reserved instruction" exception.</td></tr><tr><td>1</td><td>Coprocessor #0 exists.</td></tr></table>	RO	IM
CP1EX	1 (1)	Coprocessor #1 existence status bit. <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Coprocessor #1 does not exist. Any encountering of coprocessor #1 instruction will cause "Reserved</td></tr></table>	RO	IM

Field Name	Bits	Description	Type	Reset
		instruction" exception. 1: Coprocessor #1 exists.		
CP2EX	1 (2)	Coprocessor #2 existence status bit. Value: Meaning 0: Coprocessor #2 does not exist. Any encountering of coprocessor #2 instruction will cause "Reserved instruction" exception. 1: Coprocessor #2 exists.	RO	IM
CP3EX	1 (3)	Coprocessor #3 existence status bit. Value: Meaning 0: Coprocessor #3 does not exist. Any encountering of coprocessor #3 instruction will cause "Reserved instruction" exception. 1: Coprocessor #3 exists.	RO	IM
Reserved	27 (30,4)	RAZWI	-	0
CP0ISFPU	1 (31)	Indicates if Coprocessor #0 is FPU or not when CP0EX is 1. Value: Meaning 0: Coprocessor #0 is not FPU when CP0EX is 1. 1: Coprocessor #0 is FPU when CP0EX is 1.	RO	IM

4.6. FPU and Coprocessor Enable Control Register

Type: AndesCore system register

Mnemonic Name: fucpr (FUCOP_CTL)

IM Requirement: Required

Access Mode: Superuser

SR Value {Major, Minor, Extension}: {4, 5, 0}

This system register controls enabling/disabling of audio extension, coprocessors, and the floating-point unit. Note that the floating-point unit enable is "CP0EN" when the floating-point unit is supported (i.e. CR6.CP0EX is 1 and CR6.CP0ISFPU is 1).

31	30 … 4	3	2	1	0
AUEN		CP3EN	CP2EN	CP1EN	CP0EN

Field Name	Bits	Description	Type	Reset
CP0EN	1 (0)	Coprocessor #0 enable bit. Value | Meaning 0 | Coprocessor #0 is disabled. Any encountering of Coprocessor #0 instruction when Coprocessor #0 exists will cause "Coprocessor disabled" exception. 1 | Coprocessor #0 is enabled.	RW	0
CP1EN	1 (1)	Coprocessor #1 enable bit. Value | Meaning 0 | Coprocessor #1 is disabled. Any encountering of coprocessor #1 instruction	RW	0

Field Name	Bits	Description	Type	Reset
		(continued) 0: when coprocessor #1 exists will cause "Coprocessor disabled" exception. 1: Coprocessor #1 is enabled.		
CP2EN	1 (2)	Coprocessor #2 enable bit. Value 0: Coprocessor #2 is disabled. Any encountering of coprocessor #2 instruction when coprocessor #2 exists will cause "Coprocessor disabled" exception. Value 1: Coprocessor #2 is enabled.	RW	0
CP3EN	1 (3)	Coprocessor #3 enable bit. Value 0: Coprocessor #3 is disabled. Any encountering of coprocessor #3 instruction when coprocessor #3 exists will cause "Coprocessor disabled" exception. Value 1: Coprocessor #3 is enabled.	RW	0
Reserved	27 (30,4)	RAZWI	-	-

Nested tables in Description:

Value	Meaning
	when coprocessor #1 exists will cause "Coprocessor disabled" exception.
1	Coprocessor #1 is enabled.

Coprocessor #2 enable bit.

Value	Meaning
0	Coprocessor #2 is disabled. Any encountering of coprocessor #2 instruction when coprocessor #2 exists will cause "Coprocessor disabled" exception.
1	Coprocessor #2 is enabled.

Coprocessor #3 enable bit.

Value	Meaning
0	Coprocessor #3 is disabled. Any encountering of coprocessor #3 instruction when coprocessor #3 exists will cause "Coprocessor disabled" exception.
1	Coprocessor #3 is enabled.

Field Name	Bits	Description	Type	Reset
AUEN	1 (31)	Audio extension enable bit. **Value** / **Meaning** 0 — Audio extension is disabled. Any encountering of Audio extension instruction when Audio extension exists will cause "Audio extension disabled" exception. 1 — Audio extension is enabled.	RW	0

5. Instruction Latency for Andes FPU Implementations

This chapter should be in each Andes FPU implementation guide. It is here for easy reference.

5.1. FPU Implementation for N13/N12/N10/D10

This section describes the N13/N12/N10/D10 FPU instruction latency between a producer instruction and a corresponding consumer instruction. This information is useful for compiler optimization of instruction scheduling.

Terminology

- Producer: an instruction that produces a new register state.
- Consumer: an instruction that consumes the new register state produced by a producer.
- Latency: the minimum number of cycles between the completion of a producer and that of a consumer. Assuming a producer and a corresponding consumer cannot complete at the same time, the smallest possible latency is 1.
- Bubble: the minimum number of extra cycles that exceeds the smallest possible latency (i.e. 1) between the completion of a producer and that of a consumer. Thus it is equal to (latency – 1).
- FPU Pipelined latency (PL): the minimum number of cycles between the completion of a producer and that of a consumer that is caused by successive FPU pipeline stages. Inserting a minimum number of (PL-1) independent FPU instructions between the producer and the consumer will make the use of FPU pipeline resources more efficient.
- FPU Self-stalled bubble (SBB): the fixed number of extra cycles that will cause FPU pipeline stall by a producer. Inserting a minimum number of SBB independent integer instructions between the producer and the consumer will make the use of pipeline resources more efficient.
- Observed latency (= PL + SBB): the total observed minimum number of cycles between the completion of a producer and that of a consumer.

Table 47. N13/N12/N10/D10 FPU Instruction Latency Table

No	Producer	FPU Consumer	FPU Pipelined Latency	FPU Self-stalled Bubble	Observed Latency
1	FADDx, FSUBx, Fx2y,	FSR/FDR	5	0	5
2	FMULS	FSR	5	0	5
3	FMULD	FDR	5	1	6
4	FDIVS, FSQRTS	FSR	5	14	19
5	FDIVD, FSQRTD	FDR	5	28	33
6	FCMP*x, FABSx, FCPY*x, FCMOV*x	FSR/FDR	2	0	2
7	FMTSR, FLS (D$ hit)	FSR	5	0	5
8	FMTDR, FLD (D$ hit)	FDR	5	1	6
9	FMFSR	GPR_E1	2	1	3
		GPR_E2	1	1	2
10	FMFDR	GPR_E1 (even # reg)	1	2	3
		GPR_E1 (odd # reg)	2	2	4
		GPR_E2 (all reg)	1	2	3

5.1.1. Instruction Sequence Penalty Cycles for N10/D10

The following table lists the penalty (bubble) cycles between a producer instruction (F1) that has a pipelined latency of 5 cycles and its consumer instruction (F2) depending on various number of data-independent integer (i) and floating-point (f) instructions inserted between F1 and F2.

F1 can be the following instructions,

No.	F1 (Producer)	Register Produced.
1	FADDx, FSUBx, Fx2y,	FSR or FDR
2	FMULS	FSR
3	FMULD	FDR
4	FDIVS, FSQRTS	FSR
5	FDIVD, FSQRTD	FDR

Table 48. Cycle Penalty between Dependent Floating-point Instructions

Penalty (bubble) cycles	Instruction Sequenes
4	{F1, F2}
3	{F1, i, F2}, {F1, f, F2}
2	{F1, i, i, F2}, {F1, f, f, F2}, {F1, i, f, F2}, {F1, f, i, F2}
1	{F1, i, i, i, F2}, {F1, f, f, f, F2}, {F1, f, f, i, F2}, {F1, f, i, f, F2}, {F1, i, f, f, F2}, {F1, i, i, f, F2},

Penalty (bubble) cycles	Instruction Sequenes
	{F1, i, f, i, F2}, {F1, f, i, i, F2}

5.2. FPU Implementation for N15/D15

This section describes the N15/D15 FPU instruction latency between a producer instruction and a corresponding consumer instruction. This information is useful for compiler optimization of instruction scheduling.

Table 49. N15/D15 FPU Instruction Latency Table

No	Producer	FPU Consumer	Data ready Stage	FPU Pipelined Latency	FPU Self-stalled Bubble	Observed Latency
1	FADDx, FSUBx, Fx2y,	FSR/FDR	RF	4	0	4
2	FMULS	FSR	RF	4	0	4
3	FMULD	FDR	RF	4	1	5
4	F(N)MADDS, F(N)MSUBS	FSR	RF	4	2	6
5	F(N)MADDD, F(N)MSUBD	FDR	RF	4	3	7
6	FDIVS, FSQRTS	FSR	RF	4	14	18
7	FDIVD, FSQRTD	FDR	RF	4	28	32
8	FCMP*x, FABSx, FCPY*x, FCMOV*x	FSR/FDR	F2	2	0	2
9	FMTSR, FMTDR	FSR/FDR	F1	1	0	1
10	FLS, FLD (D$ hit)	FSR/FDR	F3	3	0	3

No	Producer	FPU Consumer	Data ready Stage	FPU Pipelined Latency	FPU Self-stalled Bubble	Observed Latency
11	FMTCSR	FPCSR	RF	4	0	4
12	FMFCSR	GPR	EX	1	0	1
13	FMFSR, FMFDR	GPR	EX	1	0	1

6. IEEE Standard Compliance

A subset of IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Standard 754-1985) is provided by the Andes floating-point instruction extension. This section describes how to conform to the Standard using these instructions.

6.1. Choices for IEEE Options

6.1.1. Supported Format

Andes FPU architecture supports IEEE *single-precision* and *double-precision* formats. When FPCFG.SP is set the *single-presision* format is supported in the implementation. When FPCFG.DP is set the *double-precision* format is supported in the implementation. The current FPU architecture mandates that when the *double* format is supported, the *single* format should be supported as well.

6.1.2. Underflow Detection

Underflow exception is generated based on two correlated events: *tininess* and *loss of accuracy*. In Andes FPU architecture, *tininess* is detected "*after rounding*" and *loss of accuracy* is detected as "*an inexact result*".

6.1.3. Delivery of Comparison Result

The result of a comparison is delivered as a true-false response to a predicate named in the instruction mnemonic.

Andes FPU architecture provides hardware support for the IEEE Standard required six predicates (=,≠,<,≦,≧,>) and the optional un-ordered predicate. The other 19 optional predicates can be constructed from sequences of two comparisons and two branches.

Two flavors of compare instructions are provided to assist in different IVO exception generation behaviors specified in Table 4 of the IEEE 754 standard specification.

7. Floating point instruction encoding

opc_6 Encoding

opcode		bit 27-25							
bit 30-28		0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
0	000	LBI	LHI	LWI	LDI	LBI.bi	LHI.bi	LWI.bi	LDI.bi
1	001	SBI	SHI	SWI	SDI	SBI.bi	SHI.bi	SWI.bi	SDI.bi
2	010	LBSI	LHSI	LWSI	DPREFI	LBSI.bi	LHSI.bi	LWSI.bi	
3	011	**LWC/0**	**SWC/0**	**LDC/0**	**SDC/0**	**MEM**	**LSMW**		
4	100	**ALU_1**	**ALU_2**	MOVI	SETHI	**JI**	**JREG**	**BR1**	**BR2**
5	101	ADDI	SUBRI	ANDI	XORI	ORI		SLTI	SLTSI
6	110	**AEXT**	**CEXT**	**MISC**			**COP/0**		
7	111								

COP/0 (Coprocessor #0 Space: b[5:4] = 0b00)

opcode		**Bit 1-0**			
Bit 3-2		0	1	2	3
		00	01	10	11
0	00	**FS1**	**MFCP**	**FLS(.bi)**	**FLD(.bi)**
1	01	**FS2**	*	*	*
2	10	**FD1**	**MTCP**	**FSS(.bi)**	**FSD(.bi)**
3	11	**FD2**	*	*	*

* These six reserved fields are checked by the main processor.

FS1

opcode		Bit 7-6			
Bit 9-8		0	1	2	3
		00	01	10	11
0	00	FADDS	FSUBS	FCPYNSS	FCPYSS
1	01	FMADDS	FMSUBS	FCMOVNS	FCMOVZS
2	10	FNMADDS	FNMSUBS		
3	11	FMULS	FDIVS		**F2OP**

FS1/F2OP

opcode		Bit 11-10		
Bit 14-12		0	1	2-3
		00	01	1x
0	000	FS2D	FSQRTS	
1	001		FABSS	
2	010	FUI2S		
3	011	FSI2S		
4	100	FS2UI		
5	101	FS2UI.z		
6	110	FS2SI		
7	111	FS2SI.z		

FS2

opcode		Bit 6	
Bit 9-7		0	1
		0	1
0	000	FCMPEQS	FCMPEQS.e
1	001	FCMPLTS	FCMPLTS.e
2	010	FCMPLES	FCMPLES.e
3	011	FCMPUNS	FCMPUNS.e
4-7	1xx		

FD1

opcode		Bit 7-6			
Bit 9-8		0	1	2	3
		00	01	10	11
0	00	FADDD	FSUBD	FCPYNSD	FCPYSD
1	01	FMADDD	FMSUBD	FCMOVND	FCMOVZD
2	10	FNMADDD	FNMSUBD		
3	11	FMULD	FDIVD		**F2OP**

FD1/F2OP

opcode		Bit 11-10		
Bit 14-12		0	1	2-3
		00	01	1x
0	000	FD2S	FSQRTd	
1	001		FABSD	
2	010	FUI2D		
3	011	FSI2D		
4	100	FD2UI		
5	101	FD2UI.z		
6	110	FD2SI		
7	111	FD2SI.z		

FD2

opcode		Bit 6	
Bit 9-7		0	1
		0	1
0	000	FCMPEQD	FCMPEQD.e
1	001	FCMPLTD	FCMPLTD.e
2	010	FCMPLED	FCMPLED.e
3	011	FCMPUND	FCMPUND.e
4-7	1xx		

MFCP/MTCP

opcode		Bit 7-6			
Bit 9-8		0	1	2	3
		00	01	10	11
0	00	FMFSR / FMTSR	FMFDR / FMTDR	*	
1	01			*	
2	10			*	
3	11	**XR**		*	

* These four reserved fields are checked by the main processor.

XR

Bit 14-10		**Group**
0	00000	FMFCFG
1	00001	FMFCSR / FMTCSR
2-31	00010 - 11111	

FLS/FSS/FLD/FSD

Bit 7-6		**Instruction**
0	00	FLS / FSS / FLD / FSD
1	01	*
2	10	FLS.bi / FSS.bi / FLD.bi / FSD.bi
3	11	*

* These two reserved fields are checked by the main processor.

LWC/0, SWC/0, LDC/0, SDC/0 (Coprocessor #0 Space: b[14:13] = 0b00)

Bit 12	Instruction
0	FLSI / FSSI / FLDI / FSDI
1	FLSI.bi / FSSI.bi / FLDI.bi / FSDI.bi

Note: a coprocessor implementation should check all fields/encodings it does not use for reserved instruction exception even if some aforementioned fields are checked by the main processor. This is because a coprocessor design implemented earlier may connect with a main processor based on a newer version of coprocessor ISA architecture that will define new instructions from these reserved fields. If a coprocessor implementation does not check these reserved fields, then the newly-defined coprocessor instructions which were not implemented in the coprocessor earlier will pass the main processor and the coprocessor pipeline without generating any reserved exception.

🟦	Instructions present if SP or DP is implemented
🟧	Instructions present if both SP and DP are implemented
🟩	Instructions present if SP is implemented
🟨	Instructions present is DP is implemented

SP extension instruction	🟦🟩
DP extension	🟦🟨
SP and DP extension	🟦🟧🟩🟨