OpenPOWER ISA Compliance Definition

Workgroup Specification

Version 2.0 (April 7, 2020)

www.openpowerfoundation.org

OpenPOWER ISA Compliance Definition:

Compliance Work Group <compliance-chair@openpowerfoundation.org>
OpenPower Foundation

Version 2.0 (April 7, 2020)

Abstract

The purpose of the POWER ISA - OpenPOWER Profile Compliance Test Harness and Test Suite (TH/TS) Specification, Revision 2.0 is to provide the test suite requirements to be able to demonstrate OpenPOWER ISA Profile, Revision 2.0 compliance for POWER9 systems. The input to this specification is the IBM POWER ISA Version 3.0 B. Implementation of all of the ISA is required. There are no optional sections.

This document is a Standard Track, Work Group Specification work product owned by the Compliance Workgroup and handled in compliance with the requirements outlined in the *OpenPOWER Foundation Work Group (WG) Process* document. It was created using the *Master Template Guide* version 1.0.0. Comments, questions, etc. can be submitted to the public mailing list for this document at <openpower-isa-thts@mailinglist.openpowerfoundation.org>.

Table of Contents

Preface

1. Conventions

The OpenPOWER Foundation documentation uses several typesetting conventions.

Notices

Notices take these forms:

Note

A handy tip or reminder.

Important

Something you must be aware of before proceeding.

Warning

Critical information about the risk of data loss or security issues.

Changes

At certain points in the document lifecycle, knowing what changed in a document is important. In these situations, the following conventions will used.

- *New text will appear like this.* Text marked in this way is completely new.
- ~~Deleted text will appear like this.~~ Text marked in this way was removed from the previous version and will not appear in the final, published document.
- Changed text will appear like this. Text marked in this way appeared in previous versions but has been modified.

Command prompts

In general, examples use commands from the Linux operating system. Many of these are also common with Mac OS, but may differ greatly from the Windows operating system equivalents.

For the Linux-based commands referenced, the following conventions will be followed:

$ prompt	Any user, including the `root` user, can run commands that are prefixed with the $ prompt.
# prompt	The `root` user must run commands that are prefixed with the # prompt. You can also prefix these commands with the **sudo** command, if available, to run them.

Document links

Document links frequently appear throughout the documents. Generally, these links include a text for the link, followed by a page number in parenthesis. For example, this link, Preface [vi], references the Preface chapter on page vi.

2. Document change history

This version of the guide replaces and obsoletes all earlier versions.

The following table describes the most recent changes:

Revision Date	Summary of Changes
April 7, 2020	• Version 2.0 - PRD : OpenPOWER ISA Compliance Definition - Approved Specification
November 6, 2019	• Version 2.0 - PRD : OpenPOWER ISA Compliance Definition - Workgroup Approved Public Review Draft
October 29, 2019	• Version 2.0 - pre 11 : Updates for changes identified at the October 29, 2019, Compliance Work Group meeting
October 24, 2019	• Version 2.0 - pre 10 : Updates for changes identified from peer review (updates to section 18.2. Radix Tree Translation)
October 17, 2019	• Version 2.0 - pre 9 : Updates for changes identified from peer review (change doubleword[0], doubleword[2], and doubleword[3] to be word[0], word[2], word[3] respectively for I-7.4.1.VE0.37 xscvqpswz test results and for I-7.4.1.VE0.39 xscvqpuwz test results)
October 11, 2019	• Version 2.0 - pre 8 : Updates for changes identified at the October 2, 2019, Compliance Work Group meeting and minor changes in part 1
September 6, 2019	• Version 2.0 - pre 7 : Updates for changes identified at the September 3, 2019, Compliance Work Group meeting, fixes to sections 19.2.19.1 and 19.2.19.2, and changes to 19.2.20 through the end of the document for POWER9 systems (ISA 3.0B)
August 30, 2019	• Version 2.0 - pre 6 : Fixes to sections 9.4.2, 12, and 19.2.5.1 through 19.2.19.2
August 26, 2019	• Version 2.0 - pre 5 : Updated for POWER9 systems (ISA 3.0B) through end of Part III Section 19.2.19.2
July 26, 2019	• Version 2.0 - pre 4 : Updated for POWER9 systems (ISA 3.0B) through end of Part III Section 18 and for changes from July 11, 2019 Compliance Work Group meeting
June 27, 2019	• Version 2.0 - pre 3 : Updated for POWER9 systems (ISA 3.0B) through end of Part III 18.6 Storage Protection section
May 28, 2019	• Version 2.0 - pre 2 : Updated for POWER9 systems (ISA 3.0B) through end of Part II and for changes identified at the May 23, 2019 Compliance Work Group meeting
May 13, 2019	• Version 2.0 - pre 1 : Updated for POWER9 systems (ISA 3.0B) through end of Part I

1. Introduction

The purpose of the Power ISA - OpenPOWER Profile Compliance Test Harness and Test Suite (TH/TS) Specification, Version 2.0 is to provide the test suite requirements to be able to demonstrate OpenPOWER ISA Profile compliance for POWER9 systems. It contains the following:

- Section describing the test harness needed to execute the test suite
- Section describing the tests required to be in the test suite
- Section describing the successful execution of the test suite, including what it means for an optional feature to fail

The input to this specification is the following:

1. IBM Power ISA™ Version 3.0 B
2. OpenPOWER Specification: Power Instruction Set Architecture (ISA) – OpenPOWER Profile, Revision 2.0 which requires all of the IBM Power ISA Version 3.0B to be implemented.

Note

Power ISA Version 3.0 B chose a single Book III and a set of widely used categories to become part of the base architecture for all forward-looking Power implementations. All other optional architecture categories have been eliminated to ensure increased application portability between Power processors.

The testing of a processor implementation's compliance against the *Power ISA - OpenPOWER Profile* is to ensure that software shown to execute properly on one compliant processor implementation will execute properly on a different also compliant processor implementation.

The testing is not intended to show that the processor implementation under test is robust under all possible operating conditions, inputs, or event time interactions. It is intended to show that the processor implementation under test implemented the ISA as specified and the specification was interpreted by the processor developers as intended by the specification authors.

1.1. Conformance to this Specification

The following lists a set of numbered conformance clauses to which any implementation of this specification must adhere in order to claim conformance to this specification (or any optional portion thereof):

1. The required tests in the Power ISA - OpenPOWER Profile Test Suite Required Tests Section must be successfully executed.
2. For optional facilities that are implemented, the optional tests in the Power ISA - OpenPOWER Profile Test Suite Optional Tests Section must be successfully executed.

Note

There are no optional sections.

2. Power ISA - OpenPOWER Profile Test Harness and Test Suite

The goals and methodology of architectural compliance testing are fundamentally different from those of design verification. Architectural compliance testing checks that the architecture specification has been correctly interpreted in the design. It does not check that the implementation works correctly in every possible case, and does not intend to find bugs related to a specific implementation. Compliance testing focuses on behaviors defined in the architecture specification, and on potential misinterpretations of these definitions. Therefore, the methodology for compliance testing is based on directed tests, each checking a specific architecture behavior. The tests are intended to demonstrate correct interpretation of the behavior, and to expose incorrect interpretations. In contrast to design verification, these tests are not randomized and are not executed multiple times (except for timing randomization in the case of multiprocessing tests). Compliance testing is not based on high coverage of a random space, but on systematic testing of predefined behaviors.

The methodology for architectural compliance testing described in this document is based on *scenarios*. Each scenario describes a set of tests that should be performed, and successful execution of all of these tests is necessary for complete compliance testing. There are two general categories of scenarios: *instruction-driven scenarios* and *mechanism-driven scenarios*.

Instruction-driven scenarios are based on definitions of instruction behavior in the architecture specification. Each scenario deals with an aspect of the instruction behavior, such as setting a specific register field. For each instruction, executing all of its related scenarios is necessary for fully testing the interpretation of the instruction description.

Mechanism-driven scenarios are based on definitions of mechanisms in the architecture specification, which may involve interactions between several instructions or architectural resources. Each scenario describes a different sequence of events related to the mechanism, and executing all of the related scenarios is necessary for complete checking of the mechanism interpretation.

Checking a particular feature may require several types of scenarios. *Straightforward* scenarios check that the required action is indeed taken. *No change* scenarios check that unnecessary actions are not taken. *Misconception* scenarios are aimed at exposing potential misconceptions, such as confusion between similar instructions. In a few cases, *boundary* scenarios check values that are near a threshold, where a boundary between different behaviors should be maintained.

The structure of this document follows the structure of the architecture specification document. Part I, “Scenarios for Compliance Testing - User Instruction Set (Book I) and Related Supervisor Instructions (Book III)” [5] is devoted to architecture Book I, and mainly includes instruction-driven scenarios. Part II, “Scenarios for Compliance Testing - Virtual Environment (Book II) and Related Supervisor Instructions (Book III)” [167] and Part III, “Scenarios for Compliance Testing - Operating Environment (Book III)” [192] are devoted to Books II and III, respectively, and mainly include mechanism-driven scenarios. Within each section of this document, subsections correspond to chapters in the architecture books.

Generally, this document uses the notation and terminology of the architecture specification. Any terms used in scenario descriptions are either defined in the notes preceding the scenario, or can be found in the relevant section in the architecture specification. Instruction-driven scenarios use symbols and expressions taken from the RTL descriptions of the relevant instructions.

Note

This compliance document is based on POWER9 systems and the Instruction Set Architecture 3.0 B.

2.1. General guidelines

- Redundant tests: tests for checking different instructions and mechanisms are described separately, for clarity. However, it is possible to use the same test for checking several scenarios, if it conforms with their descriptions.

- Longer tests: tests are described as executing a single scenario (a single instruction or specified sequence of instructions). It is possible to create longer tests, executing several scenarios in sequence, provided that the **Observability preconditions** and **Compliance conditions** hold for each scenario in the sequence.

- Observability preconditions: these are necessary preconditions in order to ensure that incorrect behavior will be observed, if it occurs during test execution. In some scenarios (mainly mechanism-driven scenarios), **Observability preconditions** are explicitly defined. In most scenarios, **Observability preconditions** are not explicitly defined, and should be added in a standard way to every test: for every requirement that appears in the **Expected results**, there should be a corresponding precondition for the same resource, with an initial value that differs from the expected value. For example, if the **Expected results** require that FR=1, the corresponding observability precondition is that FR=0.

- Multiple conditions: the **Compliance conditions, Observability conditions**, or **Expected results** of a scenario may include more than one condition. In this case, the meaning is the conjunction (AND) of all conditions.

2.2. Instruction-driven scenarios

Instruction-driven scenarios define tests that execute a single instruction.

Each scenario description includes:

- **Instructions tested**
- **Compliance conditions**
- **Observability preconditions**(explicit or implicit)
- **Expected results**

The scenario defines a set of tests. For each instruction in the **Instructions tested** list, the following test should be performed:

- Set initial values according to the **Observability preconditions**
- Execute the instruction, with operands and settings that cause the **Compliance conditions** to occur
- Check that the **Expected results** occur after executing the instruction

In some scenarios, the **Instructions Tested** are specified as Representative of... <*a set of instructions*>. In this case, the scenario can be executed for a single representative instruction, selected randomly from the given set

2.3. Mechanism-driven scenarios

Mechanism-driven scenarios require execution of a sequence of one or more instructions.

Each scenario description includes:

- **Instruction sequence**
- **Compliance conditions**
- **Observability preconditions**(explicit or implicit)
- **Expected results**

In multiprocessing scenarios, the **Instruction sequence** may define sequences for several processes. No order is implied between instructions of different processes.

The scenario defines a single test, to be performed for representative instructions selected from the relevant group of tested instructions, as follows:

- Set initial values according to the **Observability preconditions**
- Execute the **Instruction sequence**, with operands and settings that cause the **Compliance conditions** to occur
- Check that the **Expected results** occur after executing the instruction sequence

For multiprocessing scenarios, each test should be executed multiple times, with random timing, so that instruction interleavings between the different processes have been covered.

Part I. Scenarios for Compliance Testing - User Instruction Set (Book I) and Related Supervisor Instructions (Book III)

This Part describes the scenarios required for compliance testing the User Instruction Set (Book I) and related Supervisor Instructions (Book III). The methodology and guidelines specified in Chapter 2, “Power ISA - OpenPOWER Profile Test Harness and Test Suite” [2] apply to all of the scenarios in this Part.

3. Branch Facility (Chapter I.2)

Table of Contents

3.1. Branch Instructions

Architecture sections:

- I-2.4 Branch Instructions

Scenario groups:

- Condition calculation
- Setting the Link Register (LR)
- Target address computing
- CTR setting and BO_2 value

3.1.1. Condition Calculation

Branch conditional and branch conditional to LR/TAR instructions: bc bca bcl bcla bclr bclrl bctar bctarl

Branch conditional to CTR instructions: bcctr bcctrl

Branch unconditional instructions: b ba bl bla

Guideline:

In the following test, each instruction should be tested in both 32-bit and 64-bit computation modes taking into account the tested bits of the Count Register (CTR) in each mode.

Notation:

In the table, M=0 in 64-bit mode and M=32 in 32-bit mode

Instructions tested	Compliance conditions	Observability preconditions	Expected result
I-2.4.Cond.1 The BO field and a tested bit in the Condition Register (CR) resolves the branch TAKEN case			
Branch conditional to CTR instructions	$(BO)_0 = 1$ OR $(CR)_{(BI)+32} = (BO)_1$ $(BO)_2 \neq 0$		Branch is Taken
I-2.4.Cond.2 The BO field and a tested bit in the Condition Register (CR) resolves the branch NOT-TAKEN case			
Branch conditional to CTR instructions	$(BO)_0 \neq 1$ AND $(CR)_{(BI)+32} \neq (BO)_1$ $(BO)_2 \neq 0$		Branch is Not-Taken
I-2.4.Cond.3 The BO field , a tested bit in the Condition Register (CR) and the decremented value of the Count Register (CTR) are used to resolve the branch TAKEN case			

Instructions tested	Compliance conditions	Observability preconditions	Expected result
Branch conditional to LK/TAR instructions	$[(BO)_0 = 1$ OR $(CR)_{(BI)+32} = (BO)_1]$ $[(BO)_2 = 1$ OR $((CTR)_{M:63} \neq 0 \oplus (BO)_3)]$		Branch is Taken
I-2.4.Cond.4 The BO field , a tested bit in the Condition Register (CR) and the decremented value of the Count Register (CTR) are used to resolve the branch NOT-TAKEN case			
Branch conditional to LK/TAR instructions	$[(BO)_0 \neq 1$ AND $(CR)_{(BI)+32} \neq (BO)_1]$ OR $[(BO)_2 \neq 1$ AND $(((CTR)_{M:63} = 0) \oplus (BO)_3)]$		Branch is Not- Taken
I-2.4.Cond.5 Branch is always taken with *unconditional instructions*			
Branch unconditional instructions			Branch is Taken

Note

See Section 17.5, “Move To/From System Register Instructions” [204] for setting the TAR and LR using the Move To System Register instructions.

3.1.2. Setting the Link Register (LR)

Branch setting the LR instructions: bl bla bcl bcla bclrl bcctrl bctarl

Branch not-setting the LR instructions: b ba bc bca bclr bcctr bctar

Notation:

CIA current instruction address

Instructions tested	Compliance conditions	Observability preconditions	Expected result
I-2.4.LR.1 The LR holds the return address after Branch instructions for the relevant instructions			
Branch setting the LR instructions	LK = 1	(LR) != CIA + 4	(LR) = CIA + 4
I-2.4.LR.2 Non setting the LR instructions doesnt set the LR			
Branch not-setting the LR instructions	LK = 0		LR is unchanged

3.1.3. Target address computing

Notation:

NIA next instruction address

Instructions tested	Compliance conditions	Observability preconditions	Expected result
I-2.4.NIA.1 Adding a displacement to the address of the unconditional Branch instruction in 64-bit mode			
b bl			NIA = CIA + sign_extension(LI || 0b00)
I-2.4.NIA.2 Adding a displacement to the address of the unconditional Branch instruction in 32-bit mode			
b bl	32-bit computation mode		$NIA_{0:31}$ = CIA + sign_extension(LI || 0b00)

Instructions tested	Compliance conditions	Observability preconditions	Expected result
	AA = 0		$NIA_{32:63} = {}^{32}0$
I-2.4.NIA.3 Specifying the absolute address (AA) of the unconditional branch instruction in 64-bit mode			
ba bla	64-bit computation mode AA = 1		NIA = sign_extension(LI || 0b00)
I-2.4.NIA.4 Specifying the absolute address (AA) of the unconditional branch instruction in 32-bit mode			
ba bla	32-bit computation mode AA = 1		$NIA_{0:31}$ = sign_extension(LI || 0b00) $NIA_{32:63} = {}^{32}0$
I-2.4.NIA.5 Adding a displacement to the address of the conditional branch instruction when branch is **taken** in 64-bit mode			
bc bcl	64-bit computation mode AA = 0		NIA = CIA + sign_extension(BD || 0b00)
I-2.4.NIA.6 Adding a displacement to the address of the conditional branch instruction when branch is **taken** in 32-bit mode			
bc bcl	32-bit computation mode AA = 0		$NIA_{0:31}$ = CIA + sign_extension(BD || 0b00) $NIA_{32:63} = {}^{32}0$
I-2.4.NIA.7 Specifying the absolute address (AA) of the conditional branch instruction when branch is **taken** in 64-bit mode			
bca bcla	64-bit computation mode AA = 1		NIA = sign_extension(BD || 0b00)
I-2.4.NIA.8 Specifying the absolute address (AA) of the conditional branch instruction when branch is **taken** in 32-bit mode			
bca bcla	32-bit computation mode AA = 1		$NIA_{0:31}$ = sign_extension(BD || 0b00) $NIA_{32:63} = {}^{32}0$
I-2.4.NIA.9 Using the address contained in the Link Register when the branch is **taken** in 64-bit mode			
bclr bclrl	64-bit computation mode		NIA = $LR_{0:61}$ || 0b00
I-2.4.NIA.10 Using the address contained in the Link Register when the branch is **taken** in 32-bit mode			
bclr bclrl	32-bit computation mode		NIA = $LR_{0:31} \|\|^{32}0$
I-2.4.NIA.11 Using the address contained in the Count Register when the branch is **taken** in 64-bit mode			
bcctr bcctrl	64-bit computation mode		NIA = $CTR_{0:61}$ || 0b00
I-2.4.NIA.12 Using the address contained in the Count Register when the branch is **taken** in 32-bit mode			
bcctr bcctrl	32-bit computation mode		NIA = $CTR_{0:31} \|\|^{32}0$
I-2.4.NIA.13 Using the address contained in the Target Address Register when the branch is **taken** in 64-bit mode			
bctar bctarl	64-bit computation mode		NIA = $TAR_{0:61}$ || 0b00
I-2.4.NIA.14 Using the address contained in the Target Address Register when the branch is **taken** in 32-bit mode			
bctar bctarl	32-bit computation mode		NIA = $TAR_{0:31} \|\|^{32}0$
I-2.4.NIA.15 The next instruction address of conditional branch instructions when the branch is **not taken**			
bc bcl bca bcla bclr bclrl bcctr bcctrl bctar bctarl			NIA = CIA + 4

3.1.4. CTR setting and BO_2 value

Branch conditional instructions:

Branch conditional and branch conditional to LR/TAR instructions: bc bca bcl bcla bclr bclrl bctar bctarl

Branch conditional to CTR instructions: bcctr bcctrl

Instructions tested	Compliance conditions	Observability preconditions	Expected result
I-2.4.BO2.1 The Count Register is decremented by one when the BO bit 2 equals zero in branch conditional not to CTR instructions			
Branch conditional and branch conditional to LR/TAR instructions	$BO_2 = 0$		CTR = CTR - 1
I-2.4.BO2.2 If the BO bit-2 equals zero in *branch conditional to CTR instructions* then the instruction is invalid			
Branch conditional to CTR instructions	$BO_2 = 0$		Instruction form is invalid
I-2.4.BO2.3 BO bit 2 equals 1 doesnt change the CTR or affect the form validity			
Branch conditional instructions	$BO_2 = 1$		Instruction form valid and CTR unchanged

3.2. Condition Register Instructions

Architecture sections:

- I-2.5 Condition Register Instructions

Scenario groups:

- Setting a specified bit in the CR
- Moving a CR field

3.2.1. Setting a specified bit in the CR

Condition Register Logical Instruction: crand crnand cror crxor crnor creqv crandc crorc

Instructions tested	Compliance conditions	Observability preconditions	Expected result
I-2.5.1.CRBT.1 Setting the bit in the CR specified by BT+32 according to the instructions logical operation between the bits in the CR specified by BA+32 and BB+32			
Condition Register Logical Instruction		CR_{BT+32} != result of the instructions logical operation applied to $CR_{BA+32}$and CR_{BB+32}	CR_{BT+32} = result of the instructions logical operation applied to $CR_{BA+32}$and CR_{BB+32}

3.2.2. Moving a CR field

Instructions tested	Compliance conditions	Observability preconditions	Expected result
I-2.5.2.CRBF.1 Copying the contents of CR field BFA to the CR field BF			
mcrf		$CR_{4BF+32 : 4BF+35}$!= $CR_{4BFA+32 : 4BFA+35}$	$CR_{4BF+32 : 4BF+35}$ = $CR_{4BFA+32 : 4BFA+35}$

3.3. System Call Instructions

Architecture sections:

- I-2.6 System Call Instructions
- III-3.3 Branch Facility Instructions

Scenario groups:

- Fetching the next instruction

- Setting SRR0
- Setting SRR1

3.3.1. Fetching the next instruction

Instructions tested	Compliance conditions	Observability preconditions	Expected result
I-2.6.NIA.1 The interrupt causes the next instruction to be fetched from the right effective address			
sc scv			NIA = 0X0000_0000_0000_0C00

3.3.2. Setting SRR0

Instructions tested	Compliance conditions	Observability preconditions	Expected result
I-2.6.SRR0.1 Setting the value of SRR0 correctly to the next instruction to be executed when the control returns to the program			
sc scv		SRR0 != CIA+4	SRR0 = CIA+4

3.3.3. Setting SRR1

Instructions tested	Compliance conditions	Observability preconditions	Expected result
I-2.6.SRR1.1 Setting the SRR1 according to the MSR			
sc scv		$SRR1_{33:36\ 42:47}$!= 0 $SRR1_{0:32\ 37:41\ 48:63}$!= $MSR_{0:32\ 37:41\ 48:63}$	$SRR1_{33:36\ 42:47}$ = 0 $SRR1_{0:32\ 37:41\ 48:63}$ = $MSR_{0:32\ 37:41\ 48:63}$

3.4. Branch History Rolling Buffer Instructions

Architecture sections:

- I-8.2 Branch History Rolling Buffer Instructions

Scenario groups:

- Clearing the BHRB
- Moving a BHRBE

3.4.1. Clearing the BHRB

Notation:

The number of BHRB entries are implementation-dependent and can be from 0 through 1023

Instructions tested	Compliance conditions	Observability preconditions	Expected result
I-8.2.CLR.1 Clearing the BHRB entries			
clrbhrb			All BHRB entries are set to 0

3.4.2. Moving a BHRBE

Instructions tested	Compliance conditions	Observability preconditions	Expected result
I-8.2.MV.1 If it is in the range implemented; Moving the contents of the designated BHRB entry into the RT			

Instructions tested	Compliance conditions	Observability preconditions	Expected result
mfbhrbe	The BHRB entry is in the range implemented	RT != (BHRBE)	RT = (BHRBE)
I-8.2.MV.2 If the designated entry is not in the range implemented, then set RT to zero			
mfbhrbe	The BHRBE is not in the implemented range	RT != $^{64}0$	RT = $^{64}0$

4. Fixed Point Facility (Chapter I.3)

Table of Contents

General guidelines

Each scenario in this section should be run in both computation modes: 32-bit and 64-bit.

4.1. Fixed-Point Load and Store Instructions

Architecture sections:

- I-3.3.1 Fixed-Point Storage Access Instructions
- I-3.3.2 Fixed-Point Load Instructions
- I-3.3.3 Fixed-Point Store Instructions
- I-3.3.4 Fixed-Point Load and Store Quadword Instructions
- I-3.3.5 Fixed-Point Load and Store with Byte Reversal Instructions
- I-3.3.6 Fixed-Point Load and Store Multiple Instructions
- I-3.3.7 Fixed-Point Move Assist Instructions [Phased Out]

Scenario groups:

- Correct loading data into the target register
- Validity of Load Instructions
- Validity of Load Move Assist Instructions
- Undefined Load Move Assist Instructions
- Validity of Store Instructions
- Correct Stored data in storage addressed by EA
- Storage Access Exceptions

Guideline: each scenario in this section should be tested in both Big-Endian and Little-Endian mode, with the exception of testing the *Fixed-Point Load and Store Multiple Instructions* and *Fixed-Point Move Assist Instructions*. These instructions can be tested only in Big Endian mode, otherwise a system alignment error handler is invoked.

4.1.1. Correct loading of data into the target register

Architecture sections:

- I-3.3.2 Fixed-Point Load Instructions
- I-3.3.5 Fixed-Point Load and Store with Byte Reversal Instructions
- I-3.3.4 Fixed-Point Load and Store Quadword Instructions

Fixed-Point load instructions: lbz lbzx lbzu lbzux lhz lhzx lhzu lhzux lhbrx lwz lwzx lwzu lwzux lwbrx lha lhax lhau lhaux lwa lwax lwaux ld ldx ldu ldux ldbrx lq

Note: the following scenarios can be tested with valid instructions forms only. (for more information, see 3.1.2.1.2)

Instructions tested	Compliance conditions	Expected result
I-3.3.2.RT.1 Setting the target register correctly with the loaded storage addressed by the effective address		
Fixed-Point load instructions		Correct data is loaded into the target register

4.1.2. Validity of Load Instructions

Architecture sections:

- I-3.3.2 Fixed-Point Load Instructions
- I-3.3.5 Fixed-Point Load and Store with Byte Reversal Instructions
- I-3.3.4 Fixed-Point Load and Store Quadword Instructions
- I-3.3.6 Fixed-Point Load and Store Multiple Instructions

Load Update instructions:

32-bit instructions: lbzu lbzux lhzu lhzux lhau lhaux lwzu lwzux

64-bit instructions: lwaux , ldu , ldux

Load Non-update instructions:

32-bit instructions: lbz lbzx lhz lhzx lha lhax lwz lwzx lhbrx lwbrx

64-bit instructions: lwa lwax ldbrx ld ldx

Instructions tested	Compliance conditions	Expected result
I-3.3.2.Valid.1 Determine the validity when RA = 0 and it is in the range of registers to be loaded		
lmw	RA = 0 and RA = RT	Invalid instruction form
I-3.3.2.Valid.2 Determine the validity when RA = 0 and it is not in the range of registers to be loaded		
lmw	RA = 0 and RA < RT	Valid instruction form
I-3.3.2.Valid.3 Determine the validity when RA != 0 but it is in the range of registers to be loaded		
lmw	RA != 0 and RA >= RT	Invalid instruction form
I-3.3.2.Valid.4 Determine the validity when RA != 0 and it is not in the range of registers to be loaded		
lmw	1 <= RA <= 30 RA < RT	Valid instruction form

Instructions tested	Compliance conditions	Expected result
I-3.3.2.Valid.5 Determine the validity of *Load Non-update instructions*		
Load Non-update instructions		Valid instruction form
I-3.3.2.Valid.6 Determine the validity of *Load Update instructions* when RA = 0		
Load Update instructions	RA = 0	Invalid instruction form
I-3.3.2.Valid.7 Determine the validity of *Load Update instructions* when RA != 0 but RA = RT		
Load Update instructions	RA != 0 RA = RT	Invalid instruction form
I-3.3.2.Valid.8 Determine the validity of *Load Update instructions* when RA != 0 and RA != RT		
Load Update instructions	RA !=0 RA != RT 0 ≤ RT ≤ 31	Valid instruction form
I-3.3.2.Valid.9 Determine the validity of load quadword instruction when RA = RTp or the RTp is odd		
lq	RA = RTp Or RTp is odd	Invalid instruction form
I-3.3.2.Valid.10 Determine the validity of load quadword instruction when RA != RTp and the RTp is even		
lq	RA != RTp RTp is even	Valid instruction form

4.1.3. Validity of Load Move Assist Instructions

Architecture sections:

- I-3.3.7 Fixed-Point Move Assist Instructions [Phased Out]

Instructions tested	Compliance conditions	Expected result
I-3.3.7.Valid.1 Determine the validity when RA = 0 and it is in the range of registers to be loaded		
lswi lswx	RA = 0 RA = RT	Invalid instruction form
I-3.3.7.Valid.2 Determine the validity when RA = 0 and it is not in the range of registers to be loaded		
lswi	RA = 0 RA < RT	Valid instruction form
I-3.3.7.Valid.3 Determine the validity when RA != 0 but it is in the range of registers to be loaded		
lswi	RA != 0 RA ≥ RT	Invalid instruction form
I-3.3.7.Valid.4 Determine the validity when RA != 0 and it is not in the range of registers to be loaded		
lswi	1 =< RA <= 30 RA < RT	Valid instruction form
I-3.3.7.Valid.5 Determine the validity when RA is not in the range of registers to be loaded but RB equals to RT		
lswx	RA < RT RB = RT	Invalid instruction form
I-3.3.7.Valid.6 Determine the validity when RA and RB are not in the range of registers to be loaded		
lswx	RA < RT	Valid instruction form

Instructions tested	Compliance conditions	Expected result
	RB < RT	
I-3.3.7.Valid.7 Determine the validity when RA is not in the range of registers to be loaded but RB is in the range		
lswx	RA < RT RB > RT RB <= RT+n (n number of bytes to be loaded)	Invalid instruction form
I-3.3.7.Valid.8 Determine the validity when RA = 0 and RB is not in the range of registers to be loaded		
lswx	RA = 0 RA < RT RB > RT+n (n number of bytes to be loaded)	Valid instruction form
I-3.3.7.Valid.9 Determine the validity when RA is in the range of registers to be loaded		
lswx	RA > RT (RA !=0) RA <= RT + n	Invalid instruction form
I-3.3.7.Valid.10 Determine the validity when RA and RB are not in the range of registers to be loaded		
lswx	RA > RT (RA !=0) RA+RB > RT + n	Valid instruction form
I-3.3.7.Valid.11 Determine the validity when RB is in the range of registers to be loaded		
lswx	RA > RT (RA !=0) RA > RT + n RB <= RT + n	Invalid instruction form

4.1.4. Undefined Load Move Assist Instructions

Architecture sections:

- I-3.3.7 Fixed-Point Move Assist Instructions [Phased Out]

Instructions tested	Compliance conditions	Expected result
I-3.3.7.define.1 The contents of register RT are undefined when the $XER_{57:63}$ are zeros, no indication of the number of bits to load		
lswx	$XER_{57:63}$ = 0	undefined
I-3.3.7.define.2 The contents of register RT are defined when the $XER_{57:63}$!= 0		
lswx	$XER_{57:63}$!= 0	defined

4.1.5. Validity of Store Instructions

Architecture sections:

- I-3.3.3 Fixed-Point Store Instructions
- I-3.3.4 Fixed-Point Load and Store Quadword Instructions
- I-3.3.5 Fixed-Point Load and Store with Byte Reversal Instructions
- I-3.3.6 Fixed-Point Load and Store Multiple Instructions
- I-3.3.7 Fixed-Point Move Assist Instructions [Phased Out]

Store Update instructions:

32-bit instructions: stbu , stbux , sthu , sthux , stwu , stwux

64-bit instructions: stdu , stdux

Store Non-update instructions:

32-bit instructions: stb , stbx , sth , sthx , stw , stwx, sthbrx , stwbrx , stmw , stswi , stswx

64-bit instructions: std , stdx , stdbrx

Instructions tested	Compliance conditions	Expected result
I-3.3.3.Valid.1 Determine the validity of *Store Non-update instructions*		
Store Non-update instructions		Valid instruction form
I-3.3.3.Valid.2 Determine the validity when RA = 0 of *Store Update instructions*		
Store Update instructions	RA = 0	Invalid instruction form
I-3.3.3.Valid.3 Determine the validity when RA != 0 of *Store Update instructions*		
Store Update instructions	RA != 0	Valid instruction form
I-3.3.3.Valid.4 Determine the validity for store quadword instruction when the RSp is odd		
stq	RSp is odd	Invalid instruction form
I-3.3.3.Valid.5 Determine the validity for store quadword instruction when the RSp is even		
stq	RSp is even	Valid instruction form

4.1.6. Correct Stored data in storage addressed by EA

Architecture sections:

- I-3.3.3 Fixed-Point Store Instructions
- I-3.3.4 Fixed-Point Load and Store Quadword Instructions
- I-3.3.5 Fixed-Point Load and Store with Byte Reversal Instructions

Fixed-Point Store Instructions: stb stbx stbu stbux sth sthx sthu sthux stw stwx stwu stwux std stdx stdu stdux stq sthbrx stwbrx stdbrx

Instructions tested	Compliance conditions	Expected result
I-3.3.3.Store.1 Storing the data correctly in the storage memory addressed by the effective address		
Fixed-Point Store Instructions		Storing the data correctly in the memory storage addressed by the effective address

4.1.7. Storage Access Exceptions

Instructions tested	Compliance conditions	Expected result
I-3.3.3.StorageEx.1 A system data storage error handler will be invoked when an instruction attempts to access unavailable storage		
Fixed-Point load instructions *Fixed-Point Store Instructions*	The program is not allowed to modify the target storage (Store only) or the program attempts to access storage that is unavailable.	system data storage error handler is invoked

4.2. Fixed-Point Arithmetic Instructions

Architecture sections:

- I-3.3.8 Other Fixed-Point Instructions
- I-3.3.9 Fixed-point Arithmetic Instructions

Scenario groups:

- Setting CA (carry) and CA32 (32-bit carry)
- Setting CR0
- Setting OV (overflow) and OV32 (32-bit overflow)
- Setting SO (summary overflow)
- Arithmetic computation

4.2.1. Setting CA (carry) and CA32 (32-bit carry)

Carrying instructions: addic addic. subfic addc[o][.] subfc[o][.] adde[o][.] subfe[o][.] addme[o][.] subfme[o][.] addze[o][.] subfze[o][.]

Non-carrying instructions: addi addis addpcis add[o][.] subf[o][.] addex neg[o][.] mulli mulhw[.] mullw[o][.] mulhwu[.] divw[o][.] divwu[o][.] divwe[o][.] divweu[o][.] modsw moduw darn mulld[o][.] mulhd[.] mulhdu[.] maddhd maddhdu maddld divd[o][.] divdu[o][.] divde[o][.] divdeu[o][.] modsd modud

Note:

Carry out is mode-dependent: carry out of bit 32 for 32-bit mode, carry out of bit 64 for 64-bit mode.

Note:

CA32 is set whenever CA is set, and is set to the same value that CA is defined to be set to in 32-bit mode.

Instructions tested	Compliance conditions	Expected result
I-3.3.9.CA.1 Carrying instructions turn on the CA bit when carry out is 1		
Carrying instructions	Carry out is 1	CA = 1
I-3.3.9.CA.2 Carrying instructions do not turn on the CA bit when carry out is 0		
Carrying instructions	Carry out is 0	CA = 0
I-3.3.9.CA.3 Non-carrying instructions do not change the CA bit		
Non-carrying instructions		CA unchanged

4.2.2. Setting CR0

Recording instructions: addic. addc[o]. subfc[o]. adde[o]. subfe[o]. addme[o]. subfme[o]. addze[o]. subfze[o]. add[o]. subf[o]. neg[o]. mulhw. mullw[o]. mulhwu. divw[o]. divwu[o]. divwe[o]. divweu[o]. mulld[o]. mulhd. mulhdu. divd[o]. divdu[o]. divde[o]. divdeu[o].

Non-recording instructions: addic subfic addc[o] subfc[o] adde[o] subfe[o] addme[o] subfme[o] addze[o] subfze[o] addi addis addpcis add[o] subf[o] addex neg[o] mulli mulhw mullw[o] mulhwu divw[o] divwu[o] divwe[o] divweu[o] modsw moduw darn mulld[o] mulhd mulhdu maddhd maddhdu maddld divd[o] divdu[o] divde[o] divdeu[o] modsd modud

Instructions tested	Compliance conditions	Expected result
I-3.3.9.CR0.1 Recording instructions set CR0 correctly when result is zero		

Instructions tested	Compliance conditions	Expected result
Recording instructions	Result of operation is zero	First three bits of CR0 = 001
I-3.3.9.CR0.2 Recording instructions set CR0 correctly when result is positive		
Recording instructions	Result of operation is positive	First three bits of CR0 = 010
I-3.3.9.CR0.3 Recording instructions set CR0 correctly when result is negative		
Recording instructions	Result of operation is negative	First three bits of CR0 = 100
I-3.3.9.CR0.4 Non-recording instructions do not change CR0		
Non-recording instructions		First three bits of CR0 unchanged

4.2.3. Setting OV (overflow) and OV32 (32-bit overflow)

Add instructions: addo[.] addco[.] addeo[.] addmeo[.] addzeo[.] addex(if CY=0)

Subtract From instructions: subfo[.] subfco[.] subfeo[.] subfmeo[.] subfzeo[.]

Negate instructions: nego[.]

Multiply Low, and Divide:

32-bit instructions: mullwo[.] divwo[.] divwuo[.] divweo[.] divweuo[.]

64-bit instructions: mulldo[.] divdo[.] divduo[.] divdeo[.] divdeuo[.]

Note:

OV32 is set whenever OV is set, and is set to the same value that OV is defined to be set to in 32-bit mode.

Instructions tested	Compliance conditions	Expected result
I-3.3.OV.1 *Add, Subtract From, and Negate instructions* having OE = 1 set OV to 1 if the carry out of bit M is not equal to the carry out of bit M+1		
Add instructions *Subtract From instructions* *Negate instructions*	OE = 1 Carry out of bit M != carry out of bit M+1	OV = 1
I-3.3.OV.2 *Add, Subtract From, and Negate instructions* having OE = 1 set OV to 0 if the carry out of bit M is equal to the carry out of bit M+1		
Add instructions *Subtract From instructions* *Negate instructions*	OE = 1 Carry out of bit M = carry out of bit M+1	OV = 0
I-3.3.OV.3 *Multiply Low, and Divide 32-bit instructions* having OE = 1 set OV to 1 if the result cannot be represented in 32 bits		
Multiply Low, and Divide 32-bit instructions	OE =1 The result cannot be represented in 32 bits	OV = 1
I-3.3.OV.4 *Multiply Low, and Divide 64-bit instructions* having OE = 1 set OV to 1 if the result cannot be represented in 64 bits		
Multiply Low, and Divide 64-bit instructions	OE =1 The result cannot be represented in 64 bits	OV = 1
I-3.3.OV.5 *Multiply Low, and Divide 32-bit instructions* having OE = 1 set OV to 0 if the result can be represented in 32 bits		

Instructions tested	Compliance conditions	Expected result
Multiply Low, and Divide 32-bit instructions	OE =1 The result can be represented in 32 bits	OV = 0
I-3.3.OV.6 *Multiply Low, and Divide 64-bit instructions* having OE = 1 set OV to 0 if the result can be represented in 64 bits		
Multiply Low, and Divide 64-bit instructions	OE =1 The result can be represented in 64 bits	OV = 0

4.2.4. Setting SO (summary overflow)

Overflowing instructions: addo[.] addco[.] subfo[.] subfco[.] addeo[.] subfeo[.] addmeo[.] subfmeo[.] addzeo[.] subfzeo[.] nego[.] mullwo[.] divwo[.] divwuo[.] divweo[.] divweuo[.] mulldo[.] divdo[.] divduo[.] divdeo[.] divdeuo[.]

Note:

Overflow is mode-dependent. For 32-bit mode, overflow is of the low-order 32-bit result. For 64-bit mode, overflow is of the 64-bit result.

Note:

OV32 is set whenever OV is set, and is set to the same value that OV is defined to be set to in 32-bit mode.

Instructions tested	Compliance conditions	Expected result
I-3.3.9.SO.1 Add operations turn on XER_{SO} when overflow occurs		
Overflowing instructions	OV = 1	SO = 1
I-3.3.9.SO.2 Add operations do not turn on XER_{SO} when overflow does not occur		
Overflowing instructions	OV = 0	SO = 0
I-3.3.9.SO.3 Add operations leave XER_{SO} turned on, even if overflow does not occur (SO is sticky)		
Overflowing instructions	OV = 0 SO = 1	SO = 1
I-3.3.9.SO.4 Add operations leave XER_{SO} turned on when overflow occurs		
Overflowing instructions	Overflow occurs SO = 1	SO = 1

4.2.5. Arithmetic computation

Instructions tested	Compliance conditions	Expected result
I-3.3.9.Comp.1 Result of arithmetical computation is correct		
All arithmetic operations		Correct arithmetical result

4.3. Fixed-Point Compare Instructions

Architecture sections:

- I-3.3.10 Fixed-point Compare Instructions

Scenario groups:

- Setting CR BF
- Compare computation

4.3.1. Setting CR BF

Notes:

- The L field controls whether the operands are treated as 64-bit or 32-bit quantities, as follows:

  L = 0: 32-bit operands

  L = 1: 64- bit operands
- XER_{SO} is copied to the 3rd bit of the designated CR field.

Instructions tested	Compliance conditions	Expected result
I-3.3.10.BF.1 The comparing instructions set CR-BF leftmost three bits correctly when result is EQUAL		
cmpi cmp cmpli cmpl	Result of operation is Equal	$(CR\ field\ BF)_{0:2} = 001$ $(CR\ field\ BF)_3 = XER_{SO}$
I-3.3.10.BF.2 The comparing instructions set CR-BF leftmost three bits correctly when result is GT		
cmpi cmp cmpli cmpl	Result of operation is GT (greater than)	$(CR\ field\ BF)_{0:2} = 010$ $(CR\ field\ BF)_3 = XER_{SO}$
I-3.3.10.BF.3 The comparing instructions set CR-BF leftmost three bits correctly when result is LT		
cmpi cmp cmpli cmpl	Result of operation is LT (less than)	$(CR\ field\ BF)_{0:2} = 100$ $(CR\ field\ BF)_3 = XER_{SO}$

4.3.2. Compare computation

Instructions tested	Compliance conditions	Expected result
I-3.3.10.Comp.1 Result of compare computation is correct		
cmpi cmp cmpli cmpl		Correct comparison result

4.3.3. Compare Ranged Byte and Compare Equal Byte

Instructions tested	Compliance conditions	Expected result
I-3.3.10.BFComp.1 The comparing instructions set CR-BF correctly and result of compare computation is correct		
cmprb cmpeqb		Correct CR field BF setting Correct comparison result

4.4. Fixed-Point Trap Instructions

Architecture sections:

- I-3.3.11 Fixed-point Trap Instructions

Scenario groups:

- Trap instructions computation

4.4.1. Trap instructions Computation

Instructions tested	Compliance conditions	Expected result
I-3.3.11.Comp.1 Result of trap instructions computation is correct		
twi tw tdi td		The system trap handler is invoked correctly

4.5. Fixed-Point Select

Architecture sections:

- I-3.3.12 Fixed-point Select

Scenario groups:

- Select Computation

4.5.1. Select Computation

Instructions tested	Compliance conditions	Expected result
I-3.3.12.Select.1 Target register is set correctly		
isel		RT gets the correct result according to the select computation

4.6. Fixed-Point Logical Instructions

Architecture sections:

- I-3.3.8 Other Fixed-Point Instructions
- I-3.3.13 Fixed-Point Logical Instructions

Scenario groups:

- Setting CR0
- Correct computations

4.6.1. Setting CR0

Recording instructions:

32-bit instructions: andi. andis. and. or. xor. nand. nor. eqv. andc. orc. extsb. extsh. cntlzw. cnttzw.

64 bit instructions: extsw. cntlzd. cnttzd.

Non-recording instructions:

32-bit instructions: ori oris xori xoris and or xor nand nor eqv andc orc extsb extsh cntlzw cnttzw cmpb popcntb popcntw prtyw

64 bit instructions: prtyd extsw popcntd cntlzd cnttzd bpermd

Instructions tested	Compliance conditions	Expected result
I-3.3.13.CR0.1 Recording instructions set CR0 correctly when result is zero		
Recording instructions	Result of operation is zero	First three bits of CR0 = 001
I-3.3.13.CR0.2 Recording instructions set CR0 correctly when result is positive		
Recording instructions	Result of operation is positive	First three bits of CR0 = 010
I-3.3.13.CR0.3 Recording instructions set CR0 correctly when result is negative		
Recording instructions	Result of operation is negative	First three bits of CR0 = 100
I-3.3.13.CR0.4 Non-recording instructions do not change CR0		
Non-recording instructions		First three bits of CR0 unchanged

4.6.2. Correct computations

Fixed-point simple logical instructions: and[.] or[.] xor[.] nand[.] nor[.] eqv[.] andc andc. orc orc.

Fixed-Point Logical Extending Instructions: extsb[.] extsh[.] extsw[.]

Fixed-Point Logical Count leading zero instructions: cntlzw[.] cntlzd[.]

Fixed-Point Logical Count trailing zero instructions: cnttzw[.] cnttzd[.]

Fixed-Point Logical immediate operations: andi. ori xori andis. xoris oris

Fixed-Point Logical Population Count Instructions: popcntb popcntw popcntd

Fixed-Point Logical Permute Instruction: bpermd

Fixed-Point Logical Parity Bits Instruction: prtyd prtyw

Fixed-Point Logical Compare Bytes Instruction: cmpb

Instructions tested	Compliance conditions	Expected result
I-3.3.13.Logic.1 Result of logical computation is correct		
Fixed-point simple logical instructions *Fixed-Point Logical Extending Instructions* *Fixed-Point Logical Count leading zero instructions* *Fixed-Point Logical Count trailing zero instructions* *Fixed-Point Logical immediate operations* *Fixed-Point Logical Population Count Instructions* *Fixed-Point Logical Permute Instruction* *Fixed-Point Logical Parity Bits Instruction* *Fixed-Point Logical Compare Bytes Instruction*		Correct logical result

4.7. Fixed-Point Rotate Instructions

Architecture sections:

- I-3.3.8 Other Fixed-Point Instructions
- I-3.3.14.1 Fixed-Point Rotate Instructions

Scenario groups:

- Setting CR0
- Correct Computations

4.7.1. Setting CR0

Recording instructions:

32-bit instructions: rlwinm. rlwnm. rlwimi.

64 bit instructions: rldicl. rldicr. rldic. rldcl. rldcr. rldimi.

Non-recording instructions:

32-bit instructions: rlwinm rlwnm rlwimi

64 bit instructions: rldicl rldicr rldic rldcl rldcr rldimi

Instructions tested	Compliance conditions	Expected result
I-3.3.14.1.CR0.1 Recording instructions set CR0 correctly when result is zero		
Recording instructions	Result of operation is zero	First three bits of CR0 = 001
I-3.3.14.1.CR0.2 Recording instructions set CR0 correctly when result is positive		
Recording instructions	Result of operation is positive	First three bits of CR0 = 010
I-3.3.14.1.CR0.3 Recording instructions set CR0 correctly when result is negative		
Recording instructions	Result of operation is negative	First three bits of CR0 = 100
I-3.3.14.1.CR0.4 Non-recording instructions do not change CR0		
Non-recording instructions		First three bits of CR0 unchanged

4.7.2. Correct Computations

All the Rotate instructions:

32-bit instructions: rlwinm. rlwnm. rlwimi. rlwinm rlwnm rlwimi

64 bit instructions: rldicl. rldicr. rldic. rldcl. rldcr. rldimi. rldicl rldicr rldic rldcl rldcr rldimi

Instructions tested	Compliance conditions	Expected result
I-3.3.14.1.Computation.1 By accurate rotate operation and mask calculation we get the correct result in the RA (target register)		
All the Rotate instructions		Correct result in the target register

4.8. Fixed-Point Shift Instructions

Architecture sections:

- I-3.3.8 Other Fixed-Point Instructions

- I-3.3.14.2 Fixed-Point Shift Instructions

Scenario groups:

- Setting CR0
- Setting CA and CA32
- Correct Computations

4.8.1. Setting CR0

Recording instructions:

32-bit instructions: slw. srw. srawi. sraw.

64 bit instructions: sld. srd. sradi. srad. extswsli.

Non-recording instructions:

32-bit instructions: slw srw srawi sraw

64 bit instructions: sld srd sradi srad extswsli

Instructions tested	Compliance conditions	Expected result
I-3.3.14.2.CR0.1 Recording instructions set CR0 correctly when result is zero		
Recording instructions	Result of operation is zero	First three bits of CR0 = 001
I-3.3.14.2.CR0.2 Recording instructions set CR0 correctly when result is positive		
Recording instructions	Result of operation is positive	First three bits of CR0 = 010
I-3.3.14.2.CR0.3 Recording instructions set CR0 correctly when result is negative		
Recording instructions	Result of operation is negative	First three bits of CR0 = 100
I-3.3.14.2.CR0.4 Non-recording instructions do not change CR0		
Non-recording instructions		First three bits of CR0 unchanged

4.8.2. Setting CA and CA32

Shift Algebraic Instructions:

32-bit instructions: srawi srawi. sraw sraw.

64-bit instructions: sradi sradi. srad srad.

Non-Algebraic instructions:

32-bit instructions: slw slw. srw srw.

64-bit instructions: sld sld. srd srd. extswsli extswsli.

Notation:

n the number of bits to be shifted

Note:

CA32 is set whenever CA is set, and is set to the same value that CA is defined to be set to in 32-bit mode.

Instructions tested	Compliance conditions	Expected result
I-3.3.14.2.CA.1 Algebraic instructions set CA correctly when no shift occurs		
Shift Algebraic Instructions	n=0	CA = 0
I-3.3.14.2.CA.2 Algebraic instructions set CA correctly when the shifted amount is positive		
Shift Algebraic Instructions	n!=0 (RS) is positive	CA = 0
I-3.3.14.2.CA.3 Algebraic instructions set CA correctly when the shifted bits are all zeros		
Shift Algebraic Instructions	n!=0 (RS) is negative 1-bits are not shifted out	CA = 0
I-3.3.14.2.CA.4 Algebraic instructions set CA correctly when the shifted bits include ones		
Shift Algebraic Instructions	n!=0 (RS) is negative Some 1-bits are shifted out	CA = 1
I-3.3.14.2.CA.5 Non-algebraic instructions do not change CA		
Non-Algebraic instructions		CA unchanged

4.8.3. Correct Computations

All Shift Instructions:

32-bit instructions: srawi srawi. sraw sraw. slw slw. srw srw.

64-bit instructions: sradi sradi. srad srad. sld sld. srd srd. extswsli extswsli.

Instructions tested	Compliance conditions	Expected result
I-3.3.14.2.Computation.1 By accurate shift operation and mask calculation we get the correct result in the RA (target register)		
All Shift Instructions		Correct result in the target register

4.9. Binary Coded Decimal Assist Instructions

Architecture sections:

- I-3.3.15 Binary Coded Decimal (BCD) Assist Instructions

Scenario groups:

- Correct Computations

4.9.1. Correct Computations

Instructions tested	Compliance conditions	Expected result
I-3.3.15.target.1 Setting the target register correctly		
cdtbcd cbcdtd addg6s		Correct result in the target register

4.10. Move To/From Vector-Scalar Register Instructions

Architecture sections:

- I-3.3.16 Move To/From Vector-Scalar Register Instructions

Scenario groups:

- Move from VSR SX BIT
- Move to VSR TX bit

4.10.1. Move from VSR SX BIT

Instructions tested	Compliance conditions	Expected result
I-3.3.16.SX.1 Cannot execute the instruction because the floating-point registers are unavailable		
mfvsrd mfvsrwz	SX = 0 MSR.FP = 0	Exiting with no results
I-3.3.16.SX.1b Cannot execute the instruction because the VSX registers are unavailable		
mfvsrld	SX = 0 MSR.VSX = 0	Exiting with no results
I-3.3.16.SX.2 Correct computation the result is taken from a floating-point VSR		
mfvsrd mfvsrwz	SX = 0 MSR.FP = 1	Correct result in the target register
I-3.3.16.SX.2b Correct computation the result is taken from a VSX VSR		
mfvsrld	SX = 0 MSR.VSX = 1	Correct result in the target register
I-3.3.16.SX.3 Cannot execute the instruction because the vector registers are unavailable		
mfvsrd mfvsrld mfvsrwz	SX = 1 MSR.VEC = 0	Exiting with no results
I-3.3.16.SX.4 Correct computation the result is taken from a vector VSR		
mfvsrd mfvsrld mfvsrwz	SX = 1 MSR.VEC= 1	Correct result in the target register

4.10.2. Move to VSR TX bit

Instructions tested	Compliance conditions	Expected result
I-3.3.16.TX.1 Cannot execute the instruction because the floating-point registers are unavailable		
mtvsrd mtvsrwa mtvsrwz	TX = 0 MSR.FP = 0	Exiting with no results
I-3.3.16.TX.1b Cannot execute the instruction because the VSX registers are unavailable		
mtvsrdd mtvsrws	TX = 0 MSR.VSX = 0	Exiting with no results
I-3.3.16.TX.2 Correctly set a floating-point VSR (VSR numbered 0-31)		

Instructions tested	Compliance conditions	Expected result
mtvsrd mtvsrwa mtvsrwz	TX = 0 MSR.FP = 1	Correct result in a floating-point VSR
I-3.3.16.TX.2b Correctly set a VSX VSR		
mtvsrdd mtvsrws	TX = 0 MSR.VSX = 1	Correct result in a VSX VSR
I-3.3.16.TX.3 Cannot execute the instruction because the vector registers are unavailable		
mtvsrd mtvsrwa mtvsrwz mtvsrdd mtvsrws	TX = 1 MSR.VEC = 0	Exiting with no results
I-3.3.16.TX.4 Correctly set a vector VSR (VSR numbered 32-63)		
mtvsrd mtvsrwa mtvsrwz mtvsrdd mtvsrws	TX = 1 MSR.VEC= 1	Correct result in a vector VSR

4.11. Move To/From System Register Instructions

Architecture sections:

- I-3.3.17 Move To/From System Register Instructions

Scenario groups:

- Move to CR
- Move from CR

4.11.1. Move to CR

Instructions tested	Compliance conditions	Expected result
I-3.3.17.To.1 Setting the CR field correctly		
mcrxrx mtocrf mtcrf		Correct result in the CR

4.11.2. Move from CR

Instructions tested	Compliance conditions	Expected result
I-3.3.17.From.1 Correct computation placing the contents of the CR correctly in the target register		
mfocrf mfcr setb		Correct computations

5. Floating Point Facility (Chapter I.4)

Table of Contents

5.1. Floating-Point Load Instructions

Architecture sections:

- I-4.6.2 Floating-Point Load Instructions

Scenario groups:

- Converting single-precision data
- Calculating the effective address (EA)
- Loaded data into FRT
- Setting RA
- Validity of the instruction
- Storage Access Exceptions

Guideline: each scenario in this section should be tested in both Big-Endian and Little-Endian mode

5.1.1. Converting single-precision data

Single-precision Load Floating-Point Instructions: lfs lfsx lfsu lfsux

Double-precision Load Floating-Point Instructions: lfd lfdx lfdu lfdux

Load Floating-Point as Integer Word instructions: lfiwzx lfiwax

Note: Because the FPRs support only floating-point double format, *single-precision Load Floating-Point Instructions* convert single-precision data to double format prior to loading the operand into the target FPR.

Instructions tested	Compliance conditions	Expected result
I-4.6.2.Convert.1 Correct conversion when the single-precision $WORD_{0:31}$ is a **Normalized** operand		
Single-precision Load Floating-Point Instructions	$WORD_{1:8} > 0$ $WORD_{1:8} < 255$	$FRT_{0:1} = WORD_{0:1}$ $FRT_2 = WORD_1$

Instructions tested	Compliance conditions	Expected result
		$FRT_3 = WORD_1$ $FRT_4 = WORD_1$ $FRT_{5:63} = WORD_{2:31} \mid\mid {}^{29}0$
I-4.6.2.Convert.2 Correct conversion when the single-precision $WORD_{0:31}$ is a **Denormalized** operand		
Single-precision Load Floating-Point Instructions	$WORD_{1:8} = 0$ $WORD_{9:31} \neq 0$	$FRT_0 = WORD_0$ $FRT_{1:11}$ = normalized_exp + 1023 $FRT_{12:63}$ = normalized_fraction$_{1:52}$
I-4.6.2.Convert.3 Correct conversion when the single-precision $WORD_{0:31}$ is a **Zero/NaN/Infinity** operand		
Single-precision Load Floating-Point Instructions	$WORD_{1:8} = 255$ Or $WORD_{1:31} = 0$	$FRT_{0:1} = WORD_{0:1}$ $FRT_2 = WORD_1$ $FRT_3 = WORD_1$ $FRT_4 = WORD_1$ $FRT_{5:63} = WORD_{2:31} \mid\mid {}^{29}0$
I-4.6.2.Convert.4 No conversion is needed in *double-precision Load Floating-Point Instructions* and for the *Load Floating-Point as Integer Word Algebraic instruction*		
double-precision Load Floating-Point Instructions *Load Floating-Point as Integer Word instructions*		The data from storage are copied directly into the FPR

5.1.2. Calculating the effective address (EA)

Instructions tested	Compliance conditions	Expected result
I-4.6.2.EA.1 The right computation of EA according to the operands and RA register		
lfs lfd		EA = (RA|0) + D
I-4.6.2.EA.2 The right computation of EA according to the RB and RA registers		
lfsx lfdx lfiwzx lfiwax		EA = (RA|0) + (RB)
I-4.6.2.EA.3 The right computation of EA according to the operands and non-zero RA register		
lfsu lfdu	RA != 0	EA = (RA)+D
I-4.6.2.EA.4 The right computation of EA according to the RB and non-zero RA registers		
lfsux lfdux	RA != 0	EA = (RA)+(RB)

5.1.3. Loaded data into FRT

Single-precision Load Floating-Point Instructions: lfs lfsx lfsu lfsux

Double-precision Load Floating-Point Instructions: lfd lfdx lfdu lfdux

Load Floating-Point as Integer Word Algebraic Indexed: lfiwax

Load Floating-Point as Integer Word and Zero Indexed: lfiwzx

Instructions tested	Compliance conditions	Expected result
I-4.6.2.FRT.1 For *Single-precision Load Floating-Point Instructions,* we convert (see *3.1.3.3.1*) the word in storage addressed by EA (see *3.1.3.3.2*) and load it into register FRT.		

Instructions tested	Compliance conditions	Expected result
Single-precision Load Floating-Point Instructions		FRT = converting-to-double(4 bytes of memory starting by EA)
I-4.6.2.FRT.2 For *Double-precision Load Floating-Point Instructions,* the doubleword in storage addressed by EA (see *3.1.3.3.2*) is loaded into register FRT.		
Double-precision Load Floating-Point Instructions		FRT = 8 bytes of memory starting by EA
I-4.6.2.FRT.3 For *Load Floating-Point as Integer Word Algebraic Indexed,* the word in storage addressed by EA (see *3.1.3.3.2*) is sign-extended and loaded into register FRT.		
Load Floating-Point as Integer Word Algebraic Indexed		$FRT_{32:63}$ = 4 bytes of memory starting by EA $FRT_{0:31}$ = sign-extension of $FRT_{32:63}$
I-4.6.2.FRT.4 For *Load Floating-Point as Integer Word and Zero Indexed,* the word in storage addressed by EA (see *3.1.3.3.2*) is zero-extended and loaded into register FRT.		
Load Floating-Point as Integer Word and Zero Indexed		$FRT_{32:63}$ = 4 bytes of memory starting by EA $FRT_{0:31} = {}^{32}0$

5.1.4. Setting RA

Update-Form Load Instructions: lfsu lfsux lfdu lfdux

Non-Update Load Instructions: lfs lfsx lfd lfdx lfiwzx lfiwax

Instructions tested	Compliance conditions	Expected result
I-4.6.2.RA.1 For *Update-Form Load Instructions,* RA is updated with the effective address (EA)		
Update-Form Load Instructions	RA != 0	(RA) = EA
I-4.6.2.RA.2 *Non-Update Load Instructions* dont change the RA		
Non-Update Load Instructions		RA unchanged

5.1.5. Validity of the instruction

Instructions tested	Compliance conditions	Expected result
I-4.6.2.Valid.1 In *Update-Form Load Instructions* if RA equals zero then the instruction is invalid		
Update-Form Load Instructions	RA = 0	Invalid instruction
I-4.6.2.Valid.2 *Non-Update Load Instructions* are always valid instructions		
Non-Update Load Instructions		Valid instruction

5.1.6. Storage Access Exceptions

Instructions tested	Compliance conditions	Expected result
I-4.6.2.StorageEx.1 A system data storage error handler will be invoked when an instruction attempts to access unavailable storage		
Floating-Point Load Instructions	The program attempts to access storage that is unavailable.	system data storage error handler is invoked

5.2. Floating-Point Store Instructions

Architecture sections:

- I-4.6.3 Floating-Point Store Instructions

Scenario groups:

- Converting double-precision data
- Calculating the effective address (EA)
- Stored data from FRS
- Setting RA
- Validity of the instruction
- Storage Access Exceptions

Guideline: each scenario in this section should be tested in both Big-Endian and Little-Endian mode

5.2.1. Converting double-precision data

Single-precision Store Floating-Point Instructions: stfs stfsx stfsu stfsux

Double-precision Store Floating-Point Instructions: stfd stfdx stfdu stfdux

Store Floating-Point as Integer Word instructions: stfiwx

Note: Because the FPRs support only floating-point double format for floating-point data, single-precision Store Floating-Point instructions convert double-precision data to single format prior to storing the operand into storage.

Instructions tested	Compliance conditions	Expected result
I-4.6.3.Convert.1 The conversion when no Denormalization is required (including Zero/ infinity / NaN)		
Single-precision Store Floating-Point Instructions	$FRS_{1:11} > 896$ Or $FRS_{1:63} = 0$	$WORD_{0:1} = FRS_{0:1}$ $WORD_{2:31} = FRS_{5:34}$
I-4.6.3.Convert.2 The conversion when the Denormalization is required and $FRS_{1:11}$ range is between 874 and 896		
Single-precision Store Floating-Point Instructions	$FRS_{1:11} \geq 874$ $FRS_{1:11} \leq 896$	$WORD_0 = FRS_0$ $WORD_{1:8} = 0x00$ $WORD_{9:31} = denormalized\_frac_{1:23}$
I-4.6.3.Convert.3 The conversion when the Denormalization is required and $FRS_{1:11}$ is not in the range 874-896		
Single-precision Store Floating-Point Instructions	$FRS_{1:11} < 874$ Or $FRS_{1:11} > 896$	WORD = undefined
I-4.6.3.Convert.4 For *Double-precision Store Floating-Point Instructions* and *Store Floating-Point as Integer Word instructions*, no conversion is required		
Double-precision Store Floating-Point Instructions *Store Floating-Point as Integer Word instructions*		The data from FRS is copied directly into storage

5.2.2. Calculating the effective address (EA)

Instructions tested	Compliance conditions	Expected result
I-4.6.3.EA.1 The right computation of EA according to the operands and RA register		

Instructions tested	Compliance conditions	Expected result
stfs stfd		EA = (RA|0) + D
I-4.6.3.EA.2 The right computation of EA according to the RB and RA registers		
stfsx stfdx stfiwx		EA = (RA|0) + (RB)
I-4.6.3.EA.3 The right computation of EA according to the operands and non-zero RA register		
stfsu stfdu	RA != 0	EA = (RA)+D
I-4.6.3.EA.4 The right computation of EA according to the RB and non-zero RA registers		
stfsux stfdux	RA != 0	EA = (RA)+(RB)

5.2.3. Stored data from FRS

Single-precision Store Floating-Point Instructions: stfs stfsx stfsu stfsux

Double-precision Store Floating-Point Instructions: stfd stfdx stfdu stfdux

Store Floating-Point as Integer Word instructions: stfiwx

Instructions tested	Compliance conditions	Expected result
I-4.6.3.FRT.1 For *Single-precision Store Floating-Point Instructions,* we convert (see *3.1.3.4.1*) the contents of register FRS to single format and store into the word in storage addressed by EA (see *3.1.3.4.2*).		
Single-precision Store Floating-Point Instructions		The word in storage addressed by EA = single-format(FRS)
I-4.6.3.FRT.2 For *Double-precision Store Floating-Point Instructions,* the contents of register FRS are stored into the doubleword in storage addressed by EA (see *3.1.3.4.2*).		
Double-precision Store Floating-Point Instructions		The doubleword in storage addressed by EA = (FRS)
I-4.6.3.FRT.3 For *Store Floating-Point as Integer Word instructions,* the contents $(FRS)_{32:63}$ are stored into the word in storage addressed by EA (see *3.1.3.4.2*).		
Store Floating-Point as Integer Word instructions		The word in storage addressed by EA = $(FRS)_{32:63}$

5.2.4. Setting RA

Update-Form Store Instructions: stfsu stfsux stfdu stfdux

Non-Update Store Instructions: stfs stfsx stfd stfdx stfiwx

Instructions tested	Compliance conditions	Expected result
I-4.6.3.RA.1 For *Update-Form Load Instructions,* RA is updated with the effective address (EA)		
Update-Form Store Instructions	RA != 0	(RA) = EA
I-4.6.3.RA.2 *Non-Update Load Instructions* dont change the RA		
Non-Update Store Instructions		RA unchanged

5.2.5. Validity of the instruction

Instructions tested	Compliance conditions	Expected result
I-4.6.3.Valid.1 In *Update-Form Store Instructions* if RA equals zero then the instruction is invalid		
Update-Form Store Instructions	RA = 0	Invalid instruction
I-4.6.3.Valid.2 *Non-Update Store Instructions* are always valid instructions		
Non-Update Store Instructions		Valid instruction

5.2.6. Storage Access Exceptions

Instructions tested	Compliance conditions	Expected result
I-4.6.3.StorageEx.1 A system data storage error handler will be invoked when an instruction attempts to access unavailable storage		
Fixed-Point store instructions	The program is not allowed to modify the target storage or the program attempts to access storage that is unavailable.	system data storage error handler is invoked

5.3. Floating-Point Move Instructions

Architecture sections:

- I-4.6.5 Floating-Point Move Instructions

Scenario groups:

- Setting CR1
- Correct computations

5.3.1. Setting CR1

Instructions tested	Compliance conditions	Expected result
I-4.6.5.CR1.1 In Recording instructions, CR1 field (bits 36:39 of the CR) is set to the FP exception status, copied from bits 32:35 of the FPSCR.		
fmr. fneg. fabs. fcpsgn. fnabs.	Rc = 1	CR1 = $FPSCR_{32:35}$
I-4.6.5.CR1.2 Non-recording instructions doesnt change the CR1 field		
fmr fneg fabs fcpsgn fnabs fmrgew fmrgow	If exists : Rc = 0	CR1 unchanged

5.3.2. Correct computations

Instructions tested	Compliance conditions	Expected result
I-4.6.5.Result.1 The contents of register FRB are placed into register FRT		
fmr fmr.		(FRT) = (FRB)
I-4.6.5.Result.2 The contents of register FRB with bit 0 inverted are placed into register FRT.		
fneg fneg.		$(FRT)_0 =_0$ $(FRT)_{1:63} = (FRB)_{1:63}$
I-4.6.5.Result.3 The contents of register FRB with bit 0 set to zero are placed into register FRT.		
fabs fabs.		$(FRT)_0 = 0$ $(FRT)_{1:63} = (FRB)_{1:63}$
I-4.6.5.Result.4 The contents of register FRB with bit 0 set to the value of bit 0 of register FRA are placed into register FRT.		
fcpsgn fcpsgn.		$(FRT)_0 = (FRA)_0$ $(FRT)_{1:63} = (FRB)_{1:63}$
I-4.6.5.Result.5 The contents of register FRB with bit 0 set to one are placed into register FRT.		
fnabs fnabs.		$(FRT)_0 = 1$ $(FRT)_{1:63} = (FRB)_{1:63}$
I-4.6.5.Result.6 Vector-Scalar instructions, that set the first words of vectors FRT and FRA into the vector FRT		

Instructions tested	Compliance conditions	Expected result
fmrgew	MSR.FP != 0	FPR[FRT].word[0] = FPR[FRA].word[0] FPR[FRT].word[1] = FPR[FRB].word[0]
I-4.6.5.Result.7 Vector-Scalar instructions, that set the second words of vectors FRT and FRA into the vector FRT		
fmrgow	MSR.FP != 0	FPR[FRT].word[0] = FPR[FRA].word[1] FPR[FRT].word[1] = FPR[FRB].word[1]

5.4. Floating-Point Load and Store Double Pair Instructions

Architecture sections:

- I-4.6.4 Floating-Point Load and Store Double Pair Instructions [Phased-Out]

Scenario groups:

- Calculating the effective address (EA)
- Loading double pair
- Storing double pair

5.4.1. Calculating the effective address (EA)

Instructions tested	Compliance conditions	Expected result
I-4.6.4.EA.1 The right computation of EA according to the operands and RA register		
lfdp stfdp stfdpx		EA = (RA|0) + (DS||0b00)
I-4.6.4.EA.2 The right computation of EA according to the RB and RA registers		
lfdpx		EA = (RA|0) + (RB)

5.4.2. Loading double pair

Instructions tested	Compliance conditions	Expected result
I-4.6.4.load.1 The doubleword in storage addressed by EA (see 3.1.3.6.1) is placed into the even-numbered register of FRTp.		
lfdp lfdpx	FRTp is not odd	$FRTp_{even}$ = the doubleword in storage addressed by EA
I-4.6.4.load.2 The doubleword in storage addressed by EA+8 (see 3.1.3.6.1) is placed into the odd-numbered register of FRTp.		
lfdp lfdpx	FRTp is not odd	$FRTp_{odd}$ = the doubleword in storage addressed by EA+8

5.4.3. Storing double pair

Instructions tested	Compliance conditions	Expected result
I-4.6.4.store.1 The contents of the even-numbered register of FRSp are stored into the doubleword in storage addressed by EA (see 3.1.3.6.1).		
stfdp stfdpx	FRSp is not odd	The doubleword in storage addressed by EA = $(FRSp)_{even}$
I-4.6.4.store.2 The contents of the odd-numbered register of FRSp are stored into the doubleword in storage addressed by EA+8.		
stfdp stfdpx	FRSp is not odd	The doubleword in storage addressed by EA+8 = $(FRSp)_{odd}$

5.4.4. Validity of the instruction

Instructions tested	Compliance conditions	Expected result
I-4.6.4.valid.1 In load double pair instructions, *if FRTp is odd then the instruction is invalid*		
lfdp lfdpx	FRTp is odd	Invalid instruction
I-4.6.4.valid.2 1 In store double pair instructions, *if FRSp is odd then the instruction is invalid*		
stfdp stfdpx	FRSp is odd	Invalid instruction

5.5. Floating-Point Select Instruction

Architecture sections:

- I-4.6.9 Floating-Point Select Instruction

Scenario groups:

- Setting CR1
- Setting FRT

5.5.1. Setting CR1

Instructions tested	Compliance conditions	Expected result
I-4.6.9.CR1.1 In Recording instructions, CR1 field (bits 36:39 of the CR) is set to the FP exception status, copied from bits 32:35 of the FPSCR.		
fsel.	Rc = 1	CR1 = $FPSCR_{32:35}$
I-4.6.9.CR1.2 Non-recording instructions doesnt change the CR1 field		
fsel	Rc = 0	CR1 unchanged

5.5.2. Setting FRT

Instructions tested	Compliance conditions	Expected result
I-4.6.9.FRT.1 FRT is set to the contents of register FRC if the FP operand in register FRA is greater than or equal to zero.		
fsel fsel.	FRA ≥ 0.0	(FRT) = (FRC)
I-4.6.9.FRT.2 FRT is set to the contents of register FRB if the operand in FRA is less than zero or is a NaN.		
fsel fsel.	FRA ≤ 0 OR FRA is NaN	(FRT) = (FRB)

5.6. Floating-Point Status and Control Register Instructions

Architecture sections:

- I-4.6.10 Floating-Point Status and Control Register Instructions

Scenario groups:

- Setting CR1

- Setting FPSCR bit BT+32
- Setting FX implicitly
- Setting FRT
- Setting CR field BF
- Setting FPSCR FX and exceptions bits
- Moving to FPSCR field immediate
- Moving to FPSCR fields

5.6.1. Setting CR1

Instructions tested	Compliance conditions	Expected result
I-4.6.10.CR1.1 In Recording instructions, CR1 field (bits 36:39 of the CR) is set to the FP exception status, copied from bits 32:35 of the FPSCR.		
mffs. mtfsfi. mtfsf. mtfsb0. mtfsb1.	Rc = 1	CR1 = $FPSCR_{32:35}$
I-4.6.10.CR1.2 Non-recording instructions doesnt change the CR1 field		
mffs mcrfs mtfsb0 mtfsb1 mtfsfi mtfsf	Rc = 0	CR1 unchanged

5.6.2. Setting FPSCR bit BT+32

Instructions tested	Compliance conditions	Expected result
I-4.6.10.BT.1 Turning off the FPSCR bit BT+32 explicitly		
mtfsb0 mtfsb0.	BT+32 != 33 BT+32 != 34	$FPSCR_{BT+32}$ = 0
I-4.6.10.BT.2 Turning on the FPSCR bit BT+32 explicitly, when no exceptions are caused		
mtfsb1 mtfsb1	BT+32 != 33 BT+32 != 34 BT+32 != 53	$FPSCR_{BT+32}$ = 1
I-4.6.10.BT.3 Turning on the FPSCR bit BT+32 explicitly, when an invalid operation exception is caused		
mtfsb1 mtfsb1	BT+32 = 53	$FPSCR_{BT+32}$ = 1 Invalid operation exception occurs ($FPSCR_{VXSOFT}$ = 1)

5.6.3. Setting FX implicitly

Instructions tested	Compliance conditions	Expected result
I-4.6.10.FX.1 Turning on the FPSCR FX bit ($FPSCR_{32}$) implicitly when the instruction causes any of the floating-point exception bits in the FPSCR to change from 0 to 1.		
mtfsb1	BT+32 $\bigvee$ {35-44 , 53-55}	$FPSCR_{BT+32}$ = 1 $FPSCR_{32}$ = 1
I-4.6.10.FX.2 The *mtfsfi[.]* instructions cant implicitly change the FPSCR FX bit, even if any exception bit is turned on by the instruction		
mtfsfi mtfsfi.	BF + 8(1-W) $\bigvee$ {9, 10, 11, 13}	$FPSCR_{32}$ unchanged
I-4.6.10.FX.3 The *mtfsf[.]* instructions cant implicitly change the FPSCR FX bit, even if any exception bit is turned on by the instruction		
mtfsf mtfsf.	i + 8(1-W) $\bigvee$ {9, 10, 11, 13}. Where i ranges from 0 to 7	$FPSCR_{32}$ unchanged

5.6.4. Setting FRT

Instructions tested	Compliance conditions	Expected result
I-4.6.10.FRT.1 Copying the appropriate contents of the FPSCR to the register FRT and performing correct instruction operation		
mffs mffs. mffsce mffscdrn mffscdrni mffscrn mffscrni mffsl		Copy appropriate contents of FPSCR to register FRT and perform correct instruction operation

5.6.5. Setting CR field BF

Instructions tested	Compliance conditions	Expected result
I-4.6.10.BF.1 Copying the contents of the $FPSCR_{32:63}$ field BFA into the CR field BF		
mcrfs		(CR field BF) = ($FPSCR_{32:63}$ field BFA)

5.6.6. Setting FPSCR FX and exceptions bits

Instructions tested	Compliance conditions	Expected result
I-4.6.10.BFA.1 Setting the FPSCR FX and OX bits to 0 when BFA operand equals 0, after they are copied into the CR field BF		
mcrfs	BFA = 0	$FPSCR_{32}$ = 0 $FPSCR_{35}$ = 0
I-4.6.10.BFA.2 Setting the FPSCR UX, ZX, XX and VXSNAN bits to 0 when BFA operand equals 1, after they are copied into the CR field BF		
mcrfs	BFA = 1	$FPSCR_{36}$ = 0 $FPSCR_{37}$ = 0 $FPSCR_{38}$ = 0 $FPSCR_{39}$ = 0
I-4.6.10.BFA.3 Setting the FPSCR VXISI, VXIDI, VXZDZ and VXIMZ bits to 0 when BFA operand equals 2, after they are copied into the CR field BF		
mcrfs	BFA = 2	$FPSCR_{40}$ = 0 $FPSCR_{41}$ = 0 $FPSCR_{42}$ = 0 $FPSCR_{43}$ = 0
I-4.6.10.BFA.4 Setting the FPSCR VXVC bit to 0 when BFA operand equals 3, after it is copied into the CR field BF		
mcrfs	BFA = 3	$FPSCR_{44}$ = 0
I-4.6.10.BFA.5 Setting the FPSCR VXSOFT, VXSQRT and VXCVI bits to 0 when BFA operand equals 5, after they are copied into the CR field BF		
mcrfs	BFA = 5	$FPSCR_{53}$ = 0 $FPSCR_{54}$ = 0 $FPSCR_{55}$ = 0

5.6.7. Moving to FPSCR field immediate

Instructions tested	Compliance conditions	Expected result
I-4.6.10.MTFFI.1 The value of the U operand field is placed into FPSCR field BF+8(1-W). (U , BF and W are operands)		

Instructions tested	Compliance conditions	Expected result
mtfsfi mtfsfi.		$FPSCR_{BF+8(1-W)} = U$
I-4.6.10.MTFFI.2 When the FPSCR field is the $FPSCR_{32:35}$, then not all the bits changes according to U. the *mtfsfi[.]* instruction cant implicitly change FX neither change FEX and VX bits		
mtfsfi mtfsfi.	BF = 0 W = 0	$FPSCR_{32} = U_0$ $FPSCR_{35} = U_3$ $FPSCR_{33}$ unchanged $FPSCR_{34}$ unchanged
I-4.6.10.MTFFI.3 When the FPSCR specified field contains $FPSCR_{53}$ (VXSOFT) bit ,and the appropriate bit in U is 1, then an invalid operation exception occurs		
	$FPSCR_{53}$ is specified in the $FPSCR_{BF+8(1-W)}$ field $U_i = 1$ (i is the equivalent bit for $FPSCR_{53}$)	$FPSCR_{53} = 1$ Invalid operation exception occurs ($FPSCR_{VXSOFT} = 1$)

5.6.8. Moving to FPSCR fields

Instructions tested	Compliance conditions	Expected result
I-4.6.10.MTFF.1 The FPSCR is modified as specified by the FLM and W when L = 0		
mtfsf mtfsf.	L = 0 $FLM_i = 1$ (i ranges from 0 to 7)	(FPSCR field i+8(1-W)) = (FRB field i+8(1-W)) *If (FRB) $_{53}$ is specified and (FRB) $_{53}$ = 1 then:* Invalid operation exception occurs ($FPSCR_{VXSOFT} = 1$)
I-4.6.10.MTFF.2 The FRB contents are placed into the FPSCR when L = 1 and (FRB)53 != 1		
mtfsf mtfsf.	L = 1 $(FRB)_{53}$!= 1	(FPSCR) = (FRB)
I-4.6.10.MTFF.3 The FRB contents are placed into the FPSCR when L = 1 and (FRB)53 = 1		
mtfsf mtfsf.	L = 1 $(FRB)_{53}$!= 1	(FPSCR) = (FRB) Invalid operation exception occurs ($FPSCR_{VXSOFT} = 1$)
I-4.6.10.MTFF.4 If the destination includes $FPSCR_{32:35}$ then FX and OX are set to the appropriate values of FRB, bits 33 and 34 (FEX and VX) dont change according to the FRB		
mtfsf mtfsf.	L = 1 or L = 0 $FPSCR_{32:35}$ is specified	$FPSCR_{32} = (FRB)_{32}$ $FPSCR_{35} = (FRB)_{35}$ $FPSCR_{33}$ unchanged $FPSCR_{34}$ unchanged

5.7. Floating-Point Arithmetic, Rounding, and Conversion Instructions

Architecture sections:

- I-4.6.6 Floating-Point Arithmetic Instructions

- I-4.6.7 Floating-Point Rounding and Conversion Instructions
- I-4.3 Floating-Point Data

Scenario groups:

- Setting FR (fraction rounded) and FI (fraction inexact)
- Setting FPRF (result flags)
- Correct computations
- Rounding modes

Floating-point arithmetic, rounding, and conversion instructions: fadd[s][.] fsub[s][.] fmul[s][.] fdiv[s][.] fsqrt[s][.] fre[s][.] frsqrte[s][.] fmadd[s][.] fmsub[s][.] fnmadd[s][.] fnmsub[s][.] frsp[.] fctid[.] fctidz[.] fctiw[.] fctiwz[.] fcfid[.] frin[.] frip[.] friz[.] frim[.] ftdiv ftsqrt fctidu[.] fctiduz[.] fctiwu[.] fctiwuz[.] fcfidu[.] fcfids[.] fcfidus[.]

5.7.1. Setting FR (fraction rounded) and FI (fraction inexact)

Floating-point potentially rounding instructions:

fadd[s][.] fsub[s][.] fmul[s][.] fdiv[s][.] fsqrt[s][.] fmadd[s][.] fmsub[s][.] fnmadd[s][.] fnmsub[s][.] frsp[.] fctid[z][.] fctiw[z][.] fcfid[.] fctidu[z][.] fctiwu[z][.] fcfidu[.] fcfids[.] fcfidus[.]

Floating-point round-to-integer instructions: frin[.] friz[.] frip[.] frim[.]

Notes:

The following instructions are not included in the list of *potentially rounding instructions*, since they set FR and FI to an undefined value: fre[.] fres[.] fsqrte[.]

Instructions tested	Compliance conditions	Expected result
I-4.6.6.FR.1 FR is set to 1 when the fraction is incremented during rounding		
Floating-point potentially rounding instructions	The fraction is incremented during rounding of the intermediate result No exception occurs	FR=1
I-4.6.6.FR.2 *Round to integer instructions* set FR to zero even if fraction is incremented during rounding		
Floating-point round-to-integer instructions	The fraction is incremented during rounding of the intermediate result No exception occurs	FR=0
I-4.6.6.FR.3 FR is not sticky		
Floating-point potentially rounding instructions	The fraction is not incremented during rounding of the intermediate result	FR=0
I-4.6.6.FR.4 FI is set to 1 when result is inexact		
Floating-point potentially rounding instructions	The rounded result differs from the intermediate result (this implies that Inexact Exception occurs) No exception other than Inexact Exception occurs	FI=1

Instructions tested	Compliance conditions	Expected result
I-4.6.6.FR.5 *Round to integer instructions* set FI to zero even when result is inexact		
Floating-point round to integer instructions	The rounded result differs from the intermediate result (this implies that Inexact Exception occurs) No exception other than Inexact Exception occurs	FI=0
I-4.6.6.FR.6 FI is not sticky		
Floating-point potentially rounding instructions	The rounded result is equal to the intermediate result	FI=0

5.7.2. Setting FPRF (result flags)

Instructions tested	Compliance conditions	Expected result
I-4.6.6.FPRF.1 Arithmetic/rounding/conversion instructions set FPRF correctly for Quiet NaN result		
Floating-point arithmetic, rounding, and conversion instructions	Result value class is Quiet NaN	FPRF = 10001
I-4.6.6.FPRF.2 Arithmetic/rounding/conversion instructions set FPRF correctly for Infinity result		
Floating-point arithmetic, rounding, and conversion instructions	Result value class is Infinity	FPRF = 01001
I-4.6.6.FPRF.3 Arithmetic/rounding/conversion instructions set correctly for Normalized result		
Floating-point arithmetic, rounding, and conversion instructions	Result value class is Normalized Number	FPRF = 01000
I-4.6.6.FPRF.4 Arithmetic/rounding/conversion instructions set FPRF correctly for Denormalized result		
Floating-point arithmetic, rounding, and conversion instructions	Result value class is Denormalized Number	FPRF = 11000
I-4.6.6.FPRF.5 Arithmetic/rounding/conversion instructions set FPRF correctly for Zero result		
Floating-point arithmetic, rounding, and conversion instructions	Result value class is Zero	FPRF = 10010
I-4.6.6.FPRF.6 Arithmetic/rounding/conversion instructions set FPRF correctly for +Zero result		
Floating-point arithmetic, rounding, and conversion instructions	Result value class is +Zero	FPRF = 00010
I-4.6.6.FPRF.7 Arithmetic/rounding/conversion instructions set FPRF correctly for +Denormalized result		
Floating-point arithmetic, rounding, and conversion instructions	Result value class is +Denormalized Number	FPRF = 10100
I-4.6.6.FPRF.8 Arithmetic/rounding/conversion instructions set FPRF correclyt for +Normalized result		
Floating-point arithmetic, rounding, and conversion instructions	Result value class is +Normalized Number	FPRF = 00100
I-4.6.6.FPRF.9 Arithmetic/rounding/conversion instructions set FPRF correctly for +Infinity result		
Floating-point arithmetic, rounding, and conversion instructions	Result value class is +Infinity	FPRF = 00101

5.7.3. Correct computation

Instructions tested	Compliance conditions	Expected result
I-4.6.6.Comp.1 Arithmetic/rounding/conversion instructions perform correct computations		
Floating-point arithmetic, rounding, and conversion instructions	No exception occurs	Correct result of operation

5.7.4. Rounding modes

Floating-point addition/subtraction instructions: fadd[s][.] fsub[s][.] fmadd[s][.] fmsub[s][.] fnmadd[s][.] fnmsub[s][.]

Instructions tested	Compliance conditions	Expected result
I-4.6.6.Round.1 Round to Nearest mode		
Representative of *Floating-point arithmetic, rounding, and conversion instructions* excluding *frin*	RN = 00 No exception occurs Result is numeric Numeric operands	Rounding is performed according to Round to Nearest mode
I-4.6.6.Round.2 For add/subtract operations, sign of a zero result is positive in Round to Nearest mode		
Representative of *Floating-point addition/ subtraction instructions*	RN = 00 No exception occurs Result is zero Numeric operands	Sign of the result is positive
I-4.6.6.Round.3 Round toward Zero mode		
Representative of *Floating-point arithmetic, rounding, and conversion instructions*	RN = 01 No exception occurs Result is numeric Numeric operands	Rounding is performed according to Round toward Zero mode
I-4.6.6.Round.4 For add/subtract operations, sign of a zero result is positive in Round toward Zero mode		
Representative of *Floating-point addition/ subtraction instructions*	RN = 01 No exception occurs Result is zero Numeric operands	Sign of the result is positive
I-4.6.6.Round.5 Round toward +Infinity mode		
Representative of *Floating-point arithmetic, rounding, and conversion instructions*	RN = 10 No exception occurs Result is numeric Numeric operands	Rounding is performed according to Round toward +Infinity mode
I-4.6.6.Round.6 For add/subtract operations, sign of a zero result is positive in Round toward +Infinity mode		
Representative of *Floating-point addition/ subtraction instructions*	RN = 10 No exception occurs Result is zero Numeric operands	Sign of the result is positive
I-4.6.6.Round.7 Round toward Infinity mode		
Representative of *Floating-point arithmetic, rounding, and conversion instructions*	RN = 11 No exception occurs Result is numeric Numeric operands	Rounding is performed according to Round toward Infinity mode

Instructions tested	Compliance conditions	Expected result
I-4.6.6.Round.8 For add/subtract operations, sign of a zero result is positive in Round toward Infinity mode		
Representative of *Floating-point addition/ subtraction instructions*	RN = 11 No exception occurs Result is zero Numeric operands	Sign of the result is negative
I-4.6.6.Round.9 The instruction "frin" does not implement the IEEE Round to Nearest function		
frin	RN = 00 No exception occurs Result is zero Numeric operands	No occurrence of the Round to Nearest mode

5.8. Floating-Point Compare Instructions

Architecture sections:

- I-4.6.8 Floating-Point Compare Instructions

Scenario groups:

- Setting CR field BF and FPCC

5.8.1. Setting CR field BF and FPCC

Instructions tested	Compliance conditions	Expected result
I-4.6.8.Compare.1 If one of the operands is a NaN, comparison result is set to reflect unordered		
fcmpu fcmpo	(FRA) is a NaN, or (FRB) is a NaN	FPCC = 0001 $CR_{4BF:4BF+3}$ = 0001
I-4.6.8.Compare.2 Comparison result is Less Than		
fcmpu fcmpo	(FRA) and (FRB) are not NaNs (FRA) < (FRB)	FPCC = 1000 $CR_{4BF:4BF+3}$ = 1000
I-4.6.8.Compare.3 Comparison result is Greater Than		
fcmpu fcmpo	(FRA) and (FRB) are not NaNs (FRA) > (FRB)	FPCC = 0100 $CR_{4BF:4BF+3}$ = 0100
I-4.6.8.Compare.4 Comparison result is Equals		
fcmpu fcmpo	(FRA) and (FRB) are not NaNs (FRA) = (FRB)	FPCC = 0010 $CR_{4BF:4BF+3}$ = 0010
I-4.6.8.Compare.5 Comparison of zeros with opposite sign gives Equal result		
fcmpu fcmpo	(FRA) = 0 (FRB) = 0 (FRA) and (FRB) have opposite signs (i.e., they are +0 and 0)	FPCC = 0010 $CR_{4BF:4BF+3}$ = 0010
I-4.6.8.Compare.6 Comparison of +Infinity gives Equal result		
fcmpu fcmpo	(FRA) = +∞	FPCC = 0010

Instructions tested	Compliance conditions	Expected result
	(FRB) = +∞	$CR_{4BF:4BF+3}$ = 0010
I-4.6.8.Compare.7 Comparison of Infinity gives Equal result		
fcmpu fcmpo	(FRA) = -∞ (FRB) = -∞	FPCC = 0010 $CR_{4BF:4BF+3}$ = 0010

5.9. Floating-Point Exceptions

Architecture sections:

- I-4.4 Floating-Point Exceptions

Notes:

In this section, wherever intermediate results (before rounding) are mentioned, they can be regarded as having infinite precision and unbounded exponent range.

Floating-point addition/subtraction instructions: fadd[s][.] fsub[s][.] fmadd[s][.] fmsub[s][.] fnmadd[s][.] fnmsub[s][.]

Floating-point multiplication instructions: fmul[s][.] fmadd[s][.] fmsub[s][.] fnmadd[s][.] fnmsub[s][.]

Floating-point conversion-to-integer instructions: fctid[z][.] fctiw[z][.] fctidu[z][.] fctiwu[z][.]

Floating-point potentially overflowing/underflowing instructions: fdiv[s][.] fsqrt[s][.] fre[s][.] frsqte[s][.] frsp[.] as well as the union of *Floating-point addition/subtraction instructions* and *Floating-point multiplication instructions*

Floating-point potentially inexact instructions: fdiv[s][.] fsqrt[s][.] frsp[.] fcfid[.] fcfidu[.] fcfids[.] fcfidus[.] as well as the union of *Floating-point addition/subtraction instructions*, *Floating-point multiplication instructions*, and *Floating-point conversion-to-integer instructions*

Floating-point potential-SNaN-operand instructions: fdiv[s][.] fsqrt[s][.] fre[s][.] frsqrte[s][.] frsp[.] frin[.] frip[.] friz[.] frim[.] fcmpu fcmpo as well as the union of *Floating-point addition/subtraction instructions*, *Floating-point multiplication instructions*, and *Floating-point conversion-to-integer instructions*

Floating-point multiply-add instructions: fmadd[s][.] fmsub[s][.] fnmadd[s][.] fnmsub[s][.]

5.9.1. Invalid Operation Exception

Architecture sections:

- I-4.4.1Invalid Operation Exception

Scenario groups:

- Setting exception bits
- Keeping exception bits unchanged
- Actions taken when the exception is enabled
- Action taken when the exception is disabled

Condition for Invalid Operation Exception (SNaN): one of the operands of a floating-point operation is a Signaling NaN.

Condition for Invalid Operation Exception (∞ - ∞): magnitude subtraction of infinities occurs.

Condition for Invalid Operation Exception (∞ ∞): division of infinity by infinity occurs.

Condition for Invalid Operation Exception (0 0): division of zero by zero occurs.

Condition for Invalid Operation Exception (∞ 0): multiplication of infinity by zero occurs.

Condition for Invalid Operation Exception (Invalid Compare): ordered comparison involving an SNaN when Invalid Operation is disabled, or involving a QNaN.

Condition for Invalid Operation Exception (Invalid Square Root): square root or reciprocal square root of a negative (and nonzero) number.

Condition for Invalid Operation Exception (Invalid Integer Convert): integer convert involving a number too large in magnitude to be represented in the target format, or involving an infinity or a NaN.

5.9.1.1. Setting exception bits

Instructions tested	Compliance conditions	Expected result
I-4.4.1.Invalid.1 VXSNAN is set to 1 when one of the operands is an SNaN		
Floating-point potential-SNaN-operand instructions	*Condition for Invalid Operation Exception (SNaN)* occurs	VXSNAN = 1
I-4.4.1.Invalid.2 Add/subtract operations set VXISI to 1 when magnitude subtraction of infinities occurs		
Floating-point addition/ subtraction instructions	*Condition for Invalid Operation Exception (∞ - ∞)* occurs	VXISI = 1
I-4.4.1.Invalid.3 Division operations set VXIDI to 1 when division of infinity by infinity occurs		
fdiv[s][.]	*Condition for Invalid Operation Exception (∞ ∞)* occurs	VXIDI = 1
I-4.4.1.Invalid.4 Division operations set VXZDZ to 1 when division of zero by zero occurs		
fdiv[s][.]	*Condition for Invalid Operation Exception (0 0)* occurs	VXZDZ = 1
I-4.4.1.Invalid.5 Multiplication operations set VXIMZ to 1 when multiplication of infinity by zero occurs		
Floating-point multiplication instructions	*Condition for Invalid Operation Exception (∞ 0)* occurs	VXIMZ = 1
I-4.4.1.Invalid.6 Ordered comparison sets VXVC to 1 when the comparison involves a NaN		
fcmpo	*Condition for Invalid Operation Exception (Invalid Compare)* occurs	VXVC = 1
I-4.4.1.Invalid.7 Square root operations set VXSQRT to 1 when applied to a negative nonzero number		
fsqrt[s][.] frsqrte[s][.]	*Condition for Invalid Operation Exception (Invalid Square Root)* occurs	VXSQRT = 1
I-4.4.1.Invalid.8 Integer conversion operations set VXCVI to 1 when number is too large		
Floating-point conversion-to-integer instructions	Integer conversion involving a number too large in magnitude to be represented in the target format occurs.	VXCVI = 1
I-4.4.1.Invalid.9 Integer conversion operations set VXCVI to 1 when conversion involves an infinity		
Floating-point conversion-to-integer instructions	Integer conversion involving an infinity occurs	VXCVI = 1
I-4.4.1.Invalid.10 Integer conversion operations set VXCVI to 1 when conversion involves a NaN		
Floating-point conversion-to-integer instructions	Integer conversion involving a NaN occurs	VXCVI = 1

5.9.1.2. Keeping exception bits unchanged

Instructions tested	Compliance conditions	Expected result
I-4.4.1.InvalidConst.1 VXSNAN is sticky		
Representative of *Floating-point potential-SNaN-operand instructions*	VXSNAN=1 before instruction execution *Condition for Invalid Operation Exception (SNaN)* does **not** occur	VXSNAN = 1
I-4.4.1.InvalidConst.2 VXSNAN stays zero when exception condition **does not occur**		
Representative of *Floating-point potential-SNaN-operand instructions*	VXSNAN=0 before instruction execution *Condition for Invalid Operation Exception (SNaN)* does **not** occur	VXSNAN = 0
I-4.4.1.InvalidConst.3 VXISI is sticky		
Representative of *Floating-point addition/subtraction instructions*	VXISI = 1 before instruction execution *Condition for Invalid Operation Exception (∞ - ∞)* does **not** occur	VXISI = 1
I-4.4.1.InvalidConst.4 VXISI stays zero when exception condition **does not occur**		
Representative of *Floating-point addition/subtraction instructions*	VXISI = 0 before instruction execution *Condition for Invalid Operation Exception (∞ - ∞)* does **not** occur	VXISI = 0
I-4.4.1.InvalidConst.5 VXIDI is sticky		
Representative of: fdiv[s][.]	VXIDI = 1 before instruction execution *Condition for Invalid Operation Exception (∞ ∞)* does **not** occur	VXIDI = 1
I-4.4.1.InvalidConst.6 VXIDI stays zero when exception condition **does not occur**		
Representative of: fdiv[s][.]	VXIDI = 0 before instruction execution *Condition for Invalid Operation Exception (∞ ∞)* does **not** occur	VXIDI = 0
I-4.4.1.InvalidConst.7 VXZDZ is sticky		
Representative of: fdiv[s][.]	VXZDZ = 1 before instruction execution *Condition for Invalid Operation Exception (0 0)* does **not** occur	VXZDZ = 1
I-4.4.1.InvalidConst.8 VXZDZ stays zero when exception condition **does not occur**		
Representative of: fdiv[s][.]	VXZDZ = 0 before instruction execution *Condition for Invalid Operation Exception (0 0)* does **not** occur	VXZDZ = 0
I-4.4.1.InvalidConst.9 VXIMZ is sticky		
Representative of *Floating-point multiplication instructions*	VXIMZ = 1 before instruction execution *Condition for Invalid Operation Exception (∞ 0)* does **not** occur	VXIMZ = 1
I-4.4.1.InvalidConst.10 VXIMZ stays zero when exception condition **does not occur**		
Representative of *Floating-point multiplication instructions*	VXIMZ = 0 before instruction execution *Condition for Invalid Operation Exception (∞ 0)* does **not** occur	VXIMZ = 0
I-4.4.1.InvalidConst.11 VXVC is sticky		
fcmpo	VXVC = 1 before instruction execution *Condition for Invalid Operation Exception (Invalid Compare)* does **not** occur	VXVC = 1

Instructions tested	Compliance conditions	Expected result
I-4.4.1.InvalidConst.12 VXVC stays zero when exception condition **does not occur**		
fcmpo	VXVC = 0 before instruction execution *Condition for Invalid Operation Exception (Invalid Compare)* does **not** occur	VXVC = 0
I-4.4.1.InvalidConst.13 VXSQRT is sticky		
Representative of: fsqrt[s][.] frsqrte[s][.]	VXSQRT = 1 before instruction execution *Condition for Invalid Operation Exception (Invalid Square Root)* does **not** occur	VXSQRT = 1
I-4.4.1.InvalidConst.14 VXSQRT stays zero when exception condition **does not occur**		
Representative of: fsqrt[s][.] frsqrte[s][.]	VXSQRT = 0 before instruction execution *Condition for Invalid Operation Exception (Invalid Square Root)* does **not** occur	VXSQRT = 0
I-4.4.1.InvalidConst.15 VXCVI is sticky		
Representative of *Floating-point conversion-to-integer instructions*	VXCVI = 1 before instruction execution *Condition for Invalid Operation Exception (Invalid Integer Convert)* does **not** occur	VXCVI = 1
I-4.4.1.InvalidConst.16 VXCVI stays zero when exception condition **does not occur**		
Representative of *Floating-point conversion-to-integer instructions*	VXCVI = 0 before instruction execution *Condition for Invalid Operation Exception (Invalid Integer Convert)* does **not** occur	VXCVI = 0

5.9.1.3. Actions taken when the exception is enabled

Floating-point multiply-add instructions: fmadd[s][.] fmsub[s][.] fnmadd[s][.] fnmsub[s][.]

Floating round to integer instructions: frin[.] frip[.] friz[.] frim[.]

Floating convert to integer instructions: fctid[.] fctidz[.] fctiw[.] fctiwz[.] fctidu[.] fctiduz[.] fctiwu[.] fctiwuz[.]

Instructions tested	Compliance conditions	Expected result
I-4.4.1.VE1.1 The actions that will be taken if the operation belongs to *Floating-point multiply-add instructions*		
Floating-point multiply-add instructions	$FPSCR_{VE}$ = 1 $FPSCR_{VXSNAN}$ = 1, or $FPSCR_{VXISI}$ = 1, or $FPSCR_{VXIMZ}$ = 1	Target FPR of the instruction is unchanged $FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is unchanged
I-4.4.1.VE1.2 The actions that will be taken if the operation includes Square Root		
fsqrt[s][.] frsqrte[s][.]	$FPSCR_{VE}$ = 1 $FPSCR_{VXSNAN}$ = 1, or $FPSCR_{VXSQRT}$ = 1	Target FPR of the instruction is unchanged $FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is unchanged
I-4.4.1.VE1.3 The actions that will be taken if the operation includes rounding or reciprocal estimate		
Floating round to integer instructions frsp[.]	$FPSCR_{VE}$ = 1 $FPSCR_{VXSNAN}$ = 1	Target FPR of the instruction is unchanged $FPSCR_{FI}$ = 0

Instructions tested	Compliance conditions	Expected result
fre[s][.]		$FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is unchanged
I-4.4.1.VE1.4 The actions that will be taken if the operation belongs to *Floating convert to integer instructions*		
Floating convert to integer instructions	$FPSCR_{VE}$ = 1 $FPSCR_{VXSNAN}$ = 1, or $FPSCR_{VXCVI}$ = 1	Target FPR of the instruction is unchanged $FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is unchanged
I-4.4.1.VE1.5 The actions that will be taken if it is divide operation		
fdiv[s][.]	$FPSCR_{VE}$ = 1 $FPSCR_{VXSNAN}$ = 1, or $FPSCR_{VXIDI}$ = 1, or $FPSCR_{VXZDZ}$ = 1	Target FPR of the instruction is unchanged $FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is unchanged
I-4.4.1.VE1.6 The actions that will be taken if it is multiplication operation		
fmul[s][.]	$FPSCR_{VE}$ = 1 $FPSCR_{VXSNAN}$ = 1, or $FPSCR_{VXIMZ}$ = 1	Target FPR of the instruction is unchanged $FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is unchanged
I-4.4.1.VE1.7 The actions that will be taken if it is add/subtract operation		
fadd[s][.] fsub[s][.]	$FPSCR_{VE}$ = 1 $FPSCR_{VXSNAN}$ = 1, or $FPSCR_{VXISI}$ = 1	Target FPR of the instruction is unchanged $FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is unchanged
I-4.4.1.VE1.8 The actions that will be taken if the operation is compare unordered		
fcmpu	$FPSCR_{VE}$ = 1 $FPSCR_{VXSNAN}$ =1	$FPSCR_{FI}$ is unchanged $FPSCR_{FR}$ is unchanged $FPSCR_{C}$ is unchanged $FPSCR_{FPCC}$ is set to reflect unordered
I-4.4.1.VE1.9 The actions that will be taken if the operation is compare ordered		
fcmpo	$FPSCR_{VE}$ = 1 $FPSCR_{VXSNAN}$ =1 Or $FPSCR_{VXVC}$ = 1	$FPSCR_{FI}$ is unchanged $FPSCR_{FR}$ is unchanged $FPSCR_{C}$ is unchanged $FPSCR_{FPCC}$ is set to reflect unordered
I-4.4.1.VE1.10 The actions that will be taken if an *mtfsfi, mtfsf,* or *mtfsb1* instruction is executed that sets $FPSCR_{VXSOFT}$ to 1		
mtfsfi mtfsf mtfsb1	$FPSCR_{VE}$ = 1 $FPSCR_{VXSOFT}$ = 1	FPSCR is set as specified in the instruction description

5.9.1.4. Action taken when the exception is disabled

Floating round to integer instructions: frin[.] frip[.] friz[.] frim[.]

Floating-point multiply-add instructions: fmadd[s][.] fmsub[s][.] fnmadd[s][.] fnmsub[s][.]

Convert to 32-bit integer instructions: fctiw[.] fctiwz[.] fctiwu[.] fctiwuz[.]

Convert to 64-bit integer instructions: fctid[.] fctidz[.] fctidu[.] fctiduz[.]

Instructions tested	Compliance conditions	Expected result
I-4.4.1.VE0.1 The actions that will be taken if the operation belongs to *Floating-point multiply-add instructions*		
Floating-point multiply-add instructions	$FPSCR_{VE}$ = 0 $FPSCR_{VXSNAN}$ = 1, or $FPSCR_{VXISI}$ = 1, or $FPSCR_{VXIMZ}$ = 1	Target FPR is set to a QNaN $FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is set to indicate the class of the result (QNaN)
I-4.4.1.VE0.2 The actions that will be taken if the operation includes Square Root		
fsqrt[s][.] frsqrte[s][.]	$FPSCR_{VE}$ = 0 $FPSCR_{VXSNAN}$ = 1, or $FPSCR_{VXSQRT}$ = 1	Target FPR is set to a QNaN $FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is set to indicate the class of the result (QNaN)
I-4.4.1.VE0.3 The actions that will be taken if it is divide operation		
fdiv[s][.]	$FPSCR_{VE}$ = 0 $FPSCR_{VXSNAN}$ = 1, or $FPSCR_{VXIDI}$ = 1, or $FPSCR_{VXZDZ}$ = 1	Target FPR is set to a QNaN $FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is set to indicate the class of the result (QNaN)
I-4.4.1.VE0.4 The actions that will be taken if it is multiplication operation		
fmul[s][.]	$FPSCR_{VE}$ = 0 $FPSCR_{VXSNAN}$ = 1, or $FPSCR_{VXIMZ}$ = 1	Target FPR is set to a QNaN $FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is set to indicate the class of the result (QNaN)
I-4.4.1.VE0.5 The actions that will be taken if it is add/subtract operation		
fadd[s][.] fsub[s][.]	$FPSCR_{VE}$ = 0 $FPSCR_{VXSNAN}$ = 1, or $FPSCR_{VXISI}$ = 1	Target FPR is set to a QNaN $FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is set to indicate the class of the result (QNaN)
I-4.4.1.VE0.6 The actions that will be taken if the operation is floating round to single-precision or reciprocal estimate		
frsp[.] fre[s][.]	$FPSCR_{VE}$ = 0 $FPSCR_{VXSNAN}$ = 1	Target FPR is set to a QNaN $FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is set to indicate the class of the result (QNaN)
I-4.4.1.VE0.7 The actions that will be taken if the operation is an *Floating round to integer instruction*		
Floating round to integer instructions	$FPSCR_{VE}$ = 0 $FPSCR_{VXSNAN}$ = 1	$FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0

Instructions tested	Compliance conditions	Expected result
		$FPSCR_{FPRF}$ is set to indicate the sign and class of the result
I-4.4.1.VE0.8 The actions that will be taken if the operation is *Convert to 64-bit integer instruction* and the source register FRB is a positive number or +∞		
Convert to 64-bit integer instruction	$FPSCR_{VE}$ = 0 $FPSCR_{VXSNAN}$ = 1, or $FPSCR_{VXCVI}$ = 1 FRB is positive number or +∞	FRT is set to the most positive 64-bit integer $FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is undefined
I-4.4.1.VE0.9 The actions that will be taken if the operation is *Convert to 64-bit integer instruction* and the source register FRB is a negative number or -∞, or NaN		
Convert to 64-bit integer instruction	$FPSCR_{VE}$ = 0 $FPSCR_{VXSNAN}$ = 1, or $FPSCR_{VXCVI}$ = 1 FRB is a negative number, -∞, or NaN	FRT is set to the most negative 64-bit integer $FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is undefined
I-4.4.1.VE0.10 The actions that will be taken if the operation is *Convert to 32-bit integer instruction* and the source register FRB is a positive number or +∞		
Convert to 32-bit integer instruction	$FPSCR_{VE}$ = 0 $FPSCR_{VXSNAN}$ = 1, or $FPSCR_{VXCVI}$ = 1 FRB is positive number or +∞	$FRT_{0:31}$ = undefined $FRT_{32:63}$ is set to the most positive 32-bit integer $FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is undefined
I-4.4.1.VE0.11 The actions that will be taken if the operation is *Convert to 32-bit integer instruction* and the source register FRB is a negative number or -∞, or NaN		
Convert to 32-bit integer instruction	$FPSCR_{VE}$ = 0 $FPSCR_{VXSNAN}$ = 1, or $FPSCR_{VXCVI}$ = 1 FRB is a negative number, -∞, or NaN	$FRT_{0:31}$ = undefined $FRT_{32:63}$ is set to the most negative 32-bit integer $FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is undefined
I-4.4.1.VE0.12 The actions that will be taken if the operation is compare ordered		
fcmpo	$FPSCR_{VE}$ = 0 $FPSCR_{VXSNAN}$ =1 or $FPSCR_{VXVC}$ = 1	$FPSCR_{FI}$ is unchanged $FPSCR_{FR}$ is unchanged $FPSCR_{C}$ is unchanged $FPSCR_{FPCC}$ is set to reflect unordered
I-4.4.1.VE0.13 The actions that will be taken if the operation is compare unordered		
Fcmpu	$FPSCR_{VE}$ = 0 $FPSCR_{VXSNAN}$ =1	$FPSCR_{FI}$ is unchanged $FPSCR_{FR}$ is unchanged $FPSCR_{C}$ is unchanged $FPSCR_{FPCC}$ is set to reflect unordered
I-4.4.1.VE0.14 The actions that will be taken if an *mtfsfi, mtfsf,* or *mtfsb1* instruction is executed that sets $FPSCR_{VXSOFT}$ to 1		

Instructions tested	Compliance conditions	Expected result
mtfsfi mtfsf mtfsb1	$FPSCR_{VE}$ = 0 $FPSCR_{VXSOFT}$ = 1	FPSCR is set as specified in the instruction description

5.9.2. Zero Divide Exception

Architecture sections:

- I-4.4.2 Zero Divide Exception

Scenario groups:

- Setting exception bit ZX
- Actions taken when the exception is enabled
- Action taken when the exception is disabled

Condition for Zero Divide Exception for divide instructions: zero divisor value and finite nonzero dividend value.

Condition for Zero Divide Exception for reciprocal estimate instructions: operand value of zero

5.9.2.1. Setting exception bit ZX

Instructions tested	Compliance conditions	Expected result
I-4.4.2.ZeroDiv.1 Divide instructions set ZX to 1 when division by zero occurs		
fdiv[s][.] fre[s][.] frsqrte[s][.]	*Condition for Zero Divide Exception for divide instructions* occurs	ZX = 1
I-4.4.2.ZeroDiv.2 ZX is sticky divide instructions		
Representative of: fdiv[s][.] fre[s][.] frsqrte[s][.]	ZX=1 before instruction execution *Condition for Zero Divide Exception for divide instructions* does **not** occur	ZX = 1
I-4.4.2.ZeroDiv.3 ZX stays zero when exception condition **does not occur** divide instructions		
Representative of: fdiv[s][.] fre[s][.] frsqrte[s][.]	ZX=0 before instruction execution *Condition for Zero Divide Exception for divide instructions* does **not** occur	ZX = 0
I-4.4.2.ZeroDiv.4 Reciprocal estimate instructions set ZX to 1 when operand is zero		
fres[.] frsqrte[.]	*Condition for Zero Divide Exception for reciprocal estimate instructions* occurs	ZX = 1
I-4.4.2.ZeroDiv.5 ZX is sticky reciprocal estimate instructions		
Representative of: fres[.] frsqrte[.]	ZX=1 before instruction execution *Condition for Zero Divide Exception for reciprocal estimate instructions* does **not** occur	ZX = 1
I-4.4.2.ZeroDiv.6 ZX stays zero when exception condition **does not occur** reciprocal estimate instructions		
Representative of: fres[.] frsqrte[.]	ZX=0 before instruction execution *Condition for Zero Divide Exception for reciprocal estimate instructions* does **not** occur	ZX = 0

5.9.2.2. Actions taken when the exception is enabled

Instructions tested	Compliance conditions	Expected result
I-4.4.2.ZE1.1 The actions taken when a Divide instruction or fre[s][.] or frsqrte[s][.] *causes a Zero Divide Exception and the Zero Divide Exception is enabled*		
fdiv[s][.] fre[s][.] or frsqrte[s][.]	$FPSCR_{ZE}$ = 1 $FPSCR_{ZX}$ = 1	The target register FPR is unchanged $FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is unchanged

5.9.2.3. Action taken when the exception is disabled

Instructions tested	Compliance conditions	Expected result
I-4.4.2.ZE0.1 The actions taken when a Divide instruction or fre[s][.] or frsqrte[s][.] *causes a Zero Divide Exception and the Zero Divide Exception is disabled, in case the sign of the XOR between the operands is positive*		
fdiv[s][.] fre[s][.] or frsqrte[s][.]	$FPSCR_{ZE}$ = 0 $FPSCR_{ZX}$ = 1 XOR between the operands is positive	The target register FPR is set to +∞ $FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is set to indicate the class and sign of +∞
I-4.4.2.ZE0.2 The actions taken when a Divide instruction or fre[s][.] or frsqrte[s][. *causes a Zero Divide Exception and the Zero Divide Exception is disabled, in case the sign of the XOR between the operands is negative*		
fdiv[s][.] fre[s][.] or frsqrte[s][.]	$FPSCR_{ZE}$ = 0 $FPSCR_{ZX}$ = 1 XOR between the operands is negative	The target register FPR is set to -∞ $FPSCR_{FI}$ = 0 $FPSCR_{FR}$ = 0 $FPSCR_{FPRF}$ is set to indicate the class and sign of -∞

5.9.3. Overflow Exception

Architecture sections:

- I-4.4.3 Overflow Exception

Scenario groups:

- Setting exception bit OX
- Actions taken when the exception is enabled
- Action taken when the exception is disabled

Condition for Overflow Exception: the magnitude of what would have been the rounded result if the exponent range were unbounded exceeds that of the largest finite number of the specified result precision

5.9.3.1. Setting exception bit OX

Instructions tested	Compliance conditions	Expected result
I-4.4.3.Over.1 OX is set to 1 when overflow occurs		
Floating-point potentially overflowing/underflowing instructions	*Condition for Overflow Exception* occurs	OX = 1

Instructions tested	Compliance conditions	Expected result
I-4.4.3.Over.2 For multiply-add instructions, OX is set based on the final result of the operation, and not on the result of the multiplication		
Floating-point multiply-add instructions	*Condition for Overflow Exception* occurs for the result of the multiplication *Condition for Overflow Exception* **does not occur** for the final result	OX = 0
I-4.4.3.Over.3 OX is sticky		
Representative of *Floating-point potentially overflowing/underflowing instructions*	OX=1 before instruction execution *Condition for Overflow Exception* does **not** occur	OX = 1
I-4.4.3.Over.4 OX stays zero when exception condition **does not occur**		
Representative of *Floating-point potentially overflowing/underflowing instructions*	OX=0 before instruction execution *Condition for Overflow Exception* does **not** occur	OX = 0

5.9.3.2. Actions taken when the exception is enabled

Single-precision arithmetic instructions: fadds[.] fsubs[.] fmuls[.] fdivs[.] fsqrts[.] fres[.] frsqrtes[.] fmadds[.] fmsubs[.] fnmadds[.] fnmsubs[.]

Double-precision arithmetic instructions: fadd[.] fsub[.] fmul[.] fdiv[.] fsqrt[.] fre[.] frsqrte[.] fmadd[.] fmsub[.] fnmadd[.] fnmsub[.]

Instructions tested	Compliance conditions	Expected result
I-4.4.3.OE1.1 The actions to be taken when an overflow exception occurs and the Overflow Exception is enabled for *double-precision arithmetic instructions*		
double-precision arithmetic instructions	$FPSCR_{OX}$ = 1 $FPSCR_{OE}$ = 1	The exponent of the normalized intermediate result is adjusted by subtracting 1536 Target FPR = adjusted rounded result $FPSCR_{FPRF}$ is set to indicate the class and sign of the result (normalized number)
I-4.4.3.OE1.2 The actions to be taken when an overflow exception occurs and the Overflow Exception is enabled for *single-precision arithmetic instructions* and Floating Round to Single-Precision instruction		
single-precision arithmetic instructions frsp[.]	$FPSCR_{OX}$ = 1 $FPSCR_{OE}$ = 1	The exponent of the normalized intermediate result is adjusted by subtracting 192 Target FPR = adjusted rounded result $FPSCR_{FPRF}$ is set to indicate the class and sign of the result (normalized number)

5.9.3.3. Action taken when the exception is disabled

Floating-point potentially overflowing instructions: fadd[s][.] fsub[s][.] fmul[s][.] fdiv[s][.] fsqrt[s][.] fre[s][.] frsqrte[s][.] fmadd[s][.] fmsub[s][.] fnmadd[s][.] fnmsub[s][.] frsp[.]

Instructions tested	Compliance conditions	Expected result
I-4.4.3.OE0.1 The actions to be taken when an overflow exception occurs and the Overflow Exception is disabled in case round mode is Round to Nearest and positive result		
Floating-point potentially overflowing instructions	$FPSCR_{OX}$ = 1 $FPSCR_{OE}$ = 0 Round mode is **Round to Nearest**	$FPSCR_{XX}$ = 1 (FPR) = +∞ $FPSCR_{FR}$ is undefined

Instructions tested	Compliance conditions	Expected result
	Positive overflow	$FPSCR_{FI}$ = 1 $FPSCR_{FPRF}$ is set to indicate +infinity
I-4.4.3.OE0.2 The actions to be taken when an overflow exception occurs and the Overflow Exception is disabled in case round mode is Round to Nearest and negative result		
Floating-point potentially overflowing instructions	$FPSCR_{OX}$ = 1 $FPSCR_{OE}$ = 0 Round mode is **Round to Nearest** **Negative overflow**	$FPSCR_{XX}$ = 1 (FPR) = -∞ $FPSCR_{FR}$ is undefined $FPSCR_{FI}$ = 1 $FPSCR_{FPRF}$ is set to indicate infinity
I-4.4.3.OE0.3 The actions to be taken when an overflow exception occurs and the Overflow Exception is disabled in case round mode is Round toward Zero and positive result		
Floating-point potentially overflowing instructions	$FPSCR_{OX}$ = 1 $FPSCR_{OE}$ = 0 Round mode is **Round toward Zero** **Positive overflow**	$FPSCR_{XX}$ = 1 (FPR) = +(format largest finite number) $FPSCR_{FR}$ is undefined $FPSCR_{FI}$ = 1 $FPSCR_{FPRF}$ is set to indicate +Normal Number
I-4.4.3.OE0.4 The actions to be taken when an overflow exception occurs and the Overflow Exception is disabled in case round mode is Round toward Zero and negative result		
Floating-point potentially overflowing instructions	$FPSCR_{OX}$ = 1 $FPSCR_{OE}$ = 0 Round mode is **Round toward Zero** **Negative overflow**	$FPSCR_{XX}$ = 1 (FPR) = -(format largest finite number) $FPSCR_{FR}$ is undefined $FPSCR_{FI}$ = 1 $FPSCR_{FPRF}$ is set to indicate -Normal Number
I-4.4.3.OE0.5 The actions to be taken when an overflow exception occurs and the Overflow Exception is disabled in case round mode is Round toward + Infinity with negative overflow		
Floating-point potentially overflowing instructions	$FPSCR_{OX}$ = 1 $FPSCR_{OE}$ = 0 Round mode is **Round toward + Infinity** **Negative overflow**	$FPSCR_{XX}$ = 1 (FPR) = formats most negative finite number $FPSCR_{FR}$ is undefined $FPSCR_{FI}$ = 1 $FPSCR_{FPRF}$ is set to indicate -Normal Number
I-4.4.3.OE0.6 The actions to be taken when an overflow exception occurs and the Overflow Exception is disabled in case round mode is Round toward + Infinity with positive overflow		
Floating-point potentially overflowing instructions	$FPSCR_{OX}$ = 1 $FPSCR_{OE}$ = 0 Round mode is **Round toward + Infinity** **Positive overflow**	$FPSCR_{XX}$ = 1 (FPR) = +infinity $FPSCR_{FR}$ is undefined $FPSCR_{FI}$ = 1 $FPSCR_{FPRF}$ is set to indicate +infinity
I-4.4.3.OE0.7 The actions to be taken when an overflow exception occurs and the Overflow Exception is disabled in case round mode is Round toward Infinity with negative overflow		

Instructions tested	Compliance conditions	Expected result
Floating-point potentially overflowing instructions	$FPSCR_{OX}$ = 1 $FPSCR_{OE}$ = 0 Round mode is **Round toward Infinity** **Negative overflow**	$FPSCR_{XX}$ = 1 (FPR) = -infinity $FPSCR_{FR}$ is undefined $FPSCR_{FI}$ = 1 $FPSCR_{FPRF}$ is set to indicate infinity
I-4.4.3.OE0.8 The actions to be taken when an overflow exception occurs and the Overflow Exception is disabled in case round mode is Round toward Infinity with positive overflow		
Floating-point potentially overflowing instructions	$FPSCR_{OX}$ = 1 $FPSCR_{OE}$ = 0 Round mode is **Round toward Infinity** **Positive overflow**	$FPSCR_{XX}$ = 1 (FPR) = formats largest finite number $FPSCR_{FR}$ is undefined $FPSCR_{FI}$ = 1 $FPSCR_{FPRF}$ is set to indicate +Normal Number

5.9.4. Underflow Exception

Architecture sections:

- I-4.4.4 Underflow Exception

Scenario groups:

- Setting exception bit UX
- Actions taken when the exception is enabled
- Action taken when the exception is disabled

Condition for enabled Underflow Exception: intermediate result is Tiny (detected before rounding, when a nonzero intermediate result would be less in magnitude than the smallest normalized number)

Condition for disabled Underflow Exception: intermediate result is Tiny (see above) and there is Loss of Accuracy (the delivered result value differs from the intermediate result)

5.9.4.1. Setting exception bit UX

Instructions tested	Compliance conditions	Expected result
I-4.4.4.Under.1 UX is set to 1 when underflow occurs and Underflow Exception is enabled		
Floating-point potentially overflowing/underflowing instructions	UE = 1 *Condition for enabled Underflow Exception* occurs	UX = 1
I-4.4.4.Under.2 For multiply-add instructions, UX is set based on the final result of the operation, and not on the result of the multiplication with enabled Underflow Exception		
Floating-point multiply-add instructions	UE = 1 *Condition for enabled Underflow Exception* occurs for the result of the multiplication *Condition for enabled Underflow Exception* **does not occur** for the final result	UX = 0

Instructions tested	Compliance conditions	Expected result
I-4.4.4.Under.3 UX is sticky enabled Underflow Exception		
Representative of *Floating-point potentially overflowing/underflowing instructions*	UE = 1 UX=1 before instruction execution *Condition for enabled Underflow Exception* does **not** occur	UX = 1
I-4.4.4.Under.4 UX stays zero when exception condition **does not occur** enabled Underflow Exception		
Representative of *Floating-point potentially overflowing/underflowing instructions*	UE = 1 UX=0 before instruction execution *Condition for enabled Underflow Exception* does **not** occur	UX = 0
I-4.4.4.Under.5 UX is set to 1 when underflow occurs and Underflow Exception is disabled		
Floating-point potentially overflowing/underflowing instructions	UE = 0 *Condition for disabled Underflow Exception* occurs	UX = 1
I-4.4.4.Under.6 For multiply-add instructions, UX is set based on the final result of the operation, and not on the result of the multiplication with disabled Underflow Exception		
Floating-point multiply-add instructions	UE = 0 *Condition for disabled Underflow Exception* occurs for the result of the multiplication *Condition for disabled Underflow Exception* **does not occur** for the final result	UX = 0
I-4.4.4.Under.7 UX is sticky disabled Underflow exception		
Representative of *Floating-point potentially overflowing/underflowing instructions*	UE = 0 UX=1 before instruction execution *Condition for disabled Underflow Exception* does **not** occur	UX = 1
I-4.4.4.Under.8 UX stays zero when exception condition **does not occur** disabled Underflow exception		
Representative of *Floating-point potentially overflowing/underflowing instructions*	UE = 0 UX=0 before instruction execution *Condition for disabled Underflow Exception* does **not** occur	UX = 0
I-4.4.4.Under.9 If Underflow Exception is enabled, UX is set to 1 when result is Tiny, even if no loss of accuracy occurs		
Representative of *Floating-point potentially overflowing/underflowing instructions*	UE = 1 Intermediate result is Tiny The delivered result is the same as the intermediate result	UX = 1

5.9.4.2. Actions taken when the exception is enabled

single-precision arithmetic instructions: fadds[.] fsubs[.] fmuls[.] fdivs[.] fsqrts[.] fres[.] frsqrtes[.] fmadds[.] fmsubs[.] fnmadds[.] fnmsubs[.]

double-precision arithmetic instructions: fadd[.] fsub[.] fmul[.] fdiv[.] fsqrt[.] fre[.] frsqrte[.] fmadd[.] fmsub[.] fnmadd[.] fnmsub[.]

Instructions tested	Compliance conditions	Expected result
I-4.4.4.UE1.1 The actions to be taken when an underflow exception occurs and the underflow Exception is enabled for *double-precision arithmetic instructions*		
double-precision arithmetic instructions	$FPSCR_{UX}$ = 1 $FPSCR_{UE}$ = 1	The exponent of the normalized intermediate result is adjusted by adding 1536 (target FPR) =the adjusted rounded result $FPSCR_{FPRF}$ indicates the class and sign of the result (normalized number)
I-4.4.4.UE1.2 The actions to be taken when an underflow exception occurs and the underflow Exception is enabled for *single-precision arithmetic instructions*		
single-precision arithmetic instructions	$FPSCR_{UX}$ = 1 $FPSCR_{UE}$ = 1	The exponent of the normalized intermediate result is adjusted by adding 192 (target FPR) =the adjusted rounded result $FPSCR_{FPRF}$ indicates the class and sign of the result (normalized number)
I-4.4.4.UE1.3 The actions to be taken when an underflow exception occurs and the underflow Exception is enabled for round to single-precision instruction		
frsp[.]	$FPSCR_{UX}$ = 1 $FPSCR_{UE}$ = 1	(target FPR) =the adjusted rounded result $FPSCR_{FPRF}$ indicates the class and sign of the result (normalized number)

5.9.4.3. Action taken when the exception is disabled

Floating-point potentially underflowing instructions: fadd[s][.] fsub[s][.] fmul[s][.] fdiv[s][.] fsqrt[s][.] fre[s][.] frsqrte[s][.] fmadd[s][.] fmsub[s][.] fnmadd[s][.] fnmsub[s][.] frsp[.]

Instructions tested	Compliance conditions	Expected result
I-4.4.4.UE0.1 The actions to be taken when an underflow exception occurs and the underflow Exception is disabled		
Floating-point potentially underflowing instructions	$FPSCR_{UX}$ = 1 $FPSCR_{UE}$ = 0	(target FPR) = rounded result $FPSCR_{FPRF}$ indicates the class and sign of the result

5.9.5. Inexact Exception

Architecture sections:

- I-4.4.5 Inexact Exception

Scenario groups:

- Setting exception bit XX
- Actions taken when Inexact Exception occurs

Condition for Inexact Exception due to rounding:

- The rounded result differs from the intermediate result
- No enabled Overflow or Underflow exception occurs

Condition for Inexact Exception due to overflow:

- The rounded result overflows
- Overflow Exception is disabled

5.9.5.1. Setting exception bit XX

Instructions tested	Compliance conditions	Expected result
I-4.4.5.Inexact.1 XX is set to 1 when rounded result differs from intermediate result		
Floating-point potentially inexact instructions	*Condition for Inexact Exception due to rounding* occurs	XX = 1
I-4.4.5.Inexact.2 XX is set to 1 when rounded result overflows and Overflow Exception is disabled		
Floating-point potentially inexact instructions	*Condition for Inexact Exception due to overflow* occurs	XX = 1
I-4.4.5.Inexact.3 XX is sticky		
Representative of *Floating-point potentially inexact instructions*	XX=1 before instruction execution *Condition for Inexact Exception due to rounding* does **not** occur *Condition for Inexact Exception due to overflow* does **not** occur	XX = 1
I-4.4.5.Inexact.4 XX stays zero when exception condition **does not occur**		
Representative of *Floating-point potentially inexact instructions*	XX=0 before instruction execution *Condition for Inexact Exception due to rounding* does **not** occur *Condition for Inexact Exception due to overflow* does **not** occur	XX = 0

5.9.5.2. Actions taken when Inexact Exception occurs

Floating-point potentially inexact instructions: fadd[s][.] fsub[s][.] fmul[s][.] fdiv[s][.] fsqrt[s][.] frsp[.] fcfid[.] fmadd[s][.] fmsub[s][.] fnmadd[s][.] fnmsub[s][.] fctid[.] fctidz[.] fctiw[.] fctiwz[.] fcfidu[.] fcfids[.] fcfidus[.] fctidu[.] fctiduz[.] fctiwu[.] fctiwuz[.]

Instructions tested	Compliance conditions	Expected result
I-4.4.5.XX.1 The actions to be taken when an inexact exception occurs		
Floating-point potentially inexact instructions	$FPSCR_{XX}$ = 1	(target FPR) = rounded or overflowed result $FPSCR_{FPRF}$ indicates the class and sign of the result

5.9.6. Combinations of exceptions

Architecture sections:

- I-4.4 Floating-Point Exceptions

Scenario groups:

- Cases where two exceptions can occur
- Setting Inexact Exception when enabled Overflow/Underflow Exception occurs

5.9.6.1. Cases where two exceptions can occur

Instructions tested	Compliance conditions	Expected result
I-4.4.Combine.1 Multiply-Add instructions may set both Invalid Operation Exception (SNaN) and Invalid Operation Exception (∞0)		
Representative of *Floating-point multiply-add instructions*	*Condition for Invalid Operation Exception (SNaN)* occurs	VXSNAN = 1

Instructions tested	Compliance conditions	Expected result
	Condition for Invalid Operation Exception (∞ 0) occurs	VXIMZ = 1
I-4.4.Combine.2 Compare Ordered instructions may set both Invalid Operation Exception (SNaN) and Invalid Operation Exception (Invalid Compare)		
fcmpo	*Condition for Invalid Operation Exception (SNaN)* occurs *Condition for Invalid Operation Exception (Invalid Compare)* occurs	VXSNAN = 1 VXVC = 1
I-4.4.Combine.3 Convert to Integer instructions may set both Invalid Operation Exception (SNaN) and Invalid Operation Exception (Invalid Integer Convert)		
Representative of *Floating-point conversion-to-integer instructions*	*Condition for Invalid Operation Exception (SNaN)* occurs *Condition for Invalid Operation Exception (Invalid Integer Convert)* occurs	VXSNAN = 1 VXCVI = 1

5.9.6.2. Setting Inexact Exception when enabled Overflow/Underflow Exception occurs

Floating-point potentially inexact and overflowing/underflowing instructions: the intersection of *Floating-point potentially inexact instructions* and *Floating-point potentially overflowing/underflowing instructions*

Instructions tested	Compliance conditions	Expected result
I-4.4.CombineInexact.1 Inexact Exception with enabled Overflow Exception, when rounding changes the significand		
Representative of *Floating-point potentially inexact and overflowing/underflowing instructions*	Significands of the rounded and intermediate results differ *Condition for Overflow Exception* occurs OE=1	XX=1
I-4.4.CombineInexact.2 Inexact Exception with enabled Overflow Exception, when rounding does not change the significand		
Representative of *Floating-point potentially inexact and overflowing/underflowing instructions*	The rounded and intermediate results differ The significands of the rounded and intermediate results are equal *Condition for Overflow Exception* occurs OE=1	XX=0
I-4.4.CombineInexact.3 Inexact Exception with enabled Underflow Exception, when rounding changes the significand		
Representative of *Floating-point potentially inexact and overflowing/underflowing instructions*	Significands of the rounded and intermediate results differ *Condition for enabled Underflow Exception* occurs UE=1	XX=1
I-4.4.CombineInexact.4 Inexact Exception with enabled Underflow Exception, when rounding does not change the significand		
Representative of *Floating-point potentially inexact and overflowing/underflowing instructions*	The rounded and intermediate results differ The significands of the rounded and intermediate results are equal *Condition for enabled Underflow Exception* occurs	XX=0

Instructions tested	Compliance conditions	Expected result
	UE=1	

5.9.7. Setting the exception summary bits

Architecture sections:

- I-4.2.2 Floating-Point Status and Control Register

Instructions tested	Compliance conditions	Expected result
I-4.2.2.ExSum.1 FX is set to 1 if any exception occurs		
Representative floating-point instruction	Any exception occurs (one or more)	FX=1
I-4.2.2.ExSum.2 FX is sticky		
Representative floating-point instruction	FX = 1 before instruction execution No exception occurs	FX=1
I-4.2.2.ExSum.3 FX stays zero when no exceptions occur		
Representative floating-point instruction	FX = 0 before instruction execution No exception occurs	FX=0
I-4.2.2.ExSum.4 FEX is set to 1 if any enabled exception occurs		
Representative floating-point instruction	Any enabled exception occurs (one or more)	FEX=1
I-4.2.2.ExSum.5 FEX is not sticky		
Representative floating-point instruction	FEX = 1 before instruction execution No enabled exception occurs	FEX=0
I-4.2.2.ExSum.6 FEX stays zero when no enabled exceptions occur		
Representative floating-point instruction	FEX = 0 before instruction execution No enabled exception occurs Any disabled exception occurs (one or more)	FEX=0
I-4.2.2.ExSum.7 VX is set to 1 if any Invalid Operation exception occurs		
Representative floating-point instruction	Any Invalid Operation exception occurs (one or more)	VX=1
I-4.2.2.ExSum.8 VX is not sticky		
Representative floating-point instruction	VX = 1 before instruction execution No Invalid Operation exception occurs	VX=0
I-4.2.2.ExSum.9 VX stays zero when no Invalid Operation exception occurs		
Representative floating-point instruction	VX = 0 before instruction execution No Invalid Operation exception occurs	VX=0

5.9.8. Floating-point exception modes

Architecture sections:

- I-4.4 Floating-Point Exceptions

Instructions tested	Compliance conditions	Expected result
I-4.4.ExMode.1 Floating-point exception mode Ignore Exceptions		

Instructions tested	Compliance conditions	Expected result
Representative floating-point instruction	MSR_{FE0} = 0 MSR_{FE1} = 0 Some enabled floating-point exception occurs	Ignore Exceptions Mode: the system floating-point enabled error handler is not invoked
I-4.4.ExMode.2 Floating-point exception mode Imprecise Nonrecoverable Mode		
Representative floating-point instruction	MSR_{FE0} = 0 MSR_{FE1} = 1 Some enabled floating-point exception occurs	Imprecise Nonrecoverable Mode: The system floating-point enabled exception error handler is invoked at some point at or beyond the instruction that caused the enabled exception, in nonrecoverable mode
I-4.4.ExMode.3 Floating-point exception mode Imprecise Recoverable Mode		
Representative floating-point instruction	MSR_{FE0} = 1 MSR_{FE1} = 0 Some enabled floating-point exception occurs	Imprecise Recoverable Mode: the system floating-point enabled exception error handler is invoked at some point at or beyond the instruction that caused the enabled exception, in recoverable mode
I-4.4.ExMode.4 Floating-point exception mode Precise Mode		
Representative floating-point instruction	MSR_{FE0} = 1 MSR_{FE1} = 1 Some enabled floating-point exception occurs	Precise Mode: the system floating-point enabled exception error handler is invoked precisely at the instruction that caused the enabled exception.

5.10. Floating-Point Data

Architecture sections:

- I-4.3 Floating-Point Data

Scenario groups:

- QNaN propagation
- Single Precision

5.10.1. NaN propagation

Architecture sections:

- I-4.2.2 Floating-Point Status and Control Register
- I-4.3.2 Value Representation

Instructions tested	Compliance conditions	Expected result
I-4.3.2.NaN.1 Propagation of NaN in FRA		
Representative floating-point instruction with an FRA operand specified	(FRA) is a NaN	(FRT) = (FRA)
I-4.3.2.NaN.2 Propagation of NaN in FRB		
Representative floating-point instruction with an FRB operand specified, other than frsp	(FRA) is not a NaN, or no FRA operand is specified	(FRT) = (FRB)

Instructions tested	Compliance conditions	Expected result
	(FRB) is a NaN	
I-4.3.2.NaN.3 Propagation of NaN in FRB for frsp		
frsp	(FRB) is a NaN	(FRT) = $(FRB)_{0:34} \mid\mid {}^{29}0$
I-4.3.2.NaN.4 Propagation of NaN in FRC		
Representative floating-point instruction with an FRC operand specified	(FRA) is not a NaN, or no FRA operand is specified (FRB) is not a NaN, or no FRB operand is specified (FRC) is a NaN	(FRT) = (FRC)
I-4.3.2.NaN.5 Propagation of Nan in FRC		
Representative floating-point instruction with an FRC operand specified	(FRA) is not a NaN, or no FRA operand is specified (FRB) is not a NaN, or no FRB operand is specified (FRC) is not a NaN The operation generates a QNaN	(FRT) = 0x7FF8_0000_0000_0000

5.10.2. Single Precision

Architecture sections:

- I-4.3.5 Data Handling and Precision

Instructions tested	Compliance conditions	Expected result
I-4.3.5.Single.1 When the result of a *Load Floating-Point Single*, *Floating Round to Single-Precision*, or *single-precision arithmetic instruction* is stored in an FPR, the low-order 29 FRACTION bits are zero.		
Load Floating-Point Single *Floating Round to Single-Precision* *single-precision arithmetic instructions*	Single-Precision result	low-order 29 FRACTION bits are zero.

6. Decimal Floating-Point (Chapter I.5)

Table of Contents

6.1. DFP Exceptions

Architecture sections:

- I-5.5.10 DFP Exceptions

6.1.1. Invalid Operation Exception

Architecture sections:

- I-5.5.10.1 Invalid Operation Exception

Scenario groups:

- Setting exception bits
- Keeping exception bits unchanged
- Actions taken when the exception is enabled
- Action taken when the exception is disabled

Condition for Invalid Operation Exception **SNaN**: any DFP operation on a signaling NaN (SNaN), except for Test, Round to DFP Short, Convert to DFP Long, Decode DPD to BCD, Extract Biased Exponent, Insert Biased Exponent, Shift Significand Left Immediate, and Shift Significand Right Immediate.

Condition for Invalid Operation Exception **InfinityInfinity**: magnitude subtraction of infinities occurs.

Condition for Invalid Operation Exception **InfinityInfinity**: division of infinity by infinity occurs.

Condition for Invalid Operation Exception **ZeroZero**: division of zero by zero occurs.

Condition for Invalid Operation Exception **Infinity Zero**: multiplication of infinity by zero occurs.

Condition for Invalid Operation Exception **Invalid Compare**: ordered comparison involving a NaN.

Condition for Invalid Operation Exception **Invalid Conversion**: one of the following:

- The Quantize operation detects that the significand associated with the specified target exponent would have more significant digits than the target-format precision
- For the Quantize operation, when one source operand specifies an infinity and the other specifies a finite number

- The Reround operation detects that the target exponent associated with the specified target significance would be greater than Xmax
- The Encode BCD To DPD operation detects an invalid BCD digit or sign code
- The Convert to Fixed operation involving a number too large in magnitude to be represented in the target format, or involving a NaN.

6.1.1.1. Setting exception bits

DFP potential-SNaN-operand instructions: dadd[.] daddq[.] dsub[.] dsubq[.] dmul[.] dmulq[.] ddiv[.] ddivq[.] dcmpu dcmpuq dcmpo dcmpoq dquai[.] dquaiq[.] dqua[.] dquaq[.] drrnd[.] drrndq[.] drintx[.] drintxq[.] drintn[.] drintnq[.] dctqpq[.] drdpq[.] dctfix[.] dctfixq[.]

DFP addition/subtraction instructions: dadd[.] daddq[.] dsub[.] dsubq[.]

DFP potentially causing Invalid Conversion exception: dquai[.] dquaiq[.] dqua[.] dquaq[.] drrnd[.] drrndq[.] dctfix[.] dctfixq[.] denbcd[.] denbcdq[.]

Instructions tested	Compliance conditions	Expected result
I-5.5.10.1.Invalid.1 VXSNAN is set to 1 when one of the operands is an SNaN		
DFP potential-SNaN-operand instructions	*Condition for Invalid Operation Exception* ***SNaN*** occurs	VXSNAN = 1
I-5.5.10.1.Invalid.2 Add/subtract operations set VXISI to 1 when magnitude subtraction of infinities occurs		
DFP addition/subtraction instructions	*Condition for Invalid Operation Exception* ***InfinityInfinity*** occurs	VXISI = 1
I-5.5.10.1.Invalid.3 Division operations set VXIDI to 1 when division of infinity by infinity occurs		
ddiv[.] ddivq[.]	*Condition for Invalid Operation Exception* ***InfinityInfinity*** occurs	VXIDI = 1
I-5.5.10.1.Invalid.4 Division operations set VXZDZ to 1 when division of zero by zero occurs		
ddiv[.] ddivq[.]	*Condition for Invalid Operation Exception* ***ZeroZero*** occurs	VXZDZ = 1
I-5.5.10.1.Invalid.5 Multiplication operations set VXIMZ to 1 when multiplication of infinity by zero occurs		
dmul[.] dmulq[.]	*Condition for Invalid Operation Exception* ***Infinity Zero*** occurs	VXIMZ = 1
I-5.5.10.1.Invalid.6 Ordered comparison sets VXVC to 1 when the comparison involves a NaN		
dcmpo dcmpoq	*Condition for Invalid Operation Exception* ***Invalid Compare*** occurs	VXVC = 1
I-5.5.10.1.Invalid.7 Quantize, Reround, Encode BCD To DPD and Convert to Fixed operations set VXCVI to 1 when *Condition for Invalid Operation Exception* ***Invalid Conversion*** occurs		
DFP potentially causing Invalid Conversion exception	*Condition for Invalid Operation Exception* ***Invalid Conversion*** occurs	VXCVI = 1
I-5.5.10.1.Invalid.8 VX is set to 1 if any Invalid Operation Exception occurs		
Representative of the instructions tested in I-5.5.10.1.Invalid.1 to I-5.5.10.1.Invalid.7	VXSNAN | VXISI | VXIDI | VXZDZ | VXIMZ | VXVC | VXCVI = 1	VX = 1
I-5.5.10.1.Invalid.9 VX is not sticky		
Representative of the instructions tested in I-5.5.10.1.Invalid.1 to I-5.5.10.1.Invalid.7	VX = 1 before instruction execution No Invalid Operation exception occurs	VX=0
I-5.5.10.1.Invalid.10 VX stays zero when no Invalid Operation exception occurs		
Representative of the instructions tested in I-5.5.10.1.Invalid.1 to I-5.5.10.1.Invalid.7	VX = 0 before instruction execution No Invalid Operation exception occurs	VX=0

6.1.1.2. Keeping exception bits unchanged

Instructions tested	Compliance conditions	Expected result
I-5.5.10.1.InvalidConst.1 VXSNAN is sticky		
Representative of *DFP potential-SNaN-operand instructions*	VXSNAN=1 before instruction execution *Condition for Invalid Operation Exception* ***SNaN*** occurs does **not** occur.	VXSNAN = 1
I-5.5.10.1.InvalidConst.2 VXSNAN stays zero when exception condition **does not occur**		
Representative of *DFP potential-SNaN-operand instructions*	VXSNAN=0 before instruction execution *Condition for Invalid Operation Exception* ***SNaN*** occurs does **not** occur.	VXSNAN = 0
I-5.5.10.1.InvalidConst.3 VXISI is sticky		
Representative of *DFP addition/subtraction instructions*	VXISI = 1 before instruction execution *Condition for Invalid Operation Exception* ***Infinity-Infinity*** does **not** occur	VXISI = 1
I-5.5.10.1.InvalidConst.4 VXISI stays zero when exception condition **does not occur**		
Representative of *DFP addition/subtraction instructions*	VXISI = 0 before instruction execution *Condition for Invalid Operation Exception* ***Infinity-Infinity*** does **not** occur	VXISI = 0
I-5.5.10.1.InvalidConst.5 VXIDI is sticky		
Representative of ddiv[.] ddivq[.]	VXIDI = 1 before instruction execution *Condition for Invalid Operation Exception* ***Infinity÷Infinity*** does **not** occur	VXIDI = 1
I-5.5.10.1.InvalidConst.6 VXIDI stays zero when exception condition **does not occur**		
Representative of ddiv[.] ddivq[.]	VXIDI = 0 before instruction execution *Condition for Invalid Operation Exception* ***Infinity÷Infinity*** does **not** occur	VXIDI = 0
I-5.5.10.1.InvalidConst.7 VXZDZ is sticky		
Representative of ddiv[.] ddivq[.]	VXZDZ = 1 before instruction execution *Condition for Invalid Operation Exception* ***Zero÷Zero*** does **not** occur	VXZDZ = 1
I-5.5.10.1.InvalidConst.8 VXZDZ stays zero when exception condition **does not occur**		
Representative of ddiv[.] ddivq[.]	VXZDZ = 0 before instruction execution *Condition for Invalid Operation Exception* ***Zero÷Zero*** does **not** occur	VXZDZ = 0
I-5.5.10.1.InvalidConst.9 VXIMZ is sticky		
Representative of dmul[.] dmulq[.]	VXIMZ = 1 before instruction execution *Condition for Invalid Operation Exception* ***Infinity÷Zero*** does **not** occur	VXIMZ = 1
I-5.5.10.1.InvalidConst.10 VXIMZ stays zero when exception condition **does not occur**		
Representative of dmul[.] dmulq[.]	VXIMZ = 0 before instruction execution *Condition for Invalid Operation Exception* ***Infinity÷Zero*** does **not** occur	VXIMZ = 0
I-5.5.10.1.InvalidConst.11 VXVC is sticky		
Representative of dcmpo dcmpoq	VXVC = 1 before instruction execution *Condition for Invalid Operation Exception* ***Invalid Compare*** does **not** occur	VXVC = 1

Instructions tested	Compliance conditions	Expected result
I-5.5.10.1.InvalidConst.12 VXVC stays zero when exception condition **does not occur**		
Representative of dcmpo dcmpoq	VXVC = 0 before instruction execution *Condition for Invalid Operation Exception* ***Invalid Compare*** does **not** occur	VXVC = 0
I-5.5.10.1.InvalidConst.13 VXCVI is sticky		
Representative of *DFP potentially causing Invalid Conversion exception*	VXCVI = 1 before instruction execution *Condition for Invalid Operation Exception* ***Invalid Integer Convert*** does **not** occur	VXCVI = 1
I-5.5.10.1.InvalidConst.14 VXCVI stays zero when exception condition **does not occur**		
Representative of *DFP potentially causing Invalid Conversion exception*	VXCVI = 0 before instruction execution *Condition for Invalid Operation Exception* ***Invalid Integer Convert*** does **not** occur	VXCVI = 0

6.1.1.3. Actions taken when the exception is enabled

Instructions tested	Compliance conditions	Expected result
I-5.5.10.1.VE1.1 The actions that will be taken when the Invalid Operation Exception is enabled for arithmetic, quantum adjustment, conversion or format potentially causing an exception instructions		
dadd[.] daddq[.] dsub[.] dsubq[.] dmul[.] dmulq[.] ddiv[.] ddivq[.] dquai[.] dquaiq[.] dqua[.] dquaq[.] drrnd[.] drrndq[.] drintx[.] drintxq[.] drintn[.] drintnq[.] dctqpq[.] drdpq[.] dctfix[.] dctfixq[.] denbcd[.] denbcdq[.]	VE = 1 VXSNAN | VXISI | VXIDI | VXZDZ | VXIMZ | VXCVI = 1	The target FPR is unchanged FR = 0 FI = 0 FPRF is unchanged
I-5.5.10.1.VE1.2 The actions that will be taken when the Invalid Operation Exception is enabled for compare potentially causing an exception instructions		
dcmpu dcmpuq dcmpo dcmpoq	VE = 1 VXSNAN | VXVC = 1	FR is unchanged FI is unchanged C is unchanged FPCC is set to reflect unordered

6.1.1.4. Action taken when the exception is disabled

Instructions tested	Compliance conditions	Expected result
I-5.5.10.1.VE0.1 The actions that will be taken when the Invalid Operation Exception is disabled for arithmetic, quantum-adjustment, Round to DFP Long, Convert to DFP Extended, or format instructions		
dadd[.] daddq[.] dsub[.] dsubq[.] dmul[.] dmulq[.] ddiv[.] ddivq[.] dquai[.] dquaiq[.] dqua[.] dquaq[.] drrnd[.] drrndq[.] drintx[.] drintxq[.] drintn[.] drintnq[.] dctqpq[.] drdpq[.] denbcd[.] denbcdq[.]	VE = 0 VXSNAN | VXISI | VXIDI | VXZDZ | VXIMZ | VXCVI = 1	The target FPR is set to QNaN FR = 0 FI = 0 FPRF is set to indicate the class of QNaN
I-5.5.10.1.VE0.2 The actions that will be taken when the Invalid Operation Exception is disabled for Convert to Fixed instructions and the operand in FRB is a positive number or +Infinity		

Instructions tested	Compliance conditions	Expected result
dctfix[.] dctfixq[.]	VE = 0 VXSNAN | VXCVI = 1	Target FRT is set to the most positive 64-bit binary integer FR = 0 FI = 0 FPRF is unchanged
I-5.5.10.1.VE0.3 The actions that will be taken when the Invalid Operation Exception is disabled for Convert to Fixed instructions and the operand in FRB is a negative number, - Infinity or NaN		
dctfix[.] dctfixq[.]	VE = 0 VXSNAN | VXCVI = 1	Target FRT is set to the most negative 64-bit binary integer FR = 0 FI = 0 FPRF is unchanged
I-5.5.10.1.VE0.4 The actions that will be taken when the Invalid Operation Exception is disabled for compare potentially causing an exception instructions		
dcmpu dcmpuq dcmpo dcmpoq	VE = 0 VXSNAN | VXVC = 1	FR is unchanged FI is unchanged C is unchanged FPCC is set to reflect unordered

6.1.2. Zero Divide Exception

Architecture sections:

- I-5.5.10.2 Zero Divide Exception

Scenario groups:

- Setting exception bit ZX
- Actions taken when the exception is enabled
- Action taken when the exception is disabled

Condition for Zero Divide Exception for divide instructions: zero divisor value and a finite nonzero dividend value.

6.1.2.1. Setting exception bit ZX

Instructions tested	Compliance conditions	Expected result
I-5.5.10.2.ZeroDiv.1 DFP Divide instructions set ZX to 1 when division by zero occurs		
ddiv[.] ddivq[.]	*Condition for Zero Divide Exception for divide instructions* occurs	ZX = 1
I-5.5.10.2.ZeroDiv.2 ZX is sticky divide instructions		
Representative of: ddiv[.] ddivq[.]	ZX=1 before instruction execution *Condition for Zero Divide Exception for divide instructions* does **not** occur	ZX = 1
I-5.5.10.2.ZeroDiv.3 ZX stays zero when exception condition **does not occur** divide instructions		
Representative of: ddiv[.] ddivq[.]	ZX=0 before instruction execution *Condition for Zero Divide Exception for divide instructions* does **not** occur	ZX = 0

6.1.2.2. Actions taken when the exception is enabled

Instructions tested	Compliance conditions	Expected result
I-5.5.10.2.ZE1.1 The actions that will be taken when the Zero Divide Exception is enabled		
ddiv[.] ddivq[.]	ZE = 1 ZX = 1	Target FPR is unchanged FR = 0 FI = 0 FPRF is unchanged

6.1.2.3. Action taken when the exception is disabled

Instructions tested	Compliance conditions	Expected result
I-5.5.10.2.ZE0.1 The actions that will be taken when the Zero Divide Exception is disabled if the XOR of the signs of the operands is Zero		
ddiv[.] ddivq[.]	ZE = 0 ZX = 1 XOR of the signs of the operands = 0	Target FPR is set to +Infinity FR = 0 FI = 0 FPRF is set to indicate +Infinity
I-5.5.10.2.ZE0.2 The actions that will be taken when the Zero Divide Exception is disabled if the XOR of the signs of the operands is One		
ddiv[.] ddivq[.]	ZE = 0 ZX = 1 XOR of the signs of the operands = 1	Target FPR is set to -Infinity FR = 0 FI = 0 FPRF is set to indicate -Infinity

6.1.3. Overflow Exception

Architecture sections:

- I-5.5.10.3 Overflow Exception

Scenario groups:

- Setting exception bit OX
- Actions taken when the exception is enabled
- Action taken when the exception is disabled

Condition for Overflow Exception: the target formats largest finite number is exceeded in magnitude by what would have been the rounded result if the exponent range were unbounded.

6.1.3.1. Setting exception bit OX

DFP potentially overflowing/ underflowing instructions: dadd[.] daddq[.] dsub[.] dsubq[.] dmul[.] dmulq[.] ddiv[.] ddivq[.] drsp[.] drdpq[.]

Instructions tested	Compliance conditions	Expected result
I-5.5.10.3.Over.1 OX is set to 1 when overflow occurs		
DFP potentially overflowing/ underflowing instructions	*Condition for Overflow Exception* occurs	OX = 1
I-5.5.10.3.Over.2 OX is sticky		

Instructions tested	Compliance conditions	Expected result
Representative of *DFP potentially overflowing/ underflowing instructions*	OX=1 before instruction execution *Condition for Overflow Exception* does **not** occur	OX = 1
I-5.5.10.3.Over.3 OX stays zero when exception condition **does not occur**		
Representative of *DFP potentially overflowing/ underflowing instructions*	OX=0 before instruction execution *Condition for Overflow Exception* does **not** occur	OX = 0

6.1.3.2. Actions taken when the exception is enabled

Notes:

Wrapped result: the result after the exponent adjustment is subtracted from the infinitely precise results exponent.

Wrapped rounded result: the wrapped result rounded to the target-format precision.

Instructions tested	Compliance conditions	Expected result
I-5.5.10.3.OE1.1 The actions that will be taken when the Overflow Exception is enabled for the arithmetic non-quad instructions if the wrapped rounded result has only one form		
dadd[.] dsub[.] dmul[.] ddiv[.]	OE = 1 OX = 1 Wrapped rounded result has only one form	Wrapped result = the result with subtracting 576 from the exponent Target result = the wrapped rounded result FPRF is set to indicate ±Normal Number
I-5.5.10.3.OE1.2 The actions that will be taken when the Overflow Exception is enabled for the arithmetic non-quad instructions if the wrapped rounded result has redundant forms and is exact		
dadd[.] dsub[.] dmul[.] ddiv[.]	OE = 1 OX = 1 Wrapped rounded result has redundant forms and is exact	Wrapped result = the result with subtracting 576 from the exponent Target result = the wrapped rounded result of the form that has exponent closest to the wrapped ideal exponent FPRF is set to indicate ±Normal Number
I-5.5.10.3.OE1.3 The actions that will be taken when the Overflow Exception is enabled for the arithmetic non-quad instructions if the wrapped rounded result has redundant forms and is inexact		
dadd[.] dsub[.] dmul[.] ddiv[.]	OE = 1 OX = 1 Wrapped rounded result has redundant forms and is inexact	Wrapped result = the result with subtracting 576 from the exponent Target result = the wrapped rounded result of the form that has the smallest exponent FPRF is set to indicate ±Normal Number
I-5.5.10.3.OE1.4 The actions that will be taken when the Overflow Exception is enabled for the arithmetic quad instructions if the wrapped rounded result has only one form		
daddq[.] dsubq[.] dmulq[.] ddivq[.]	OE = 1 OX = 1 Wrapped rounded result has only one form	Wrapped result = the result with subtracting 9216 from the exponent Target result = the wrapped rounded result FPRF is set to indicate ±Normal Number
I-5.5.10.3.OE1.5 The actions that will be taken when the Overflow Exception is enabled for the arithmetic quad instructions if the wrapped rounded result has redundant forms and is exact		
daddq[.] dsubq[.] dmulq[.] ddivq[.]	OE = 1 OX = 1	Wrapped result = the result with subtracting 9216 from the exponent

Instructions tested	Compliance conditions	Expected result
	Wrapped rounded result has redundant forms and is exact	Target result = the wrapped rounded result of the form that has exponent closest to the wrapped ideal exponent FPRF is set to indicate ±Normal Number
I-5.5.10.3.OE1.6 The actions that will be taken when the Overflow Exception is enabled for the arithmetic quad instructions if the wrapped rounded result has redundant forms and is inexact		
daddq[.] dsubq[.] dmulq[.] ddivq[.]	OE = 1 OX = 1 Wrapped rounded result has redundant forms and is inexact	Wrapped result = the result with subtracting 9216 from the exponent Target result = the wrapped rounded result of the form that has the smallest exponent FPRF is set to indicate ±Normal Number
I-5.5.10.3.OE1.7 The actions that will be taken when the Overflow Exception is enabled for the Round to DFP Short instruction if the wrapped rounded result has only one form		
drsp[.]	OE = 1 OX = 1 Wrapped rounded result has only one form	Wrapped result = the result with subtracting 192 from the exponent Target result = the wrapped rounded result FPRF is set to indicate ±Normal Number
I-5.5.10.3.OE1.8 The actions that will be taken when the Overflow Exception is enabled for the Round to DFP Short instruction if the wrapped rounded result has redundant forms and is exact		
drsp[.]	OE = 1 OX = 1 Wrapped rounded result has redundant forms and is exact	Wrapped result = the result with subtracting 192 from the exponent Target result = the wrapped rounded result of the form that has exponent closest to the wrapped ideal exponent FPRF is set to indicate ±Normal Number
I-5.5.10.3.OE1.9 The actions that will be taken when the Overflow Exception is enabled for the Round to DFP Short instruction if the wrapped rounded result has redundant forms and is inexact		
drsp[.]	OE = 1 OX = 1 Wrapped rounded result has redundant forms and is inexact	Wrapped result = the result with subtracting 192 from the exponent Target result = the wrapped rounded result of the form that has the smallest exponent FPRF is set to indicate ±Normal Number
I-5.5.10.3.OE1.10 The actions that will be taken when the Overflow Exception is enabled for the Round to DFP Long instruction if the wrapped rounded result has only one form		
drdpq[.]	OE = 1 OX = 1 Wrapped rounded result has only one form	Wrapped result = the result with subtracting 3072 from the exponent Target result = the wrapped rounded result FPRF is set to indicate ±Normal Number
I-5.5.10.3.OE1.11 The actions that will be taken when the Overflow Exception is enabled for the Round to DFP Long instruction if the wrapped rounded result has redundant forms and is exact		
drdpq[.]	OE = 1 OX = 1 Wrapped rounded result has redundant forms and is exact	Wrapped result = the result with subtracting 3072 from the exponent Target result = the wrapped rounded result of the form that has exponent closest to the wrapped ideal exponent FPRF is set to indicate ±Normal Number
I-5.5.10.3.OE1.12 The actions that will be taken when the Overflow Exception is enabled for the Round to DFP Long instruction if the wrapped rounded result has redundant forms and is inexact		

Instructions tested	Compliance conditions	Expected result
drdpq[.]	OE = 1 OX = 1 Wrapped rounded result has redundant forms and is inexact	Wrapped result = the result with subtracting 3072 from the exponent Target result = the wrapped rounded result of the form that has the smallest exponent FPRF is set to indicate ±Normal Number

6.1.3.3. Action taken when the exception is disabled

Instructions tested	Compliance conditions	Expected result
I-5.5.10.3.OE0.1 The actions that will be taken when the Overflow Exception is disabled for the *DFP potentially overflowing/ underflowing instructions* with Round to Nearest, Ties to Even rounding mode and positive intermediate result		
DFP potentially overflowing/ underflowing instructions	OE = 0 OX = 1 round to Nearest, Ties to Even rounding mode positive intermediate result	XX = 1 Target FPR = +Infinity FR = 1 FI = 1 FPRF is set to indicate +Infinity
I-5.5.10.3.OE0.2 The actions that will be taken when the Overflow Exception is disabled for the *DFP potentially overflowing/ underflowing instructions* with Round to Nearest, Ties to Even rounding mode and negative intermediate result		
DFP potentially overflowing/ underflowing instructions	OE = 0 OX = 1 round to Nearest, Ties to Even rounding mode negative intermediate result	XX = 1 Target FPR = -Infinity FR = 1 FI = 1 FPRF is set to indicate -Infinity
I-5.5.10.3.OE0.3 The actions that will be taken when the Overflow Exception is disabled for the *DFP potentially overflowing/ underflowing instructions* with Round toward 0 rounding mode and positive intermediate result		
DFP potentially overflowing/ underflowing instructions	OE = 0 OX = 1 Round toward 0 rounding mode positive intermediate result	XX = 1 Target FPR = $+N_{max}$ FR = 0 FI = 1 FPRF is set to indicate +Normal Number
I-5.5.10.3.OE0.4 The actions that will be taken when the Overflow Exception is disabled for the *DFP potentially overflowing/ underflowing instructions* with Round toward 0 rounding mode and negative intermediate result		
DFP potentially overflowing/ underflowing instructions	OE = 0 OX = 1 Round toward 0 rounding mode negative intermediate result	XX = 1 Target FPR = $-N_{max}$ FR = 0 FI = 1 FPRF is set to indicate -Normal Number
I-5.5.10.3.OE0.5 The actions that will be taken when the Overflow Exception is disabled for the *DFP potentially overflowing/ underflowing instructions* with Round toward +Infinity rounding mode and positive intermediate result		
DFP potentially overflowing/ underflowing instructions	OE = 0 OX = 1 Round toward +Infinity rounding mode	XX = 1 Target FPR = +Infinity FR = 1

Instructions tested	Compliance conditions	Expected result
	positive intermediate result	FI = 1 FPRF is set to indicate +Infinity
I-5.5.10.3.OE0.6 The actions that will be taken when the Overflow Exception is disabled for the *DFP potentially overflowing/ underflowing instructions* with Round toward +Infinity rounding mode and negative intermediate result		
DFP potentially overflowing/ underflowing instructions	OE = 0 OX = 1 Round toward +Infinity rounding mode negative intermediate result	XX = 1 Target FPR = $-N_{max}$ FR = 0 FI = 1 FPRF is set to indicate -Normal Number
I-5.5.10.3.OE0.7 The actions that will be taken when the Overflow Exception is disabled for the *DFP potentially overflowing/ underflowing instructions* with Round toward -Infinity rounding mode and positive intermediate result		
DFP potentially overflowing/ underflowing instructions	OE = 0 OX = 1 Round toward -Infinity rounding mode positive intermediate result	XX = 1 Target FPR = $+N_{max}$ FR = 0 FI = 1 FPRF is set to indicate +Normal Number
I-5.5.10.3.OE0.8 The actions that will be taken when the Overflow Exception is disabled for the *DFP potentially overflowing/ underflowing instructions* with Round toward -Infinity rounding mode and negative intermediate result		
DFP potentially overflowing/ underflowing instructions	OE = 0 OX = 1 Round toward -Infinity rounding mode negative intermediate result	XX = 1 Target FPR = -Infinity FR = 1 FI = 1 FPRF is set to indicate -Infinity
I-5.5.10.3.OE0.9 The actions that will be taken when the Overflow Exception is disabled for the *DFP potentially overflowing/ underflowing instructions* with Round to Nearest, Ties away from 0 rounding mode and positive intermediate result		
DFP potentially overflowing/ underflowing instructions	OE = 0 OX = 1 Round to Nearest, Ties away from 0 rounding mode positive intermediate result	XX = 1 Target FPR = +Infinity FR = 1 FI = 1 FPRF is set to indicate +Infinity
I-5.5.10.3.OE0.10 The actions that will be taken when the Overflow Exception is disabled for the *DFP potentially overflowing/ underflowing instructions* with Round to Nearest, Ties away from 0 rounding mode and negative intermediate result		
DFP potentially overflowing/ underflowing instructions	OE = 0 OX = 1 Round to Nearest, Ties away from 0 rounding mode negative intermediate result	XX = 1 Target FPR = -Infinity FR = 1 FI = 1 FPRF is set to indicate -Infinity
I-5.5.10.3.OE0.11 The actions that will be taken when the Overflow Exception is disabled for the *DFP potentially overflowing/ underflowing instructions* with Round to Nearest, Ties toward 0 rounding mode and positive intermediate result		
DFP potentially overflowing/ underflowing instructions	OE = 0 OX = 1	XX = 1 Target FPR = +Infinity

Instructions tested	Compliance conditions	Expected result
	Round to Nearest, Ties toward 0 rounding mode positive intermediate result	FR = 1 FI = 1 FPRF is set to indicate +Infinity
I-5.5.10.3.OE0.12 The actions that will be taken when the Overflow Exception is disabled for the *DFP potentially overflowing/ underflowing instructions* with Round to Nearest, Ties toward 0 rounding mode and negative intermediate result		
DFP potentially overflowing/ underflowing instructions	OE = 0 OX = 1 Round to Nearest, Ties toward 0 rounding mode negative intermediate result	XX = 1 Target FPR = -Infinity FR = 1 FI = 1 FPRF is set to indicate -Infinity
I-5.5.10.3.OE0.13 The actions that will be taken when the Overflow Exception is disabled for the *DFP potentially overflowing/ underflowing instructions* with Round away from 0 rounding mode and positive intermediate result		
DFP potentially overflowing/ underflowing instructions	OE = 0 OX = 1 Round away from 0 rounding mode positive intermediate result	XX = 1 Target FPR = +Infinity FR = 1 FI = 1 FPRF is set to indicate +Infinity
I-5.5.10.3.OE0.14 The actions that will be taken when the Overflow Exception is disabled for the *DFP potentially overflowing/ underflowing instructions* with Round away from 0 rounding mode and negative intermediate result		
DFP potentially overflowing/ underflowing instructions	OE = 0 OX = 1 Round away from 0 rounding mode negative intermediate result	XX = 1 Target FPR = -Infinity FR = 1 FI = 1 FPRF is set to indicate -Infinity
I-5.5.10.3.OE0.15 The actions that will be taken when the Overflow Exception is disabled for the *DFP potentially overflowing/ underflowing instructions* with Round to prepare for shorter precision rounding mode and positive intermediate result		
DFP potentially overflowing/ underflowing instructions	OE = 0 OX = 1 Round to prepare for shorter precision rounding mode positive intermediate result	XX = 1 Target FPR = $+N_{max}$ FR = 0 FI = 1 FPRF is set to indicate +Normal Number
I-5.5.10.3.OE0.16 The actions that will be taken when the Overflow Exception is disabled for the *DFP potentially overflowing/ underflowing instructions* with Round to prepare for shorter precision rounding mode and negative intermediate result		
DFP potentially overflowing/ underflowing instructions	OE = 0 OX = 1 Round to prepare for shorter precision rounding mode negative intermediate result	XX = 1 Target FPR = $-N_{max}$ FR = 0 FI = 1 FPRF is set to indicate -Normal Number

6.1.4. Underflow Exception

Architecture sections:

- I-5.5.10.4 Underflow Exception

Scenario groups:

- Setting exception bit UX
- Actions taken when the exception is enabled
- Action taken when the exception is disabled

Condition for enabled Underflow Exception: Tininess condition is recognized (tininess conditionis recognized in both states when a result computed as though both the precision and exponent range were unbounded would be nonzero and less than the target formats smallest normal number, N_{min} , in magnitude.)

Condition for disabled Underflow Exception: When the tininess condition is recognized (see above) and when the delivered result value differs from what would have been computed were both the precision and the exponent range unbounded.

6.1.4.1. Setting exception bit UX

Instructions tested	**Compliance conditions**	**Expected result**
I-5.5.10.4.Under.1 UX is set to 1 when underflow occurs and Underflow Exception is enabled		
DFP potentially overflowing/ underflowing instructions	UE = 1 *Condition for enabled Underflow Exception* occurs	UX = 1
I-5.5.10.4.Under.2 UX is sticky enabled Underflow Exception		
Representative of *DFP potentially overflowing/ underflowing instructions*	UE = 1 UX=1 before instruction execution *Condition for enabled Underflow Exception* does **not** occur	UX = 1
I-5.5.10.4.Under.3 UX stays zero when exception condition **does not occur** enabled Underflow Exception		
Representative of *DFP potentially overflowing/ underflowing instructions*	UE = 1 UX=0 before instruction execution *Condition for enabled Underflow Exception* does **not** occur	UX = 0
I-5.5.10.4.Under.4 UX is set to 1 when underflow occurs and Underflow Exception is disabled		
DFP potentially overflowing/ underflowing instructions	UE = 0 *Condition for disabled Underflow Exception* occurs	UX = 1
I-5.5.10.4.Under.5 UX is sticky disabled Underflow exception		
Representative of *DFP potentially overflowing/ underflowing instructions*	UE = 0 UX=1 before instruction execution *Condition for disabled Underflow Exception* does **not** occur	UX = 1
I-5.5.10.4.Under.6 UX stays zero when exception condition **does not occur** disabled Underflow exception		
Representative of *DFP potentially overflowing/ underflowing instructions*	UE = 0	UX = 0

Instructions tested	Compliance conditions	Expected result
	UX=0 before instruction execution *Condition for disabled Underflow Exception* does **not** occur	
I-5.5.10.4.Under.7 If Underflow Exception is enabled, UX is set to 1 when result is Tiny, even if no loss of accuracy occurs		
Representative of *DFP potentially overflowing/ underflowing instructions*	UE = 1 Intermediate result is Tiny The delivered result is the same as the intermediate result	UX = 1

6.1.4.2. Actions taken when the exception is enabled

Notes:

- Wrapped result: the result after the exponent adjustment is added to the infinitely precise results exponent.
- Wrapped rounded result: the wrapped result rounded to the target-format precision.

Instructions tested	Compliance conditions	Expected result
I-5.5.10.4.UE1.1 The actions that will be taken when the Underflow Exception is enabled for the arithmetic non-quad instructions if the wrapped rounded result has only one form		
dadd[.] dsub[.] dmul[.] ddiv[.]	UE = 1 UX = 1 Wrapped rounded result has only one form	Wrapped result = the result with adding 576 to the exponent Target result = the wrapped rounded result FPRF is set to indicate ±Normal Number
I-5.5.10.4.UE1.2 The actions that will be taken when the Underflow Exception is enabled for the arithmetic non-quad instructions if the wrapped rounded result has redundant forms and is exact		
dadd[.] dsub[.] dmul[.] ddiv[.]	UE = 1 UX = 1 Wrapped rounded result has redundant forms and is exact	Wrapped result = the result with adding 576 to the exponent Target result = the wrapped rounded result of the form that has exponent closest to the wrapped ideal exponent FPRF is set to indicate ±Normal Number
I-5.5.10.4.UE1.3 The actions that will be taken when the Underflow Exception is enabled for the arithmetic non-quad instructions if the wrapped rounded result has redundant forms and is inexact		
dadd[.] dsub[.] dmul[.] ddiv[.]	UE = 1 UX = 1 Wrapped rounded result has redundant forms and is inexact	Wrapped result = the result with adding 576 to the exponent Target result = the wrapped rounded result of the form that has the smallest exponent FPRF is set to indicate ±Normal Number
I-5.5.10.4.UE1.4 The actions that will be taken when the Underflow Exception is enabled for the arithmetic quad instructions if the wrapped rounded result has only one form		
daddq[.] dsubq[.] dmulq[.] ddivq[.]	UE = 1 UX = 1 Wrapped rounded result has only one form	Wrapped result = the result with adding 9216 to the exponent Target result = the wrapped rounded result FPRF is set to indicate ±Normal Number
I-5.5.10.4.UE1.5 The actions that will be taken when the Underflow Exception is enabled for the arithmetic quad instructions if the wrapped rounded result has redundant forms and is exact		
daddq[.] dsubq[.] dmulq[.] ddivq[.]	UE = 1	Wrapped result = the result with adding 9216 to the exponent

Instructions tested	Compliance conditions	Expected result
	UX = 1 Wrapped rounded result has redundant forms and is exact	Target result = the wrapped rounded result of the form that has exponent closest to the wrapped ideal exponent FPRF is set to indicate ±Normal Number
I-5.5.10.4.UE1.6 The actions that will be taken when the Underflow Exception is enabled for the arithmetic quad instructions if the wrapped rounded result has redundant forms and is inexact		
daddq[.] dsubq[.] dmulq[.] ddivq[.]	UE = 1 UX = 1 Wrapped rounded result has redundant forms and is inexact	Wrapped result = the result with adding 9216 to the exponent Target result = the wrapped rounded result of the form that has the smallest exponent FPRF is set to indicate ±Normal Number
I-5.5.10.4.UE1.7 The actions that will be taken when the Underflow Exception is enabled for the Round to DFP Short instruction if the wrapped rounded result has only one form		
drsp[.]	UE = 1 UX = 1 Wrapped rounded result has only one form	Wrapped result = the result with adding 192 to the exponent Target result = the wrapped rounded result FPRF is set to indicate ±Normal Number
I-5.5.10.4.UE1.8 The actions that will be taken when the Underflow Exception is enabled for the Round to DFP Short instruction if the wrapped rounded result has redundant forms and is exact		
drsp[.]	UE = 1 UX = 1 Wrapped rounded result has redundant forms and is exact	Wrapped result = the result with adding 192 to the exponent Target result = the wrapped rounded result of the form that has exponent closest to the wrapped ideal exponent FPRF is set to indicate ±Normal Number
I-5.5.10.4.UE1.9 The actions that will be taken when the Underflow Exception is enabled for the Round to DFP Short instruction if the wrapped rounded result has redundant forms and is inexact		
drsp[.]	UE = 1 UX = 1 Wrapped rounded result has redundant forms and is inexact	Wrapped result = the result with adding 192 to the exponent Target result = the wrapped rounded result of the form that has the smallest exponent FPRF is set to indicate ±Normal Number
I-5.5.10.4.UE1.10 The actions that will be taken when the Underflow Exception is enabled for the Round to DFP Long instruction if the wrapped rounded result has only one form		
drdpq[.]	UE = 1 UX = 1 Wrapped rounded result has only one form	Wrapped result = the result with adding 3072 to the exponent Target result = the wrapped rounded result FPRF is set to indicate ±Normal Number
I-5.5.10.4.UE1.11 The actions that will be taken when the Underflow Exception is enabled for the Round to DFP Long instruction if the wrapped rounded result has redundant forms and is exact		
drdpq[.]	UE = 1 UX = 1 Wrapped rounded result has redundant forms and is exact	Wrapped result = the result with adding 3072 to the exponent Target result = the wrapped rounded result of the form that has exponent closest to the wrapped ideal exponent FPRF is set to indicate ±Normal Number
I-5.5.10.4.UE1.12 The actions that will be taken when the Underflow Exception is enabled for the Round to DFP Long instruction if the wrapped rounded result has redundant forms and is inexact		

Instructions tested	Compliance conditions	Expected result
drdpq[.]	UE = 1 UX = 1 Wrapped rounded result has redundant forms and is inexact	Wrapped result = the result with adding 3072 to the exponent Target result = the wrapped rounded result of the form that has the smallest exponent FPRF is set to indicate ±Normal Number

6.1.4.3. Action taken when the exception is disabled

Note:

Rounded result: the infinitely precise result is rounded to the target-format precision.

Instructions tested	Compliance conditions	Expected result
I-5.5.10.4.UE0.1 The actions that will be taken when the Underflow Exception is disabled for the *DFP potentially overflowing/ underflowing instructions* when the *rounded result* has only one form		
DFP potentially overflowing/ underflowing instructions	UE = 0 UX = 1 *rounded result* has only one form	Target result = *rounded result* FPRF is set to indicate the sign and class of the target result
I-5.5.10.4.UE0.2 The actions that will be taken when the Underflow Exception is disabled for the *DFP potentially overflowing/ underflowing instructions* when the *rounded result* has redundant formats		
DFP potentially overflowing/ underflowing instructions	UE = 0 UX = 1 *rounded result* has redundant formats	Target result = the *rounded result* of the form that is closest to the ideal exponent. FPRF is set to indicate the sign and class of the target result

6.1.5. Inexact Exception

Architecture sections:

- I-5.5.10.5 Inexact Exception

Scenario groups:

- Setting exception bit XX
- Actions taken when Inexact Exception occurs

Condition for Inexact Exception due to unbounding during rounding:

The delivered result differs from what would have been computed were both the precision and exponent range unbounded.

Condition for Inexact Exception due to overflow:

The rounded result overflows

Overflow Exception is disabled

6.1.5.1. Setting exception bit XX

DFP potentially inexact instructions: dadd[.] daddq[.] dsub[.] dsubq[.] dmul[.] dmulq[.] ddiv[.] ddivq[.] dquai[.] dquaiq[.] dqua[.] dquaq[.] drrnd[.] drrndq[.] drintx[.] drintxq[.] drsp[.] drdpq[.] dcffix[.] dctfix[.] dctfixq[.]

Instructions tested	Compliance conditions	Expected result
I-5.5.10.5.Inexact.1 XX is set to 1 when rounded result differs from what would have been computed were both the precision and exponent range unbounded		
DFP potentially inexact instructions	*Condition for Inexact Exception due to unbounding during rounding*	XX = 1
I-5.5.10.5.Inexact.2 XX is set to 1 when rounded result overflows and Overflow Exception is disabled		
DFP potentially inexact instructions	*Condition for Inexact Exception due to overflow* occurs	XX = 1
I-5.5.10.5.Inexact.3 XX is sticky		
Representative of *DFP potentially inexact instructions*	XX=1 before instruction execution *Condition for Inexact Exception due to unbounding during rounding* does **not** occur *Condition for Inexact Exception due to overflow* does **not** occur	XX = 1
I-5.5.10.5.Inexact.4 XX stays zero when exception condition **does not occur**		
Representative of *DFP potentially inexact instructions*	XX=0 before instruction execution *Condition for Inexact Exception due to unbounding during rounding* does **not** occur *Condition for Inexact Exception due to overflow* does **not** occur	XX = 0

6.1.5.2. Actions taken when Inexact Exception occurs

Instructions tested	Compliance conditions	Expected result
I-5.5.10.5.XX.1 The actions to be taken when an Inexact exception occurs		
DFP potentially inexact instructions	XX = 1	(target FPR) = rounded or overflowed result FPRF indicates the class and sign of the result

6.1.6. Combinations of exceptions

Architecture sections:

- I-5.5.10 DFP Exceptions

Scenario groups:

- Cases where two exceptions can occur

6.1.6.1. Cases where two exceptions can occur

Instructions tested	Compliance conditions	Expected result
I-5.5.10.Combine.1 Inexact Exception may be set with Overflow Exception.		
Representative of *DFP potentially overflowing/ underflowing instructions*	Condition for Overflow Exception *Condition for Inexact Exception due to overflow* occurs	OX = 1 XX = 1
I-5.5.10.Combine.2 Inexact Exception may be set with Underflow Exception.		
Representative of *DFP potentially overflowing/ underflowing instructions*	*Condition for enabled Underflow Exception or Condition for disabled Underflow Exception*	UX = 1 XX = 1

Instructions tested	Compliance conditions	Expected result
	Condition for Inexact Exception due to unbounding during rounding	
I-5.5.10.Combine.3 Invalid Operation Exception (SNaN) may be set with Invalid Operation Exception (Invalid Compare) for *Compare Ordered* instructions		
Representative of dcmpo dcmpoq	*Condition for Invalid Operation Exception SNaN* *Condition for Invalid Operation Exception Invalid Compare*	VXSNAN = 1 VXVC = 1
I-5.5.10.Combine.4 Invalid Operation Exception (SNaN) may be set with Invalid Operation Exception (Invalid Conversion) for Convert To Fixed instructions.		
Representative of dctfix[.] dctfixq[.]	*Condition for Invalid Operation Exception SNaN* *The Convert to Fixed operation involving a number too large in magnitude to be represented in the target format, or involving a NaN.*	VXSNAN = 1 VXCVI = 1

6.1.7. Setting the exception summary bits

Architecture sections:

- I-5.2.1 DFP Usage of Floating-Point Registers
- I-7.2.2 Floating-Point Status and Control Register

Instructions tested	Compliance conditions	Expected result
I-7.2.2.ExSum.1 FX is set to 1 if any exception occurs		
Representative DFP instructions	Any exception occurs (one or more)	FX=1
I-7.2.2.ExSum.2 FX is sticky		
Representative DFP instructions	FX = 1 before instruction execution No exception occurs	FX=1
I-7.2.2.ExSum.3 FX stays zero when no exceptions occur		
Representative DFP instructions	FX = 0 before instruction execution No exception occurs	FX=0
I-7.2.2.ExSum.4 FEX is set to 1 if any enabled exception occurs		
Representative DFP instructions	Any enabled exception occurs (one or more)	FEX=1
I-7.2.2.ExSum.5 FEX is not sticky		
Representative DFP instructions	FEX = 1 before instruction execution No enabled exception occurs	FEX=0
I-7.2.2.ExSum.6 FEX stays zero when no enabled exceptions occur		
Representative DFP instructions	FEX = 0 before instruction execution No enabled exception occurs Any disabled exception occurs (one or more)	FEX=0

6.1.8. Floating-point exception modes

Architecture sections:

- I-5.5.10 DFP Exceptions

Instructions tested	Compliance conditions	Expected result
I-5.5.10.ExMode.1 Floating-point exception mode Ignore Exceptions		
Representative DFP instruction	MSR_{FE0} = 0 MSR_{FE1} = 0 Some enabled DFP exception occurs	Ignore Exceptions Mode: the system floating-point enabled error handler is not invoked
I-5.5.10.ExMode.2 Floating-point exception mode Imprecise Nonrecoverable Mode		
Representative DFP instruction	MSR_{FE0} = 0 MSR_{FE1} = 1 Some enabled DFP exception occurs	Imprecise Nonrecoverable Mode: The system floating-point enabled exception error handler is invoked at some point at or beyond the instruction that caused the enabled exception, in nonrecoverable mode
I-5.5.10.ExMode.3 Floating-point exception mode Imprecise Recoverable Mode		
Representative DFP instruction	MSR_{FE0} = 1 MSR_{FE1} = 0 Some enabled DFP exception occurs	Imprecise Recoverable Mode: the system floating-point enabled exception error handler is invoked at some point at or beyond the instruction that caused the enabled exception, in recoverable mode
I-5.5.10.ExMode.4 Floating-point exception mode Precise Mode		
Representative DFP instruction	MSR_{FE0} = 1 MSR_{FE1} = 1 Some enabled DFP exception occurs	Precise Mode: the system floating-point enabled exception error handler is invoked precisely at the instruction that caused the enabled exception.

6.2. DFP Arithmetic, Quantum Adjustment and Conversion Instructions

Architecture sections:

- I- 5.6.1 DFP Arithmetic Instructions
- I- 5.6.4 DFP Quantum Adjustment Instructions
- I- 5.6.5 DFP Conversion Instructions

Scenario groups:

- Setting FR (fraction rounded) and FI (fraction inexact)
- Setting FPRF (result flags)
- Setting CR1
- Correct computations
- Rounding modes

DFP Arithmetic Instructions: dadd[.] daddq[.] dsub[.] dsubq[.] dmul[.] dmulq[.] ddiv[.] ddivq[.]

DFP Quantum Adjustment Instructions: dquai[.] dquaiq[.] dqua[.] dquaq[.] drrnd[.] drrndq[.] drintx[.] drintxq[.] drintn[.] drintnq[.]

DFP Conversion Instructions:

- DFP Data-Format Conversion Instructions: dctdp[.] dctqpq[.] drsp[.] drdpq[.]

- DFP Data-Type Conversion Instructions: dcffix[.] dcffixq[.] dctfix[.]

6.2.1. Setting FR (fraction rounded) and FI (fraction inexact)

DFP potentially rounding instructions: *DFP Arithmetic Instructions*, dquai[.] dquaiq[.] dqua[.] dquaq[.] drrnd[.] drrndq[.] drintx[.] drintxq[.] drsp[.] drdpq[.] dcffix[.] dctfix[.]

The following instructions are not included in the list of *potentially rounding instructions*, since they set FR and FI to an undefined value: dctdp[.] dcffixq[.]

Instructions tested	Compliance conditions	Expected result
I-5.6.1.FR.1 FR is set to 1 during rounding if the rounded result is greater in magnitude than the intermediate result		
DFP potentially rounding instructions	The rounded result is greater in magnitude than the intermediate result No exception occurs	FR=1
I-5.6.1.FR.2 Some instructions always set FR to zero		
drintn[.] drintnq[.] dctqpq[.]	The rounded result is greater in magnitude than the intermediate result No exception occurs	FR=0
I-5.6.1.FR.3 FR is not sticky		
DFP potentially rounding instructions	The rounded result is not greater in magnitude than the intermediate result No exception occurs	FR=0
I-5.6.1.FR.4 FI is set to 1 when an inexact exception occurs		
DFP potentially rounding instructions	An Inexact Exception occurs (XX = 1)	FI=1
I-5.6.1.FR.5 Some instructions always set FI to zero		
drintn[.] drintnq[.] dctqpq[.]	No exception other than Inexact Exception occurs	FI=0
I-5.6.1.FR.6 FI is not sticky		
DFP potentially rounding instructions	No exception occurs	FI=0

6.2.2. Setting FPRF (result flags)

DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions: *DFP Arithmetic Instructions, DFP Quantum Adjustment Instructions, DFP Data-Format Conversion Instructions* and dcffix[.] dcffixq[.]

Note:

- The tests in this section assume no exceptions occurrences

- dctfix[.] is not included in the *DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions* since it sets FPRF to an undefined value

Instructions tested	Compliance conditions	Expected result
I-5.6.1.FPRF.1 *DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions* set FPRF correctly for Signaling NaN result		
DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions	Result value class is Signaling NaN	FPRF = 00001
I-5.6.1.FPRF.2 *DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions* set FPRF correctly for Quiet NaN result		

Instructions tested	Compliance conditions	Expected result
DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions	Result value class is Quiet NaN	FPRF = 10001
I-5.6.1.FPRF.3 *DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions* set FPRF correctly for Infinity result		
DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions	Result value class is Infinity	FPRF = 01001
I-5.6.1.FPRF.4 *DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions* set correctly for Normalized result		
DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions	Result value class is Normalized Number	FPRF = 01000
I-5.6.1.FPRF.5 *DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions* set FPRF correctly for Subnormal Number result		
DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions	Result value class is Subnormal Number	FPRF = 11000
I-5.6.1.FPRF.6 *DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions* set FPRF correctly for Zero result		
DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions	Result value class is Zero	FPRF = 10010
I-5.6.1.FPRF. *7 DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions* set FPRF correctly for +Zero result		
DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions	Result value class is +Zero	FPRF = 00010
I-5.6.1.FPRF.8 *DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions* set FPRF correctly for +Subnormal Number result		
DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions	Result value class is +Subnormal Number	FPRF = 10100
I-5.6.1.FPRF.9 *DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions* set FPRF correclyt for +Normalized result		
DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions	Result value class is +Normalized Number	FPRF = 00100
I-5.6.1.FPRF.10 *DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions* set FPRF correctly for +Infinity result		
DFP Arithmetic, Quantum Adjustment and Conversion setting FPRF instructions	Result value class is +Infinity	FPRF = 00101

6.2.3. Setting CR1

DFP Arithmetic, Quantum Adjustment and Conversion recording instructions: dadd. daddq. dsub. dsubq. dmul. dmulq. ddiv. ddivq. dquai. dquaiq. dqua. dquaq. drrnd. drrndq. drintx. drintxq. drintn. drintnq. dctdp. dctqpq. drsp. drdpq. dcffix. dcffixq. dctfix.

Instructions tested	Compliance conditions	Expected result
I-5.6.1.CR1.1 *DFP Arithmetic, Quantum Adjustment and Conversion recording instructions*, set CR1 field (bits 36:39 of the CR) to the FP exception status, copied from bits 32:35 of the FPSCR.		
DFP Arithmetic, Quantum Adjustment and Conversion recording instructions	Rc = 1	CR1 = $FPSCR_{32:35}$
I-5.6.1.CR1.2 Non-recording instructions dont change the CR1 field		
DFP Arithmetic, Quantum Adjustment and Conversion recording instructions	Rc = 0	CR1 unchanged

6.2.4. Correct Computations

Instructions tested	Compliance conditions	Expected result
I-5.6.1.Comp. 1 DFP Arithmetic, Quantum Adjustment and Conversion Instructions perform correct computations		

Instructions tested	Compliance conditions	Expected result
DFP Arithmetic Instructions *DFP Quantum Adjustment Instructions* *DFP Conversion Instructions*	No exception occurs	Correct result of operation

6.2.5. Rounding modes

Instructions tested	Compliance conditions	Expected result
I-5.6.1.Round.1 Round to Nearest, Ties to Even		
Representative of *DFP Arithmetic Instructions* *DFP Quantum Adjustment Instructions* *DFP Conversion Instructions*	DRN = 000 No exception occurs Result is numeric Numeric operands	Rounding is performed according to Round to Nearest, Ties to Even mode
I-5.6.1.Round.2 Round toward 0 mode		
Representative of *DFP Arithmetic Instructions* *DFP Quantum Adjustment Instructions* *DFP Conversion Instructions*	DRN = 001 No exception occurs Result is numeric Numeric operands	Rounding is performed according to Round toward 0 mode
I-5.6.1.Round.3 Round toward +Infinity mode		
Representative of *DFP Arithmetic Instructions* *DFP Quantum Adjustment Instructions* *DFP Conversion Instructions*	DRN = 010 No exception occurs Result is numeric Numeric operands	Rounding is performed according to Round toward +Infinity mode
I-5.6.1.Round.4 Round toward Infinity mode		
Representative of *DFP Arithmetic Instructions* *DFP Quantum Adjustment Instructions* *DFP Conversion Instructions*	DRN = 011 No exception occurs Result is numeric Numeric operands	Rounding is performed according to Round toward Infinity mode
I-5.6.1.Round.5 Round to Nearest, Ties away from 0 mode		
Representative of *DFP Arithmetic Instructions* *DFP Quantum Adjustment Instructions* *DFP Conversion Instructions*	DRN = 100 No exception occurs Result is numeric Numeric operands	Rounding is performed according to Round to Nearest, Ties away from 0 mode
I-5.6.1.Round.6 Round to Nearest, Ties toward 0 mode		
Representative of *DFP Arithmetic Instructions* *DFP Quantum Adjustment Instructions* *DFP Conversion Instructions*	DRN = 101 No exception occurs Result is numeric Numeric operands	Rounding is performed according to Round to Nearest, Ties toward 0 mode
I-5.6.1.Round.7 Round away from 0 mode		
Representative of *DFP Arithmetic Instructions*	DRN = 110 No exception occurs	Rounding is performed according to Round away from 0 mode

Instructions tested	Compliance conditions	Expected result
DFP Quantum Adjustment Instructions *DFP Conversion Instructions*	Result is numeric Numeric operands	
I-5.6.1.Round.8 Round to Prepare for Shorter Precision mode		
Representative of *DFP Arithmetic Instructions* *DFP Quantum Adjustment Instructions* *DFP Conversion Instructions*	DRN = 111 No exception occurs Result is numeric Numeric operands	Rounding is performed according to Round to Prepare for Shorter Precision mode

6.3. DFP Compare Instructions

Architecture sections:

- I- 5.6.2 DFP Compare Instructions

Scenario groups:

- Setting CR field BF and FPCC

6.3.1. Setting CR field BF and FPCC

Instructions tested	Compliance conditions	Expected result
I-5.6.2.Compare.1 Comparison result is Less Than		
dcmpu dcmpuq dcmpo dcmpoq	(FRA[p]) and (FRB[p]) are not NaNs (FRA[p]) < (FRB[p])	FPCC = 1000 $CR_{4xBF:4xBF+3}$ = 1000
I-5.6.2.Compare.2 Comparison result is Greater Than		
dcmpu dcmpuq dcmpo dcmpoq	(FRA[p]) and (FRB[p]) are not NaNs (FRA[p]) > (FRB[p])	FPCC = 0100 $CR_{4xBF:4xBF+3}$ = 0100
I-5.6.2.Compare.3 Comparison result is Equals		
dcmpu dcmpuq dcmpo dcmpoq	(FRA[p]) and (FRB[p]) are not NaNs (FRA[p]) = (FRB[p])	FPCC = 0010 $CR_{4xBF:4xBF+3}$ = 0010
I-5.6.2.Compare.4 Ordered Comparison when at least one operand is NaN (SNaN or QNaN)		
dcmpo dcmpoq	(FRA[p]) = NaN or (FRB[p]) = NaN	FPCC = 0001 $CR_{4xBF:4xBF+3}$ = 0001
I-5.6.2.Compare.5 Unordered Comparison when at least one operand is QNaN and the other differs from SNaN		
dcmpu dcmpuq	(FRA[p]) and (FRB[p]) are not SNaNs (FRA[p]) = QNaN or (FRB[p]) = QNaN	FPCC = 0001 $CR_{4xBF:4xBF+3}$ = 0001
I-5.6.2.Compare.6 Unordered Comparison when at least one operand is SNaN		
dcmpu dcmpuq	(FRA[p]) = SNaN or (FRB[p]) = SNaN	FPCC = unchanged $CR_{4xBF:4xBF+3}$ = unchanged

6.4. DFP Test Instructions

Architecture sections:

- I- 5.6.3 DFP Test Instructions

Scenario groups:

- Setting CR field BF and FPCC

6.4.1. Setting CR field BF and FPCC

Instructions tested	Compliance conditions	Expected result
I-5.6.3.Test.1 The DFP test data class instructions set FPCC and CR field BF to indicate positive operand in FRA[p] and the data class of FRA[p] doesnt match the data class specified by DCM		
dtstdc dtstdcq	FRA[p] is positive with no match to DCM	FPCC = 0000 $CR_{4xBF:4xBF+3}$ = 0000
I-5.6.3.Test.2 The DFP test data class instructions set FPCC and CR field BF to indicate positive operand in FRA[p] and the data class of FRA[p] matches the data class specified by DCM		
dtstdc dtstdcq	FRA[p] is positive with match to DCM	FPCC = 0010 $CR_{4xBF:4xBF+3}$ = 0010
I-5.6.3.Test.3 The DFP test data class instructions set FPCC and CR field BF to indicate negative operand in FRA[p] and the data class of FRA[p] doesnt match the data class specified by DCM		
dtstdc dtstdcq	FRA[p] is negative with no match to DCM	FPCC = 1000 $CR_{4xBF:4xBF+3}$ = 1000
I-5.6.3.Test.4 The DFP test data class instructions set FPCC and CR field BF to indicate negative operand in FRA[p] and the data class of FRA[p] matches the data class specified by DCM		
dtstdc dtstdcq	FRA[p] is negative with match to DCM	FPCC = 1010 $CR_{4xBF:4xBF+3}$ = 1010
I-5.6.3.Test.5 The DFP test data group instructions set FPCC and CR field BF to indicate positive operand in FRA[p] and the data group of FRA[p] doesnt match the data class specified by DGM		
dtstdg dtstdgq	FRA[p] is positive with no match to DGM	FPCC = 0000 $CR_{4xBF:4xBF+3}$ = 0000
I-5.6.3.Test.6 The DFP test data group instructions set FPCC and CR field BF to indicate positive operand in FRA[p] and the data group of FRA[p] matches the data class specified by DGM		
dtstdg dtstdgq	FRA[p] is positive with match to DGM	FPCC = 0010 $CR_{4xBF:4xBF+3}$ = 0010
I-5.6.3.Test.7 The DFP test data group instructions set FPCC and CR field BF to indicate negative operand in FRA[p] and the data group of FRA[p] doesnt match the data class specified by DGM		
dtstdg dtstdgq	FRA[p] is negative with no match to DGM	FPCC = 1000 $CR_{4xBF:4xBF+3}$ = 1000
I-5.6.3.Test.8 The DFP test data group instructions set FPCC and CR field BF to indicate negative operand in FRA[p] and the data group of FRA[p] matches the data class specified by DGM		
dtstdg dtstdgq	FRA[p] is negative with match to DGM	FPCC = 1010 $CR_{4xBF:4xBF+3}$ = 1010
I-5.6.3.Test.9 DFP Test Exponent instructions set the FPCC and CR field BF when the comparison result of the exponents of the operands is less than		
dtstex dtstexq	FRA[p] and FRB[p] are finite numbers (including zeros) Exponent of FRA[p] is less than the exponent of FRB[p]	FPCC = 1000 $CR_{4xBF:4xBF+3}$ = 1000
I-5.6.3.Test.10 DFP Test Exponent instructions set the FPCC and CR field BF when the comparison result of the exponents of the operands is greater than		
dtstex dtstexq	FRA[p] and FRB[p] are finite numbers (including zeros)	FPCC = 0100

Instructions tested	Compliance conditions	Expected result
	Exponent of FRA[p] is greater than the exponent of FRB[p]	$CR_{4xBF:4xBF+3}$ = 0100
I-5.6.3.Test.11 DFP Test Exponent instructions set the FPCC and CR field BF when the comparison result of the exponents of the finite numbers operands is equal		
dtstex dtstexq	FRA[p] and FRB[p] are finite numbers (including zeros) Exponent of FRA[p] equals to the exponent of FRB[p]	FPCC = 0010 $CR_{4xBF:4xBF+3}$ = 0010
I-5.6.3.Test.12 DFP Test Exponent instructions set the FPCC and CR field BF when the operands are equal but are not finite numbers		
dtstex dtstexq	FRA[p] = FRB[p] FRA[p] and FRB[p] are infinities or NaNs	FPCC = 0010 $CR_{4xBF:4xBF+3}$ = 0010
I-5.6.3.Test.13 DFP Test Exponent instructions set the FPCC and CR field BF when the operands are different but not finite numbers		
dtstex dtstexq	FRA[p] != FRB[p] FRA[p] and FRB[p] are infinities or NaNs	FPCC = 0001 $CR_{4xBF:4xBF+3}$ = 0001
I-5.6.3.Test.14 DFP Test Significance instructions set the FPCC and CR field BF when the number of significant digits in operand FRB[p] is less than the reference significance (which is the contents of bits 58:63 of FRA for dtstsf dtstsfq and is the UMI value for dtstsfi dtstsfiq)		
dtstsf dtstsfq dtstsfi dtstsfiq	FRB[p] is a finite number (including zeros) The number of significant digits in FRB[p] is less than the reference significance reference significance != 0	FPCC = 0100 $CR_{4xBF:4xBF+3}$ = 0100
I-5.6.3.Test.15 DFP Test Significance instructions set the FPCC and CR field BF when the reference significance (which is the contents of bits 58:63 of FRA for dtstsf dtstsfq and is the UMI value for dtstsfi dtstsfiq) is zero and FRB[p] is finite number		
dtstsf dtstsfq dtstsfi dtstsfiq	FRB[p] is a finite number (including zeros) reference significance = 0	FPCC = 0100 $CR_{4xBF:4xBF+3}$ = 0100
I-5.6.3.Test.16 DFP Test Significance instructions set the FPCC and CR field BF when the number of significant digits in operand FRB[p] is greater than the reference significance (which is the contents of bits 58:63 of FRA for dtstsf dtstsfq and is the UMI value for dtstsfi dtstsfiq)		
dtstsf dtstsfq dtstsfi dtstsfiq	FRB[p] is a finite number (including zeros) The number of significant digits in FRB[p] is greater than the reference significance reference significance != 0	FPCC = 1000 $CR_{4xBF:4xBF+3}$ = 1000
I-5.6.3.Test.17 DFP Test Significance instructions set the FPCC and CR field BF when the number of significant digits in operand FRB[p] equals to the reference significance (which is the contents of bits 58:63 of FRA for dtstsf dtstsfq and is the UMI value for dtstsfi dtstsfiq)		
dtstsf dtstsfq dtstsfi dtstsfiq	FRB[p] is a finite number (including zeros) The number of significant digits in FRB[p] equals to the reference significance reference significance != 0	FPCC = 0010 $CR_{4xBF:4xBF+3}$ = 0010
I-5.6.3.Test.18 DFP Test Significance instructions set the FPCC and CR field BF when the FRB[p] is an Infinity or a NaN		
dtstsf dtstsfq dtstsfi dtstsfiq	FRB[p] is infinity or NaN	FPCC = 0001 $CR_{4xBF:4xBF+3}$ = 0001

6.5. DFP Format Instructions

Architecture sections:

- I- 5.6.6 DFP Format Instructions

Scenario groups:

- Setting CR1
- Setting FPRF
- Setting FR and FI
- Correct Computations

6.5.1. Setting CR1

DFP Format Recording Instructions: ddedpd. ddedpdq. denbcd. denbcdq. dxex. dxexq. diex. diexq. dscli. dscliq. dscri. dscriq.

Instructions tested	Compliance conditions	Expected result
I-5.6.6.CR1.1 *DFP Format Recording Instructions*, set CR1 field (bits 36:39 of the CR) to the FP exception status, copied from bits 32:35 of the FPSCR.		
DFP Format Recording Instructions	Rc = 1	CR1 = $FPSCR_{32:35}$
I-5.6.1.CR1.2 Non-recording instructions don't change the CR1 field		
DFP Format Recording Instructions	Rc = 0	CR1 unchanged

6.5.2. Setting FPRF

Note: The tests in this section assume no exceptions occurrences

Instructions tested	Compliance conditions	Expected result
I-5.6.6.FPRF.1 DFP Encode BCD To DPD instruction sets FPRF correctly for Signaling NaN result		
denbcd[.] denbcdq[.]	Result value class is Signaling NaN	FPRF = 00001
I-5.6.6.FPRF.2 DFP Encode BCD To DPD instruction sets *FPRF correctly for Quiet NaN result*		
denbcd[.] denbcdq[.]	Result value class is Quiet NaN	FPRF = 10001
I-5.6.6.FPRF.3 DFP Encode BCD To DPD instruction sets FPRF correctly for -Infinity result		
denbcd[.] denbcdq[.]	Result value class is -Infinity	FPRF = 01001
I-5.6.6.FPRF.4 DFP Encode BCD To DPD instruction sets *correctly for -Normalized result*		
denbcd[.] denbcdq[.]	Result value class is -Normalized Number	FPRF = 01000
I-5.6.6.FPRF.5 DFP Encode BCD To DPD instruction sets *FPRF correctly for -Subnormal Number result*		
denbcd[.] denbcdq[.]	Result value class is -Subnormal Number	FPRF = 11000
I-5.6.6.FPRF.6 DFP Encode BCD To DPD instruction sets *FPRF correctly for -Zero result*		
denbcd[.] denbcdq[.]	Result value class is -Zero	FPRF = 10010
I-5.6.6.FPRF. 7 DFP Encode BCD To DPD instruction sets *FPRF correctly for +Zero result*		
denbcd[.] denbcdq[.]	Result value class is +Zero	FPRF = 00010
I-5.6.6.FPRF.8 DFP Encode BCD To DPD instruction sets *FPRF correctly for +Subnormal Number result*		
denbcd[.] denbcdq[.]	Result value class is +Subnormal Number	FPRF = 10100
I-5.6.6.FPRF.9 DFP Encode BCD To DPD instruction sets *FPRF correctly for +Normalized result*		
denbcd[.] denbcdq[.]	Result value class is +Normalized Number	FPRF = 00100

Instructions tested	Compliance conditions	Expected result
I-5.6.6.FPRF.10 DFP Encode BCD To DPD instruction sets *FPRF correctly for +Infinity result*		
denbcd[.] denbcdq[.]	Result value class is +Infinity	FPRF = 00101

6.5.3. Setting FR and FI

Instructions tested	Compliance conditions	Expected result
I-5.6.6.FPRF.1 DFP Encode BCD To DPD instruction always sets FR and FI to 0		
denbcd[.] denbcdq[.]		FR = 0 FI = 0

6.5.4. Correct Computations

DFP Format Instructions: ddedpd[.] ddedpdq[.] denbcd[.] denbcdq[.] dxex[.] dxexq[.] diex[.] diexq[.] dscli[.] dscliq[.] dscri[.] dscriq[.]

Instructions tested	Compliance conditions	Expected result
I-5.6.6.Comp.1 *DFP Format Instructions* perform correct computations		
DFP Format Instructions	No exception occurs	Correct result of operation

7. Vector Facility (Chapter I.6)

Table of Contents

7.1. Vector Storage Access Instructions

Architecture sections:

- I-6.7 Vector Storage Access Instructions

Scenario groups:

- Correct Loading data into VRT
- Correct Storing data in storage addressed by EA
- Storage Access Exceptions

Guideline: each scenario in this section should be tested in both Big-Endian and Little-Endian mode

7.1.1. Correct Loading of data into VRT

Architecture sections:

- I-6.7.2 Vector Load Instructions
- I-6.7.4 Vector Alignment Support Instructions

Vector Load and Alignment Support Instructions: lvebx lvehx lvewx lvx lvxl lvsl lvsr

Instructions tested	Compliance conditions	Expected result
I-6.7.VRT.1 Setting the VRT correctly with the loaded storage addressed by the effective address		
Vector Load and Alignment Support Instructions		Correct data is loaded into the target register

7.1.2. Stored data in storage addressed by EA

Architecture sections:

- I-6.7.3 Vector Store Instructions

Vector Store Instructions: stvebx stvehx stvewx stvx stvxl

Instructions tested	Compliance conditions	Expected result
I-6.7.3.Store.1 Storing the data correctly in the storage memory addressed by the effective address		
Vector Store Instructions		Storing the data correctly in the memory storage addressed by the effective address

7.1.3. Storage Access Exceptions

Instructions tested	Compliance conditions	Expected result
I-6.7.3.StorageEx.1 A system data storage error handler will be invoked when an instruction attempts to access unavailable storage		
Vector Load and Alignment Support Instructions *Vector Store Instructions*	The program is not allowed to modify the target storage (Store only) or the program attempts to access storage that is unavailable.	system data storage error handler is invoked

7.2. Vector Permute and Formatting Instructions

Architecture sections:

- I-6.8 Vector Permute and Formatting Instructions

Scenario groups:

- Setting SAT bit
- Correct computations

7.2.1. Setting SAT bit

Architecture sections:

- I-6.8.1 Vector Pack and Unpack Instructions

Vector pack saturating instructions: vpksdss vpksdus vpkshss vpkshus vpkswss vpkswus vpkudus vpkuhus vpkuwus

Instructions tested	Compliance conditions	Expected result
I-6.8.1.SAT.1 *Vector pack saturating instructions* set SAT bit in the VSCR to 1 when their result saturates		
Vector pack saturating instructions	The result is saturated	SAT = 1
I-6.8.1.SAT.2 *Vector pack saturating instructions* dont change SAT when their result is not saturated even if it was turned on SAT is a sticky bit		
Vector pack saturating instructions	SAT = 1 Result is not saturated	SAT = 1
I-6.8.1.SAT.3 *Vector pack saturating instructions* dont change SAT when their result is not saturated and it was turned off		
Vector pack saturating instructions	SAT = 0 Result is not saturated	SAT = 0

7.2.2. Correct computations

Vector pack and unpack instructions (section I-6.8.1): vpkpx vpksdss vpksdus vpkshss vpkshus vpkswss vpkswus vpkudus vpkudum vpkuhum vpkuhus vpkuwus vpkuwum vupkhpx vupklpx vupkhsb vupkhsh vupkhsw vupklsb vupklsh vupklsw

Vector merge instructions (section I-6.8.2): vmrghb vmrghh vmrglb vmrglh vmrghw vmrglw vmrgew vmrgow

Vector splat instructions (section I-6.8.3): vspltb vspltw vsplth vspltisb vspltish vspltisw

Vector permute instructions (section I-6.8.4): vperm vpermr

Vector select instructions (section I-6.8.5): vsel

Vector shift instructions (section I-6.8.6): vsl vsldoi vslo vsr vsro vslv vsrv

Vector extract element instructions (section I-6.8.7): vextractub vextractuh vextractuw vextractd

Vector insert element instructions (section I-6.8.8): vinsertb vinserth vinsertw vinsertd

Instructions tested	Compliance conditions	Expected result
I-6.8.Computation.1 Result of instructions computation is correct		
Vector pack and unpack instructions *Vector merge instructions* *Vector splat instructions* *Vector permute instructions* *Vector select instructions* *Vector shift instructions* *Vector extract element instructions* *Vector insert element instructions*		Correct result in the VRT

7.3. Vector Integer Instructions

Architecture sections:

- I-6.9 Vector Integer Instructions

Scenario groups:

- Setting SAT bit
- Setting CR field 6
- Correct Computations

7.3.1. Setting SAT bit

Architecture sections:

- I-6.9.1 Vector Integer Arithmetic Instructions

Vector Integer Arithmetic Saturating Instructions: vaddshs vaddsbs vaddsws vaddubs vadduhs vadduws vsubsbs vsubshs vsubsws vsububs vsubuhs vsubuws vmhaddshs vmhraddshs vmsumshs vmsumuhs vsumsws vsum2sws vsum4sbs vsum4shs vsum4ubs

Instructions tested	Compliance conditions	Expected result
I-6.9.1.SAT.1 *Vector Integer Arithmetic Saturating Instructions* set SAT bit in the VSCR to 1 when their result saturates		
Vector Integer Arithmetic Saturating Instructions	The result is saturated	SAT = 1
I-6.9.1.SAT.2 *Vector Integer Arithmetic Saturating Instructions* dont change SAT when their result is not saturated even if it was turned on SAT is a sticky bit		
Vector Integer Arithmetic Saturating Instructions	SAT = 1 Result is not saturated	SAT = 1
I-6.9.1.SAT.3 *Vector Integer Arithmetic Saturating Instructions* dont change SAT when their result is not saturated and it was turned off		
Vector Integer Arithmetic Saturating Instructions	SAT = 0 Result is not saturated	SAT = 0

7.3.2. Setting CR field 6

Architecture sections:

- I-6.9.3 Vector Integer Compare Instructions

Vector Integer Compare Recording Instructions: vcmpequb. vcmpequh. vcmpequw. vcmpequd. vcmpgtsb. vcmpgtsd. vcmpgtsh. vcmpgtsw. vcmpgtub. vcmpgtud. vcmpgtuh. vcmpgtuw. vcmpneb. vcmpnezb. vcmpneh. vcmpnezh. vcmpnew. vcmpnezw.

Instructions tested	Compliance conditions	Expected result
I-6.9.3.CR.1 CR field 6 is set by *Vector Integer Compare Recording Instructions* to reflect the result of the comparison when it is **true for all the element pairs**		
Vector Integer Compare Recording Instructions	The result of comparison is true for all the element pairs (VRT is set to all 1s)	$(\text{CR field 6})_{0:3} = 1000$
I-6.9.3.CR.2 CR field 6 is set by *Vector Integer Compare Recording Instructions* to reflect the result of the comparison when it is **false for all the element pairs**		
Vector Integer Compare Recording Instructions	The result of comparison is false for all the element pairs (VRT is set to all 0s)	$(\text{CR field 6})_{0:3} = 0010$
I-6.9.3.CR.3 CR field 6 is set by *Vector Integer Compare Recording Instructions* to reflect the result of the comparison when it **is neither true for all element pairs nor false for all the element pairs**		
Vector Integer Compare Recording Instructions	VRT includes both 0s and 1s	$(\text{CR field 6})_{0:3} = 0000$

7.3.3. Correct computations

Vector Integer Arithmetic Instructions (I-6.9.1): vaddcuw vaddshs vaddsbs vaddsws vaddudm vaddubm vadduhm vadduwm vaddubs vadduws vadduhs vadduqm vaddcuq vaddeuqm vaddecuq vsubcuw vsubshs vsubsbs vsubsws vsububm vsubuhm vsubudm vsubuwm vsububs vsubuws vsubuhs vsubuqm vsubcuq vsubeuqm vsubecuq vmulesb vmulosb vmuleub vmuloub vmulesh vmulosh vmuleuh vmulouh vmulesw vmulosw vmuleuw vmulouw vmuluwm vmhaddshs vmhraddshs vmladduhm vmsumubm vmsummbm vmsumshm vmsumshs vmsumuhm vmsumuhs vmsumudm vsumsws vsum2sws vsum4sbs vsum4shs vsum4ubs vnegw vnegd

Vector Extend Sign Instructions (I-6.9.2): vextsb2w vextsh2w vextsb2d vextsh2d vextsw2d

Vector Integer Average Instructions (I-6.9.2.1): vavgsb vavgsw vavgsh vavgub vavguh vavguw

Vector Integer Absolute Difference Instructions (I-6.9.2.2): vabsdub vabsduh vabsduw

Vector Integer Maximum and Minimum Instructions (I-6.9.2.3): vmaxsb vmaxub vmaxsd vmaxud vmaxsh vmaxuh vmaxsw vmaxuw vminsb vminub vminsd vminud vminsh vminuh vminsw vminuw

Vector Integer Compare Instructions (I-6.9.3): vcmpequb[.] vcmpequh[.] vcmpequw[.] vcmpequd[.] vcmpgtsb[.] vcmpgtsd[.] vcmpgtsh[.] vcmpgtsw[.] vcmpgtub[.] vcmpgtud[.] vcmpgtuh[.] vcmpgtuw[.] vcmpneb[.] vcmpnezb[.] vcmpneh[.] vcmpnezh[.] vcmpnew[.] vcmpnezw[.]

Vector Logical Instructions (I-6.9.4): vand veqv vandc vnand vorc vor vnor vxor

Vector Parity Byte Instructions (I-6.9.5): vprtybw vprtybd vprtybq

Vector Integer Rotate and Shift Instructions (I-6.9.6): vrlb vrlw vrlh vrld vslb vslw vslh vsld vsrb vsrw vsrh vsrd vsrab vsraw vsrah vsrad vrlwnm vrlwmi vrldnm vrldmi

Instructions tested	Compliance conditions	Expected result
I-6.9.Computation.1 Result of instructions computation is correct		
Vector Integer Arithmetic Instructions *Vector Extend Sign Instructions* *Vector Integer Average Instructions* *Vector Integer Absolute Difference Instructions* *Vector Integer Maximum and Minimum Instructions* *Vector Integer Compare Instructions* *Vector Logical Instructions* *Vector Parity Byte Instructions* *Vector Integer Rotate and Shift Instructions*		Correct result computation

7.4. Vector Floating-Point Instruction Set

Architecture sections:

- I-6.10 Vector Floating-Point Instruction Set
- I-6.6.2 Vector Floating-Point Exceptions

Scenario groups:

- Setting SAT bit
- Setting CR field 6
- Correct Computations
- NaN Operand exception
- Invalid Operation Exception
- Zero Divide Exception
- Log of Zero Exception

- Overflow Exception
- Underflow Exception
- Rounding Modes
- Exceptions with denormalized values

7.4.1. Setting SAT bit

Architecture sections:

- I-6.10.3 Vector Floating-Point Rounding and Conversion Instructions

Vector Floating-Point Convert Saturating Instructions: vctsxs vctuxs

Instructions tested	Compliance conditions	Expected result
I-6.10.3.SAT.1 *Vector Floating-Point Convert Saturating Instructions* set SAT bit in the VSCR to 1 when their result saturates		
Vector Floating-Point Convert Saturating Instructions	The result is saturated	SAT = 1
I-6.10.3.SAT.2 *Vector Floating-Point Convert Saturating Instructions* dont change SAT when their result is not saturated even if it was turned on SAT is a sticky bit		
Vector Floating-Point Convert Saturating Instructions	SAT = 1 Result is not saturated	SAT = 1
I-6.10.3.SAT.3 *Vector Floating-Point Convert Saturating Instructions* dont change SAT when their result is not saturated and it was turned off		
Vector Floating-Point Convert Saturating Instructions	SAT = 0 Result is not saturated	SAT = 0

7.4.2. Setting CR field 6

Architecture sections:

- I-6.10.4 Vector Floating-Point Compare Instructions

Instructions tested	Compliance conditions	Expected result
I-6.10.4.CR.1 CR field 6 is set by vector compare bounds floating-point instruction *to indicate whether the elements in VRA are* **within the bounds specified by the corresponding elements in VRB**		
vcmpbfp.	Rc = 1 All four elements in VRA are within the bounds specified by the corresponding elements in VRB	(CR field 6)$_{0:3}$ = 0010
I-6.10.4.CR.2 CR field 6 is set by vector compare bounds floating-point instruction *to indicate whether the elements in VRA are* **not within the bounds specified by the corresponding elements in VRB**		
vcmpbfp.	Rc = 1 Not all four elements in VRA are within the bounds specified by the correspond-ing elements in VRB	(CR field 6)$_{0:3}$ = 0000
I-6.10.4.CR.3 CR field 6 is set by Vector floating-point not bounding compare Recording Instructions *to reflect the result of the comparison when it is* **true for all the element pairs**		
vcmpeqfp. vcmpgefp. vcmpgtfp.	Rc = 1 The result of comparison is true for all the element pairs (VRT is set to all 1s)	(CR field 6)$_{0:3}$ = 1000

Instructions tested	Compliance conditions	Expected result
I-6.10.4.CR.4 CR field 6 is set by Vector floating-point not bounding compare Recording Instructions *to reflect the result of the comparison when it is* **false for all the element pairs**		
vcmpeqfp. vcmpgefp. vcmpgtfp.	Rc = 1 The result of comparison is false for all the element pairs (VRT is set to all 0s)	(CR field 6)$_{0:3}$ = 0010
I-6.10.4.CR.5 CR field 6 is set by Vector floating-point not bounding compare Recording Instructions *to reflect the result of the comparison when it* **is neither true for all element pairs nor false for all the element pairs**		
vcmpeqfp. vcmpgefp. vcmpgtfp.	Rc = 1 VRT includes both 0s and 1s	(CR field 6)$_{0:3}$ = 0000

7.4.3. Correct Computations

Vector Floating-Point Arithmetic Instructions (I-6.10.1): vaddfp vsubfp vmaddfp vnmsubfp

Vector Floating-Point Maximum and Minimum Instructions (I-6.10.2): vmaxfp vminfp

Vector Floating-Point Rounding and Conversion Instructions (I-6.10.3): vctsxs vctuxs vcfsx vcfux vrfim vrfin vrfip vrfiz

Vector Floating-Point Compare Instructions (I-6.10.4): vcmpbfp[.] vcmpeqfp[.] vcmpgefp[.] vcmpgtfp[.]

Vector Floating-Point Estimate Instructions (I-6.10.5): vexptefp vlogefp vrefp vrsqrtefp

Instructions tested	Compliance conditions	Expected result
I-6.10.Computation.1 Result of instructions computation is correct		
Vector Floating-Point Arithmetic Instructions *Vector Floating-Point Maximum and Minimum Instructions* *Vector Floating-Point Rounding and Conversion Instructions* *Vector Floating-Point Compare Instructions* *Vector Floating-Point Estimate Instructions*		Correct result computation

7.4.4. NaN Operand exception

Vector instructions that would normally produce floating-point results: vaddfp vsubfp vmaddfp vnmsubfp vmaxfp vminfp vexptefp vlogefp vrefp vrsqrtefp vcfsx vcfux vrfim vrfin vrfip vrfiz

Vector Convert to Fixed-Point Word instructions: vctsxs vctuxs

Vector Compare Bounds Floating-Point instruction: vcmpbfp[.]

Other Vector Floating-Point Compare Instructions: vcmpeqfp[.] vcmpgefp[.] vcmpgtfp[.]

Note: In all cases, if the selected source NaN is a signaling NaN, it is converted to the corresponding Quiet NaN before being placed into the target element.

Instructions tested	Compliance conditions	Expected result
I-6.6.2.NaN.1 For *vector instructions that would normally produce floating-point results* if the element in the selected source VRA or VRB or VRC is a NaN then the result is that NaN		
Representative of *vector instructions that would normally produce floating-point results*	VRA = NaN , or VRB = NaN or VRC = NaN	Result equals the source NaN. If more than one operand is NaN, the result equals the first NaN according to Alphabetical order
I-6.6.2.NaN.2 For *vector instructions that would normally produce floating-point results* if there is an invalid operation exception then the result is QNaN 0x7FC0_0000		
Representative of *vector instructions that would normally produce floating-point results*	Invalid operation exception occurs	Result is the QNaN 0x7FC0_0000
I-6.6.2.NaN.3 For *Vector Convert to Fixed-Point Word instructions,* the corresponding result is 0x0000_0000, and $VSCR_{SAT}$ is not affected		
Representative of *Vector Convert to Fixed-Point Word instructions*	A source value is NaN	Result is 0x0000_0000 $VSCR_{SAT}$ is not affected
I-6.6.2.NaN.4 For *Vector Compare Bounds Floating-Point instruction,* the corresponding result is 0xC000_0000		
Vector Compare Bounds Floating-Point instruction	A source value is NaN	Result is 0xC000_0000
I-6.6.2.NaN.5 For the *Other Vector Floating-Point Compare Instructions,* the corresponding result is 0x0000_0000		
Representative of *Other Vector Floating-Point Compare Instructions*	A source value is NaN	Result is 0x0000_0000

7.4.5. Invalid Operation Exception

Instructions tested	Compliance conditions	Expected result
I-6.6.2.IOE.1 An Invalid Operation Exception occurs when a source value or set of source values is invalid for the specified operation.		
Representative of *vector instructions that would normally produce floating-point results*	One of the following occurred: • Magnitude subtraction of infinities • Multiplication of infinity by zero • square root estimate of a negative, nonzero number or -infinity. • estimate of a negative, nonzero number or -infinity.	Result is 0x7FC0_0000

7.4.6. Zero Divide Exception

Instructions tested	Compliance conditions	Expected result
I-6.6.2.ZDE.1 A Zero Divide Exception occurs when a Vector Reciprocal Estimate Floating-Point or Vector Reciprocal Square Root Estimate Floating-Point *instruction is executed with a source value of zero. The corresponding result is an infinity, where the sign is the sign of the source value.*		
vrefp vrsqrtefp	Source value of zero	Result is an infinity, where the sign *is the sign of the source value.*

7.4.7. Log of Zero Exception

Instructions tested	Compliance conditions	Expected result
I-6.6.2.LZE.1 A Log of Zero Exception occurs when a Vector Log Base 2 Estimate Floating-Point instruction is executed with a source value of zero. The corresponding result is -Infinity.		
vlogefp	Source value of zero	Result is -Infinity

7.4.8. Overflow Exception

Instructions tested	Compliance conditions	Expected result
I-6.6.2.OE.1 Action taken when overflow exception occurs with *vector instructions that would normally produce floating-point results*		
Representative of *vector instructions that would normally produce floating-point results*	The magnitude of what would have been the result if the exponent range were unbounded exceeds that of the largest finite floating-point number for the target floating-point format.	Result is an infinity, where the sign is the sign of the intermediate result.
I-6.6.2.OE.2 Action taken when overflow exception occurs with vector convert to unsigned fixed-point word with positive source value		
vctuxs	Source value is a positive number too large to be represented in the target fixed-point format or source value is a +infinity	Result is 0xFFFF_FFFF $VSCR_{SAT}$ is set to 1
I-6.6.2.OE.3 Action taken when overflow exception occurs with vector convert to unsigned fixed-point word with negative source value		
vctuxs	Source value is a negative number too large to be represented in the target fixed-point format or -infinity	Result is 0x0000_0000 $VSCR_{SAT}$ is set to 1
I-6.6.2.OE.4 Action taken when overflow exception occurs with vector convert to signed fixed-point word with positive source value		
vctsxs	Source value is a positive number too large to be represented in the target fixed-point format or source value is a +infinity	Result is 0x7FFF_FFFF $VSCR_{SAT}$ is set to 1
I-6.6.2.OE.5 Action taken when overflow exception occurs with vector convert to signed fixed-point word with negative source value		
vctsxs	Source value is a negative number too large to be represented in the target fixed-point format or -infinity	Result is 0x8000_0000 $VSCR_{SAT}$ is set to 1

7.4.9. Underflow Exception

Instructions tested	Compliance conditions	Expected result
I-6.6.2.UE.1 Action taken when underflow exception occurs and $VSCR_{NJ}$ =0		
Representative of *vector instructions that would normally produce floating-point results*	$VSCR_{NJ}$ =0 Nonzero intermediate result computed as though both the precision and the exponent range were unbounded is less in magnitude than the smallest normalized floating-point number for the target floating-point format (underflow occurrence)	The corresponding result is the value produced by denormalizing and rounding the intermediate result.
I-6.6.2.UE.2 Action taken when underflow exception occurs and $VSCR_{NJ}$ =1		
Representative of *vector instructions that would normally produce floating-point results*	$VSCR_{NJ}$ =1 Nonzero intermediate result computed as though both the precision and the exponent range were unbounded is less in magnitude than the smallest normalized floating-point number for the target floating-point format (underflow occurrence)	The corresponding result is a zero, where the sign is the sign of the intermediate result.

7.4.10. Rounding Modes

Instructions tested	Compliance conditions	Expected result
I-6.10.Rounding.1 Round to Nearest mode		
vrfin vaddfp vsubfp vmaddfp vnmsubfp		Rounding is performed according to Round to Nearest mode
I-6.10.Rounding.2 Round toward zero		
vrfiz vctsxs vctuxs		Rounding is performed according to Round toward zero mode
I-6.10.Rounding.3 Round toward -Infinity		
vrfim		Rounding is performed according to Round toward -Infinity mode
I-6.10.Rounding.4 Round toward +Infinity		
vrfip		Rounding is performed according to Round toward +Infinity mode

7.4.11. Exceptions with denormalized values

Instructions tested	Compliance conditions	Expected result
I-6.6.2.DEN.1 Exceptions that can be caused by a zero source value can be caused by a denormalized source value when $VSCR_{NJ}$ =1.		
Representative of *vector instructions that would normally produce floating-point results*	$VSCR_{NJ}$ =1 Denormalized value	Exceptions are caused as if the source is zero
I-6.6.2.DEN.2 Exceptions that can be caused by a nonzero source value cannot be caused by a denormalized source value when $VSCR_{NJ}$ =1.		
Representative of *vector instructions that would normally produce floating-point results*	$VSCR_{NJ}$ =1 Denormalized value	Exceptions are not caused as if it is a nonzero value source
I-6.6.2.DEN.3 When $VSCR_{NJ}$ =0 exceptions treat denormalized values as nonzero values		
Representative of *vector instructions that would normally produce floating-point results*	$VSCR_{NJ}$ =0 Denormalized value	Exceptions operate with denormalized values as nonzero value

7.5. Vector Exclusive-OR-based Instructions

Architecture sections:

- I-6.11 Vector Exclusive-OR-based Instructions

Scenario groups:

- Correct Computations

7.5.1. Correct Computations

Vector AES Instructions (I-6.11.1): vcipher vcipherlast vncipher vncipherlast vsbox

Vector SHA-256 and SHA-512 Sigma Instructions (I-6.11.2): vshasigmad vshasigmaw

Vector Binary Polynomial Multiplication Instructions (I-6.11.3): vpmsumb vpmsumd vpmsumh vpmsumw

Vector Permute and Exclusive-OR Instruction (I-6.11.4): vpermxor

Instructions tested	Compliance conditions	Expected result
I-6.11.Computation.1 Result of instructions computation is correct		
Vector AES Instructions *Vector SHA-256 and SHA-512 Sigma Instructions* *Vector Binary Polynomial Multiplication Instructions* *Vector Permute and Exclusive-OR Instruction*		Correct result computation

7.6. Vector Gather Instruction

Architecture sections:

- I-6.12 Vector Gather Instruction

Scenario groups:

- Correct Computation

7.6.1. Correct Computation

Instructions tested	Compliance conditions	Expected result
I-6.12.Computation.1 Result of instructions computation is correct		
vgbbd		Correct result computation

7.7. Vector Count Leading or Trailing Zeros and Vector Extract Element Instructions

Architecture sections:

- I-6.13 Vector Count Leading Zeros Instructions
- I-6.14 Vector Count Trailing Zeros Instructions
- I-6.14.1 Vector Count Leading/Trailing Zero LSB Instructions
- I-6.14.2 Vector Extract Element Instructions

Scenario groups:

- Correct Computations

7.7.1. Correct Computations

Vector Count Leading Zeros Instructions (I-6.13): vclzb vclzw vclzh vclzd

Vector Count Trailing Zeros Instructions (I-6.14): vctzb vctzw vctzh vctzd

Vector Count Leading/Trailing Zero LSB Instructions (I-6.14.1): vclzlsbb vctzlsbb

Vector Extract Element Instructions (I-6.14.2): vextublx vextubrx vextuhlx vextuhrx vextuwlx vextuwrx

Instructions tested	Compliance conditions	Expected result
I-6.13.Computation.1 Result of instructions computation is correct		
Vector Count Leading Zeros Instructions *Vector Count Trailing Zeros Instructions* *Vector Count Leading/Trailing Zero LSB Instructions* *Vector Extract Element Instructions*		Correct result computation

7.8. Vector Population Count Instructions

Architecture sections:

- I-6.15 Vector Population Count Instructions

Scenario groups:

- Correct Computations

7.8.1. Correct Computations

Instructions tested	Compliance conditions	Expected result
I-6.15.Computation.1 Result of instructions computation is correct		
vpopcntb vpopcnth vpopcntd vpopcntw		Correct result computation

7.9. Vector Bit Permute Instructions

Architecture sections:

- I-6.16 Vector Bit Permute Instructions

Scenario groups:

- Correct Computations

7.9.1. Correct Computations

Instructions tested	Compliance conditions	Expected result
I-6.16.Computation.1 Result of instructions computation is correct		
vbpermd vbpermq		Correct result computation

7.10. Decimal Integer Instructions

Architecture sections:

- I-6.17.1 Decimal Integer Arithmetic Instructions
- I-6.17.2 Decimal Integer Format Conversion Instructions
- I-6.17.3 Decimal Integer Sign Manipulation Instructions
- I-6.17.4 Decimal Integer Shift and Round Instructions
- I-6.17.5 Decimal Integer Truncate Instructions

Scenario groups:

- Setting CR field 6
- Correct Computations

7.10.1. Setting CR field 6

Decimal Integer Arithmetic Instructions (I-6.17.1): bcdadd. bcdsub.

Decimal Integer Format Conversion Instructions (I-6.17.2): bcdcfn. bcdcfz. bcdctn. bcdctz. bcdcfsq. bcdctsq. vmul10uq vmul10euq vmul10cuq vmul10ecuq

Decimal Integer Sign Manipulation Instructions (I-6.17.3): bcdcpsgn. bcdsetsgn.

Decimal Integer Shift and Round Instructions (I-6.17.4): bcds. bcdus. bcdsr.

Decimal Integer Truncate Instructions (I-6.17.5): bcdtrunc. bcdutrunc.

Instructions tested	Compliance conditions	Expected result
I-6.17.CR.1 CR field 6 is set by Decimal Integer Arithmetic Instructions to reflect the unbounded result when it is **equal to zero**		
bcdadd. bcdsub.	Unbounded result is equal to zero	$(\text{CR field 6})_{0:3} = 0010$
I-6.17.CR.2 CR field 6 is set by Decimal Integer Arithmetic Instructions to reflect the unbounded result when it is **greater than zero and overflows**		
bcdadd. bcdsub.	Unbounded result is greater than zero and overflows	$(\text{CR field 6})_{0:3} = 0101$
I-6.17.CR.3 CR field 6 is set by Decimal Integer Arithmetic Instructions to reflect the unbounded result when it is **greater than zero and doesnt overflow**		
bcdadd. bcdsub.	Unbounded result is greater than zero and doesnt overflow	$(\text{CR field 6})_{0:3} = 0100$
I-6.17.CR.4 CR field 6 is set by Decimal Integer Arithmetic Instructions to reflect the unbounded result when it is **less than zero and overflows**		
bcdadd. bcdsub.	Unbounded result is less than zero and overflows	$(\text{CR field 6})_{0:3} = 1001$
I-6.17.CR.5 CR field 6 is set by Decimal Integer Arithmetic Instructions to reflect the unbounded result when it is **less than zero and doesnt overflow**		
bcdadd. bcdsub.	Unbounded result is less than zero and doesnt overflow	$(\text{CR field 6})_{0:3} = 1000$
I-6.17.CR.6 CR field 6 is set by Decimal Integer Arithmetic Instructions if either VR[VRA] or VR[VRB] is an invalid encoding *of a 31-digit signed decimal value*		
bcdadd. bcdsub.	src1 or src2 is an invalid encoding of a 31-digit signed decimal value	$(\text{CR field 6})_{0:3} = 0001$
I-6.17.CR.7 CR field 6 is set by Decimal Integer Instructions to reflect source value in VR[VRB] is **valid and is equal to zero**		
bcdcfn. bcdcfz. bcdctsq. bcdsetsgn.	source is equal to zero	$(\text{CR field 6})_{0:3} = 0010$
I-6.17.CR.8 CR field 6 is set by Decimal Integer Instructions to reflect source value in VR[VRB] is **valid and is greater than zero**		
bcdcfn. bcdcfz. bcdctsq. bcdsetsgn.	source is greater than zero	$(\text{CR field 6})_{0:3} = 0100$
I-6.17.CR.9 CR field 6 is set by Decimal Integer Instructions to reflect source value in VR[VRB] is **valid and is less than zero**		
bcdcfn. bcdcfz. bcdctsq. bcdsetsgn.	source is less than zero	$(\text{CR field 6})_{0:3} = 1000$
I-6.17.CR.10 CR field 6 is set by Decimal Integer Instructions to reflect source value in VR[VRB] is **an invalid encoding of the value**		
bcdcfn. bcdcfz. bcdctsq. bcdsetsgn.	source has invalid encoding	$(\text{CR field 6})_{0:3} = 0001$
I-6.17.CR.11 CR field 6 is set by Decimal Integer Instructions to reflect source value in VR[VRB] is **valid and is equal to zero**		
bcdctn. bcdctz. bcdcfsq.	source is equal to zero	$(\text{CR field 6})_{0:3} = 0010$
I-6.17.CR.12 CR field 6 is set by Decimal Integer Instructions to reflect source value in VR[VRB] is **valid, is greater than zero, and is not too large for target format**		

Instructions tested	Compliance conditions	Expected result
bcdctn. bcdctz. bcdcfsq.	source is greater than zero and is not too large for target format	(CR field 6)$_{0:3}$ = 0100
I-6.17.CR.13 CR field 6 is set by Decimal Integer Instructions to reflect source value in VR[VRB] is **valid, is greater than zero, and is too large for target format**		
bcdctn. bcdctz. bcdcfsq.	source is greater than zero and is too large for target format	(CR field 6)$_{0:3}$ = 0101
I-6.17.CR.14 CR field 6 is set by Decimal Integer Instructions to reflect source value in VR[VRB] is **valid, is less than zero, and is not too large for target format**		
bcdctn. bcdctz. bcdcfsq.	source is less than zero and is not too large for target format	(CR field 6)$_{0:3}$ = 1000
I-6.17.CR.15 CR field 6 is set by Decimal Integer Instructions to reflect source value in VR[VRB] is **valid, is less than zero, and is too large for target format**		
bcdctn. bcdctz. bcdcfsq.	source is less than zero and is too large for target format	(CR field 6)$_{0:3}$ = 1001
I-6.17.CR.16 CR field 6 is set by Decimal Integer Instructions to reflect source value in VR[VRB] has **an invalid encoding of the value**		
bcdctn. bcdctz.	source has invalid encoding	(CR field 6)$_{0:3}$ = 0001
I-6.17.CR.17 CR field 6 is set by Decimal Integer Instructions to reflect VR[VRA] and VR[VRB] are **valid and the result is equal to zero**		
bcdcpsgn.	result is equal to zero	(CR field 6)$_{0:3}$ = 0010
I-6.17.CR.18 CR field 6 is set by Decimal Integer Instructions to reflect VR[VRA] and VR[VRB] are **valid and the result is greater than zero**		
bcdcpsgn.	result is greater than zero	(CR field 6)$_{0:3}$ = 0100
I-6.17.CR.19 CR field 6 is set by Decimal Integer Instructions to reflect VR[VRA] and VR[VRB] are **valid and the result is less than zero**		
bcdcpsgn.	result is less than zero	(CR field 6)$_{0:3}$ = 1000
I-6.17.CR.20 CR field 6 is set by Decimal Integer Instructions to reflect VR[VRA] or VR[VRB] has **an invalid encoding of the value**		
bcdcpsgn.	VR[VRA] or VR[VRB] has invalid encoding	(CR field 6)$_{0:3}$ = 0001
I-6.17.CR.21 CR field 6 is set by Decimal Integer Instructions to reflect source value in VR[VRB] is **valid and is equal to zero**		
bcds. bcdus. bcdsr. bcdtrunc. bcdutrunc.	source is equal to zero	(CR field 6)$_{0:3}$ = 0010
I-6.17.CR.22 CR field 6 is set by Decimal Integer Instructions to reflect source value in VR[VRB] is **valid, is greater than zero, and significant digits were not left shifted out and were not truncated**		
bcds. bcdus. bcdsr. bcdtrunc. bcdutrunc.	source is greater than zero and significant digits were not left shifted out and were not truncated	(CR field 6)$_{0:3}$ = 0100
I-6.17.CR.23 CR field 6 is set by Decimal Integer Instructions to reflect source value in VR[VRB] is **valid, is greater than zero, and significant digits were left shifted out or were truncated**		
bcds. bcdus. bcdsr. bcdtrunc. bcdutrunc.	source is greater than zero and significant digits were left shifted out or were truncated	(CR field 6)$_{0:3}$ = 0101
I-6.17.CR.24 CR field 6 is set by Decimal Integer Instructions to reflect source value in VR[VRB] is **valid, is less than zero, and significant digits were not left shifted out and were not truncated**		
bcds. bcdus. bcdsr. bcdtrunc. bcdutrunc.	source is less than zero and significant digits were not left shifted out and were not truncated	(CR field 6)$_{0:3}$ = 1000
I-6.17.CR.25 CR field 6 is set by Decimal Integer Instructions to reflect source value in VR[VRB] is **valid, is less than zero, and significant digits were left shifted out or were truncated**		
bcds. bcdus. bcdsr. bcdtrunc. bcdutrunc.	source is less than zero and significant digits were left shifted out or were truncated	(CR field 6)$_{0:3}$ = 1001

Instructions tested	Compliance conditions	Expected result
I-6.17.CR.26 CR field 6 is set by Decimal Integer Instructions to reflect source value in VR[VRB] has **an invalid encoding of the value**		
bcds. bcdus. bcdsr. bcdtrunc. bcdutrunc.	source has invalid encoding	(CR field 6)$_{0:3}$ = 0001

7.10.2. Correct Computations

Instructions tested	Compliance conditions	Expected result
I-6.17.Computation.1 Result of instructions computation is correct		
Decimal Integer Arithmetic Instructions *Decimal Integer Format Conversion Instructions* *Decimal Integer Sign Manipulation Instructions* *Decimal Integer Shift and Round Instructions* *Decimal Integer Truncate Instructions*	VR[VRA] (when applicable) and VR[VRB] have a *valid encoding of value*	Correct result computation

7.11. Vector Status and Control Register Instructions

Architecture sections:

- I-6.18 Vector Status and Control Register Instructions

Scenario groups:

- Correct Computations

7.11.1. Correct Computations

Instructions tested	Compliance conditions	Expected result
I-6.18.Computation.1 Result of instructions computation is correct		
mtvscr mfvscr		Correct result computation

8. Vector-Scalar Floating-Point Operations (Chapter I.7)

Table of Contents

8.1. VSX Instruction Set

Architecture sections:

- I-7.6 VSX Instruction Set

Scenario groups:

- Setting FPRF
- Setting FR and FI
- Setting CR[BF]
- Setting FPCC
- Setting CR6
- Storage Access Exceptions
- Correct Computations
- Rounding Modes

VSX Scalar Load Instructions: lxsd lxsdx lxsibzx lxsihzx lxsspx lxsiwax lxsiwzx lxssp

VSX Scalar Store Instructions: stxsd stxsdx stxsibx stxsihx stxsspx stxsiwx stxssp

VSX Vector Load Instructions: lxvb16x lxvd2x lxv lxvx lxvh8x lxvw4x

VSX Vector Load and Splat Instructions: lxvdsx lxvwsx

VSX Vector Load with Length Instructions: lxvl lxvll

VSX Vector Store Instructions: stxvb16x stxvd2x stxvh8x stxvw4x stxv stxvx

VSX Vector Store with Length Instructions: stxvl stxvll

VSX Scalar Binary Floating-Point Sign Manipulation Instructions: xsabsdp xsabsqp xscpsgndp xscpsgnqp xsnabsdp xsnabsqp xsnegdp xsnegqp

VSX Vector Binary Floating-Point Sign Manipulation Instructions: xvabsdp xvcpsgndp xvnabsdp xvnegdp xvabssp xvcpsgnsp xvnabssp xvnegsp

VSX Scalar Binary Floating-Point Elementary Arithmetic Instructions: xsadddp xsaddqp[o] xsdivdp xsdivqp[o] xsmuldp xsmulqp[o] xssqrtdp xssqrtqp[o] xssubdp xssubqp[o] xsaddsp xsdivsp xsmulsp xssqrtsp xssubsp

VSX Scalar Binary Floating-Point Multiply-Add-class Instructions: xsmaddadp xsmaddmdp xsmaddqp[o] xsmsubadp xsmsubmdp xsmsubqp[o] xsnmaddadp xsnmaddmdp xsnmaddqp[o]

xsnmsubadp xsnmsubmdp xsnmsubqp[o] xsmaddasp xsmaddmsp xsmsubasp xsmsubmsp xsnmaddasp xsnmaddmsp xsnmsubasp xsnmsubmsp

VSX Vector Binary Floating-Point Elementary Arithmetic Instructions: xvadddp xvdivdp xvmuldp xvsqrtdp xvsubdp xvaddsp xvdivsp xvmulsp xvsqrtsp xvsubsp

VSX Vector Binary Floating-Point Multiply-Add-class Instructions: xvmaddadp xvmaddmdp xvmsubadp xvmsubmdp xvnmaddadp xvnmaddmdp xvnmsubadp xvnmsubmdp xvmaddasp xvmaddmsp xvmsubasp xvmsubmsp xvnmaddasp xvnmaddmsp xvnmsubasp xvnmsubmsp

VSX Scalar Binary Floating-Point Compare Instructions: xscmpodp xscmpoqp xscmpudp xscmpuqp

VSX Scalar Binary Floating-Point Maximum/Minimum Instructions: xsmaxcdp xsmaxdp xsmaxjdp xsmincdp xsmindp xsminjdp

VSX Scalar Software Binary Floating-Point Divide/Square Root Instructions: xsredp xsresp xsrsqrtedp xsrsqrtesp xstdivdp xstsqrtdp

VSX Vector Software Binary Floating-Point Divide/Square Root Instructions: xvredp xvresp xvrsqrtedp xvrsqrtesp xvtdivdp xvtdivsp xvtsqrtdp xvtsqrtsp

VSX Scalar Binary Floating-Point Predicate Compare Instructions: xscmpeqdp xscmpgedp xscmpgtdp

VSX Vector Binary Floating-Point Predicate Compare Instructions: xvcmpeqdp[.] xvcmpgedp[.] xvcmpgtdp[.] xvcmpeqsp[.] xvcmpgesp[.] xvcmpgtsp[.]

VSX Vector Binary Floating-Point Maximum/Minimum Instructions: xvmaxdp xvmindp xvmaxsp xvminsp

VSX Scalar Binary Floating-Point Convert to Shorter Precision Instructions: xscvdphp xscvdpsp xscvdpspn xscvqpdp[o]

VSX Vector Binary Floating-Point Convert to Shorter Precision Instructions: xvcvdpsp xvcvsphp

VSX Scalar Binary Floating-Point Convert to Integer Instructions: xscvdpsxds xscvdpsxws xscvdpuxds xscvdpuxws xscvqpsdz xscvqpswz xscvqpudz xscvqpuwz

VSX Scalar Binary Floating-Point Convert to Longer Precision Instructions: xscvdpqp xscvhpdp xscvspdp xscvspdpn

VSX Vector Binary Floating-Point Convert to Longer Precision Instructions: xvcvhpsp xvcvspdp

VSX Scalar Binary Floating-Point Convert from Integer Instructions: xscvsdqp xscvsxddp xscvudqp xscvuxddp xscvsxdsp xscvuxdsp

VSX Vector Binary Floating-Point Convert to Integer Instructions: xvcvdpsxds xvcvdpsxws xvcvdpuxds xvcvdpuxws xvcvspsxds xvcvspsxws xvcvspuxds xvcvspuxws

VSX Vector Binary Floating-Point Convert from Integer Instructions: xvcvsxddp xvcvsxwdp xvcvuxddp xvcvuxwdp xvcvsxdsp xvcvsxwsp xvcvuxdsp xvcvuxwsp

VSX Scalar Binary Floating-Point Round to Integral Instructions: xsrdpi xsrdpic xsrdpim xsrdpip xsrdpiz xsrqpi xsrqpix

VSX Scalar Binary Floating-Point Round to Shorter Precision Instructions: xsrqpxp xsrsp

VSX Scalar Binary Floating-Point Math Support Instructions: xscmpexpdp xscmpexpqp xsiexpdp xsiexpqp xststdcdp xststdcqp xststdcsp xsxexpdp xsxexpqp xsxsigdp xsxsigqp

VSX Vector Binary Floating-Point Math Support Instructions: xviexpdp xviexpsp xvtstdcdp xvtstdcsp xvxexpdp xvxexpsp xvxsigdp xvxsigsp

VSX Vector Binary Floating-Point Round to Integral Instructions: xvrdpi xvrdpic xvrdpim xvrdpip xvrdpiz xvrspi xvrspic xvrspim xvrspip xvrspiz

VSX Vector Logical Instructions: xxland xxlandc xxlnor xxlor xxlxor xxleqv xxlnand xxlorc

VSX Vector Select Instruction: xxsel

VSX Vector Byte-Reverse Instructions: xxbrd xxbrh xxbrq xxbrw

VSX Vector Insert/Extract Instructions: xxextractuw xxinsertw

VSX Vector Merge Instructions: xxmrghw xxmrglw

VSX Vector Splat Instructions: xxspltib xxspltw

VSX Vector Permute Instructions: xxpermdi xxperm xxpermr

VSX Vector Shift Left Double Instruction: xxsldwi

8.1.1. Setting FPRF

Note: The tests in this section assume no exceptions occurrences

VSX arithmetic, rounding, and conversion instructions: *VSX Scalar BFP Round to Integral Instructions, VSX Scalar BFP Round to Shorter Precision Instructions, VSX Scalar BFP Elementary Arithmetic Instructions, VSX Scalar BFP Multiply-Add-class Instructions, VSX Scalar BFP Convert from Integer Instructions, VSX Scalar BFP Convert to Shorter Precision Instructions, VSX Scalar BFP Convert to Longer Precision Instructions, and VSX Scalar Software BFP Divide/Square Root Instructions*

Instructions tested	**Compliance conditions**	**Expected result**
I-7.6.FPRF.1 Arithmetic/rounding/conversion instructions set FPRF correctly for Quiet NaN result		
VSX arithmetic, rounding, and conversion instructions	Result value class is Quiet NaN	FPRF = 10001
I-7.6.FPRF.2 Arithmetic/rounding/conversion instructions set FPRF correctly for -Infinity result		
VSX arithmetic, rounding, and conversion instructions	Result value class is -Infinity	FPRF = 01001
I-7.6.FPRF.3 Arithmetic/rounding/conversion instructions set correctly for Normalized result		
VSX arithmetic, rounding, and conversion instructions	Result value class is Normalized Number	FPRF = 01000
I-7.6.FPRF.4 Arithmetic/rounding/conversion instructions set FPRF correctly for Denormalized result		
VSX arithmetic, rounding, and conversion instructions	Result value class is Denormalized Number	FPRF = 11000
I-7.6.FPRF.5 Arithmetic/rounding/conversion instructions set FPRF correctly for Zero result		
VSX arithmetic, rounding, and conversion instructions	Result value class is Zero	FPRF = 10010
I-7.6.FPRF.6 Arithmetic/rounding/conversion instructions set FPRF correctly for +Zero result		

Instructions tested	Compliance conditions	Expected result
VSX arithmetic, rounding, and conversion instructions	Result value class is +Zero	FPRF = 00010
I-7.6.FPRF.7 Arithmetic/rounding/conversion instructions set FPRF correctly for +Denormalized result		
VSX arithmetic, rounding, and conversion instructions	Result value class is +Denormalized Number	FPRF = 10100
I-7.6.FPRF.8 Arithmetic/rounding/conversion instructions set FPRF correclyt for +Normalized result		
VSX arithmetic, rounding, and conversion instructions	Result value class is +Normalized Number	FPRF = 00100
I-7.6.FPRF.9 Arithmetic/rounding/conversion instructions set FPRF correctly for +Infinity result		
VSX arithmetic, rounding, and conversion instructions	Result value class is +Infinity	FPRF = 00101

8.1.2. Setting FR and FI

VSX potentially fraction rounding instructions: *VSX Scalar BFP Round to Integral Instructions, VSX Scalar BFP Round to Shorter Precision Instructions, VSX Scalar BFP Elementary Arithmetic Instructions, VSX Scalar BFP Convert to Shorter Precision Instructions, VSX Scalar BFP Multiply-Add-class Instructions, VSX Scalar BFP Convert to Integer Instructions, VSX Scalar BFP Convert from Integer Instructions, VSX Scalar BFP Convert to Longer Precision Instructions, and VSX Scalar Software BFP Divide/Square Root Instructions*

VSX convert / round to double-precision integer or quad-precision instructions: xscvdpqp xscvhpdp xscvspdp xsrdpi xsrdpim xsrdpip xsrdpiz xsrqpi xscvsdqp xscvudqp

VSX convert quad-precision to integer instructions: xscvqpsdz xscvqpswz xscvqpudz xscvqpuwz

Notes:

The following instructions are not included in the list of *VSX potentially fraction rounding instructions*, since they set FR and FI to an undefined value: xsresp xsrsqrtesp xsredp xsrsqrtedp

Instructions tested	Compliance conditions	Expected result
I-7.6.FR.1 FR is set to 1 when the fraction is incremented during rounding		
VSX potentially fraction rounding instructions	The fraction is incremented during rounding of the intermediate result No exception occurs	FR=1
I-7.6.FR.2 *VSX convert / round to double-precision integer or quad-precision instructions* or *VSX convert quad-precision to integer instructions* set FR to zero even if fraction is incremented during rounding		
VSX convert / round to double-precision integer or quad-precision instructions *VSX convert quad-precision to integer instructions*	The fraction is incremented during rounding of the intermediate result No exception occurs	FR=0
I-7.6.FR.3 FR is not sticky		
VSX potentially fraction rounding instructions	The fraction is not incremented during rounding of the intermediate result	FR=0
I-7.6.FR.4 FI is set to 1 when result is inexact		
VSX potentially fraction rounding instructions	The rounded result differs from the intermediate result (this implies that Inexact Exception occurs) No exception other than Inexact Exception occurs	FI=1

Instructions tested	Compliance conditions	Expected result
I-7.6.FR.5 *VSX convert / round to double-precision integer or quad-precision instructions* set FI to zero even when result is inexact		
VSX convert / round to double-precision integer or quad-precision instructions	The rounded result differs from the intermediate result (this implies that Inexact Exception occurs) No exception other than Inexact Exception occurs	FI=0
I-7.6.FR.6 FI is not sticky		
VSX potentially fraction rounding instructions	The rounded result is equal to the intermediate result	FI=0

8.1.3. Setting CR[BF]

VSX Scalar Compare Double-Precision Instructions: xscmpexpdp xscmpodp xscmpudp

VSX Scalar Compare Quad-Precision Instructions: xscmpexpqp xscmpoqp xscmpuqp

Instructions tested	Compliance conditions	Expected result
I-7.6.CRBF.1 Setting CR field BF in double-precision test for software divide instructions when the first source is Infinity		
xstdivdp xvtdivdp	VSR[XA].doubleword[0] is Infinity	CR[BF] = 1110
I-7.6.CRBF.2 Setting CR field BF in double-precision test for software divide instructions when the first source is NaN and the 2nd source is zero, an infinity, or a denormalized value.		
xstdivdp xvtdivdp	VSR[XA].doubleword[0] is NaN VSR[XB].doubleword[0] is zero, an infinity, or a denormalized value.	CR[BF] = 1110
I-7.6.CRBF.3 Setting CR field BF in double-precision test for software divide instructions when the first source is NaN and the 2nd source is not zero, an infinity, or a denormalized value.		
xstdivdp xvtdivdp	VSR[XA].doubleword[0] is NaN VSR[XB].doubleword[0] is not zero, an infinity, or a denormalized value.	CR[BF] = 1010
I-7.6.CRBF.4 Ssetting CR field BF in double-precision test for software divide instructions when the 2nd source is zero or an infinity		
xstdivdp xvtdivdp	VSR[XB].doubleword[0] is zero or an infinity	CR[BF] = 1110
I-7.6.CRBF.5 Setting CR field BF in double-precision test for software divide instructions when the 2nd source is NaN and the 1st source is not Infinity		
xstdivdp xvtdivdp	VSR[XB].doubleword[0] is NaN VSR[XA].doubleword[0] is not Infinity	CR[BF] = 1010
I-7.6.CRBF.6 Setting CR field BF in double-precision test for software divide instructions when the unbiased exponent ofthe 2nd;is less than or equal to -1022 and the 1 stsource is not Infinity		
xstdivdp xvtdivdp	the unbiased exponent ofVSR[XB].doubleword[0] is less than or equal to -1022 VSR[XA].doubleword[0] is not Infinity	CR[BF] = 1010
I-7.6.CRBF.7 Setting CR field BF in double-precision test for software divide instructions when the unbiased exponent ofthe 2nd;is greater than or equal to 1021 and the 1 stsource is not Infinity		
xstdivdp xvtdivdp	the unbiased exponent ofVSR[XB].doubleword[0] is greater than or equal to 1021 VSR[XA].doubleword[0] is not Infinity	CR[BF] = 1010
I-7.6.CRBF.8 Setting CR field BF in double-precision test for software divide instructions when the 1st source is not Infinity or Zero and the difference between the unbiased exponents is greater than or equal to 1023		

Instructions tested	Compliance conditions	Expected result
xstdivdp xvtdivdp	VSR[XA].doubleword[0] is not Infinity VSR[XA].doubleword[0] is not zero The difference, the unbiased exponent ofVSR[XA].doubleword[0] - the unbiased exponent ofVSR[XB].doubleword[0], is greater than or equal to 1023.	CR[BF] = 1010
I-7.6.CRBF.9 Setting CR field BF in double-precision test for software divide instructions when the 1st source is not Infinity or Zero and the difference between the unbiased exponents is less than or equal to -1021		
xstdivdp xvtdivdp	VSR[XA].doubleword[0] is not Infinity VSR[XA].doubleword[0] is not zero The difference, the unbiased exponent ofVSR[XA].doubleword[0] - the unbiased exponent ofVSR[XB].doubleword[0], is less than or equal to -1021.	CR[BF] = 1010
I-7.6.CRBF.10 Setting CR field BF in double-precision test for software divide instructions when the 1st source is not Infinity or Zero and its unbiased exponent is less than or equal to -970.		
xstdivdp xvtdivdp	VSR[XA].doubleword[0] is not Infinity VSR[XA].doubleword[0] is not zero the unbiased exponent ofVSR[XA].doubleword[0] is less than or equal to -970. VSR[XB].doubleword[0] is not zero, an infinity, or a denormalized value.	CR[BF] = 1010
I-7.6.CRBF.11 Setting CR field BF in double-precision test for software divide instructions when the 1st source is not Infinity and the 2nd source is zero, an infinity, or a denormalized value.		
xstdivdp xvtdivdp	VSR[XA].doubleword[0] is not Infinity VSR[XB].doubleword[0] is zero, an infinity, or a denormalized value.	CR[BF] = 1100
I-7.6.CRBF.12 Setting CR field BF in double-precision test for software square root instructions when the source is zero or an infinity		
xstsqrtdp xvtsqrtdp	VSR[XB].doubleword[0] is zero or an infinity	CR[BF] = 1110
I-7.6.CRBF.13 Setting CR field BF in double-precision test for software square root instructions when the source is positive denormalized value and its unbiased exponent is less than or equal to -970.		
xstsqrtdp xvtsqrtdp	VSR[XB].doubleword[0] is positive denormalized value The unbiased exponent ofVSR[XB].doubleword[0] is less than or equal to -970.	CR[BF] = 1110
I-7.6.CRBF.14 Setting CR field BF in double-precision test for software square root instructions when the source is positive denormalized value and its unbiased exponent is greater than -970.		
xstsqrtdp xvtsqrtdp	VSR[XB].doubleword[0] is positive denormalized value The unbiased exponent ofVSR[XB].doubleword[0] is greater than -970.	CR[BF] = 1100
I-7.6.CRBF.15 Setting CR field BF in double-precision test for software square root instructions when the source is negative denormalized value		
xstsqrtdp xvtsqrtdp	VSR[XB].doubleword[0] is negative denormalized value	CR[BF] = 1110

Instructions tested	Compliance conditions	Expected result
I-7.6.CRBF.16 Setting CR field BF in double-precision test for software square root instructions when the source is negative normalized value or NaN		
xstsqrtdp xvtsqrtdp	VSR[XB].doubleword[0] is negative normalized or NaN	CR[BF] = 1010
I-7.6.CRBF.17 Setting CR field BF in double-precision test for software square root instructions when the unbiased exponent of the source is less than or equal to -970.		
xstsqrtdp xvtsqrtdp	The unbiased exponent ofVSR[XB].doubleword[0] is less than or equal to -970. VSR[XB].doubleword[0] is not zero, an Infinity or denormalized number	CR[BF] = 1010
I-7.6.CRBF.18 Setting CR field BF in single-precision test for software divide instructions when the first source is Infinity for each word in the vector (i: 0-3)		
xvtdivsp	VSR[XA].word[i] is Infinity	CR[BF] = 1110
I-7.6.CRBF.19 Setting CR field BF in single-precision test for software divide instructions when the first source is NaN and the 2nd source is zero, an infinity, or a denormalized value. for each word in the vector (i: 0-3)		
xvtdivsp	VSR[XA].word[i] is NaN VSR[XB].word[i] is zero, an infinity, or a denormalized value.	CR[BF] = 1110
I-7.6.CRBF.20 Setting CR field BF in single-precision test for software divide instructions when the first source is NaN and the 2nd source is not zero, an infinity, or a denormalized value. for each word in the vector (i: 0-3)		
xvtdivsp	VSR[XA].word[i] is NaN VSR[XB].word[i] is not zero, an infinity, or a denormalized value.	CR[BF] = 1010
I-7.6.CRBF.21 Setting CR field BF in single-precision test for software divide instructions when the 2nd source is zero or an infinity for each word in the vector (i: 0-3)		
xvtdivsp	VSR[XB].word[i] is zero or an infinity	CR[BF] = 1110
I-7.6.CRBF.22 Setting CR field BF in single-precision test for software divide instructions when the 2nd source is NaN and the 1st source is not Infinity, for each word in the vector (i: 0-3)		
xvtdivsp	VSR[XB].word[i] is NaN VSR[XA].word[i] is not Infinity	CR[BF] = 1010
I-7.6.CRBF.23 Setting CR field BF in single-precision test for software divide instructions when the unbiased exponent ofthe 2nd is less than or equal to -126 and the 1st source is not Infinity, for each word in the vector (i: 0-3)		
xvtdivsp	the unbiased exponent ofVSR[XB].word[i] is less than or equal to -126 VSR[XA].word[i] is not Infinity	CR[BF] = 1010
I-7.6.CRBF.24 Setting CR field BF in single-precision test for software divide instructions when the unbiased exponent ofthe 2nd source is greater than or equal to 125and the 1st source is not Infinity, for each word in the vector (i: 0-3)		
xvtdivsp	the unbiased exponent ofVSR[XB].word[i] is greater than or equal to 125 VSR[XA].word[i] is not Infinity	CR[BF] = 1010
I-7.6.CRBF.25 Setting CR field BF in single-precision test for software divide instructions when the 1st source is not Infinity or Zero and the difference between the unbiased exponents is greater than or equal to 127, for each word in the vector (i: 0-3)		
xvtdivsp	VSR[XA].word[i] is not Infinity VSR[XA].word[i] is not zero The difference, the unbiased exponent ofVSR[XA].word[i] - the unbiased exponent ofVSR[XB].word[i], is greater than or equal to 127.	CR[BF] = 1010

Instructions tested	Compliance conditions	Expected result
I-7.6.CRBF.26 Setting CR field BF in single-precision test for software divide instructions when the 1st source is not Infinity or Zero and the difference between the unbiased exponents is less than or equal to -125, for each word in the vector (i: 0-3)		
xvtdivsp	VSR[XA].word[i] is not Infinity VSR[XA].word[i] is not zero The difference, the unbiased exponent ofVSR[XA].word[i] - the unbiased exponent ofVSR[XB].word[i], is less than or equal to -125.	CR[BF] = 1010
I-7.6.CRBF.27 Setting CR field BF in single-precision test for software divide instructions when the 1st source is not Infinity or Zero and its unbiased exponents is less than or equal to -103, for each word in the vector (i: 0-3)		
xvtdivsp	VSR[XA].word[i] is not Infinity VSR[XA].word[i] is not zero The unbiased exponent ofVSR[XA].word[i] is less than or equal to -103. VSR[XB].word[i] is not zero, an infinity, or a denormalized value.	CR[BF] = 1010
I-7.6.CRBF.28 Setting CR field BF in single-precision test for software divide instructions when the 1st source is not Infinity or Zero and the 2nd source is zero, an infinity, or a denormalized value, for each word in the vector (i: 0-3)		
xvtdivsp	VSR[XA].doubleword[0] is not Infinity VSR[XB].doubleword[0] is zero, an infinity, or a denormalized value.	CR[BF] = 1100
I-7.6.CRBF.29 Setting CR field BF in single-precision test for software square root instructions when the source is zero or an infinity for each i from 0 3		
xvtsqrtsp	VSR[XB].word[i] is zero or an infinity	CR[BF] = 1110
I-7.6.CRBF.30 Setting CR field BF in single-precision test for software square root instructions when the source is positive denormalized value and its unbiased exponent is less than or equal to -103, for each i from 0 3		
xvtsqrtsp	VSR[XB].word[i] is positive denormalized value The unbiased exponent ofVSR[XB].word[i] is less than or equal to -103.	CR[BF] = 1110
I-7.6.CRBF.31 Setting CR field BF in single-precision test for software square root instructions when the source is positive denormalized value and its unbiased exponent is greater than -103, for each i from 0 3		
xvtsqrtsp	VSR[XB]. word[i] is positive denormalized value The unbiased exponent ofVSR[XB]. word[i] is greater than -103.	CR[BF] = 1100
I-7.6.CRBF.32 Setting CR field BF in single-precision test for software square root instructions when the source is negative denormalized value, for each i from 0 3		
xvtsqrtsp	VSR[XB]. word[i] is negative denormalized value	CR[BF] = 1110
I-7.6.CRBF.33 Setting CR field BF in single-precision test for software square root instructions when the source is negative normalized value or NaN, for each i from 0 3		
xvtsqrtsp	VSR[XB]. word[i] is negative normalized or NaN	CR[BF] = 1010
I-7.6.CRBF.34 Setting CR field BF in single-precision test for software square root instructions when the unbiased exponent of the source is less than or equal to -103, for each i from 0 3		
xvtsqrtsp	The unbiased exponent ofVSR[XB].word[i] is less than or equal to -103. VSR[XB]. word[i] is not zero, an Infinity or denormalized number	CR[BF] = 1010

Instructions tested	Compliance conditions	Expected result
I-7.6.CRBF.35 For the compare instructions if either of the operands is a NaN, either quiet or signaling, CR field BF is set to reflect unordered.		
VSX Scalar Compare Double-Precision Instructions *VSX Scalar Compare Quad-Precision Instructions*	One of the operands is NaN	CR[BF] = 0001
I-7.6.CRBF.36 Setting CR field BF for compare instructions when the compare result is **less than**		
VSX Scalar Compare Double-Precision Instructions *VSX Scalar Compare Quad-Precision Instructions*	Neither of the operands is NaN operand1 < operand2	CR[BF] = 1000
I-7.6.CRBF.37 Setting CR field BF for compare instructions when the compare result is **greater than**		
VSX Scalar Compare Double-Precision Instructions *VSX Scalar Compare Quad-Precision Instructions*	Neither of the operands is NaN operand1 > operand2	CR[BF] = 0100
I-7.6.CRBF.38 Setting CR field BF for compare instructions when the compare result is **equal to**		
VSX Scalar Compare Double-Precision Instructions *VSX Scalar Compare Quad-Precision Instructions*	Neither of the operands is NaN operand1 = operand2	CR[BF] = 0010
I-7.6.CRBF.39 Setting CR field BF for VSX Scalar Test Data Class Double-Precision or Quad-Precision instructions		
xststdcdp xststdcqp		CR[BF] bit 0 set to sign bit of source CR[BF] bit 1 set to 0 CR[BF] bit 2 set to indicate if data class of source matches data class specified by DCMX CR[BF] bit 3 set to 0
I-7.6.CRBF.39b Setting CR field BF for VSX Scalar Test Data Class Single-Precision instruction		
xststdcsp		CR[BF] bit 0 set to sign bit of source CR[BF] bit 1 set to 0 CR[BF] bit 2 set to indicate if data class of source matches data class specified by DCMX CR[BF] bit 3 set to indicate if src is not representable in single-precision format

8.1.4. Setting FPCC

VSX Scalar Compare Double-Precision Instructions: xscmpexpdp xscmpodp xscmpudp

VSX Scalar Compare Quad-Precision Instructions: xscmpexpqp xscmpoqp xscmpuqp

Instructions tested	Compliance conditions	Expected result
I-7.6.FPCC.1 For the compare instructions if either of the operands is a NaN, either quiet or signaling, FPCC is set to reflect unordered.		
VSX Scalar Compare Double-Precision Instructions	One of the operands is NaN	FPCC = 0001

Instructions tested	Compliance conditions	Expected result
VSX Scalar Compare Quad-Precision Instructions		
I-7.6.FPCC.2 Setting FPCC field for compare instructions when the compare result is **less than**		
VSX Scalar Compare Double-Precision Instructions *VSX Scalar Compare Quad-Precision Instructions*	Neither of the operands is NaN operand1 < operand2	FPCC = 1000
I-7.6.FPCC.3 Setting FPCC for compare instructions when the compare result is **greater than**		
VSX Scalar Compare Double-Precision Instructions *VSX Scalar Compare Quad-Precision Instructions*	Neither of the operands is NaN operand1 > operand2	FPCC = 0100
I-7.6.FPCC.4 Setting FPCC for compare instructions when the compare result is **equal to**		
VSX Scalar Compare Double-Precision Instructions *VSX Scalar Compare Quad-Precision Instructions*	Neither of the operands is NaN operand1 = operand2	FPCC = 0010
I-7.6.FPCC.5 Setting FPCC for VSX Scalar Test Data Class Double-Precision or Quad-Precision instructions		
xststdcdp xststdcqp		FPCC[BF] bit 0 set to sign bit of source FPCC[BF] bit 1 set to 0 FPCC[BF] bit 2 set to indicate if data class of source matches data class specified by DCMX FPCC[BF] bit 3 set to 0
I-7.6.FPCC.5b Setting FPCC for VSX Scalar Test Data Class Single-Precision instruction		
xststdcsp		FPCC[BF] bit 0 set to sign bit of source FPCC[BF] bit 1 set to 0 FPCC[BF] bit 2 set to indicate if data class of source matches data class specified by DCMX FPCC[BF] bit 3 set to indicate if src is not representable in single-precision format

8.1.5. Setting CR6

VSX recording instructions: xvcmpeqdp. xvcmpgedp. xvcmpgtdp. xvcmpeqsp. xvcmpgesp. xvcmpgtsp.

Instructions tested	Compliance conditions	Expected result
I-7.6.CR6.1 CR field 6 is set by *VSX recording instructions* to reflect the result of the comparison when it is **true for all the vector element pairs**		
VSX recording instructions	The result of comparison is true for all the element pairs	$(\text{CR field } 6)_{0:3} = 1000$
I-7.6.CR6.2 CR field 6 is set by *VSX recording instructions* to reflect the result of the comparison when it is **false for all the vector element pairs**		
VSX recording instructions	The result of comparison is false for all the element pairs	$(\text{CR field } 6)_{0:3} = 0010$
I-7.6.CR6.3 CR field 6 is set by *VSX recording instructions* to reflect the result of the comparison when it **is not true for all element pairs and not false for all element pairs**		

Instructions tested	Compliance conditions	Expected result
VSX recording instructions	the result of the comparison when it is not true for all element pairs neither false for all the element pairs	(CR field 6)$_{0:3}$ = 0000

8.1.6. Storage Access Exceptions

VSX Storage Access Instructions: *VSX Scalar Load Instructions , VSX Scalar Store Instructions , VSX Vector Load Instructions , VSX Vector Load and Splat Instructions , VSX Vector Store Instructions , VSX Vector Load with Length Instructions , VSX Vector Store with Length Instructions*

Instructions tested	Compliance conditions	Expected result
I-7.6.1.1.StorageEx.1 A system data storage error handler will be invoked when an instruction attempts to access unavailable storage		
VSX Storage Access Instructions	The program is not allowed to modify the target storage (Store only) or the program attempts to access storage that is unavailable.	system data storage error handler is invoked

8.1.7. Correct Computations

Guideline: for *VSX Storage Access Instructions*, the scenario should be tested in both Big-Endian and Little-Endian modes

Instructions tested	Compliance conditions	Expected result
I-7.6.Comp.1 VSX instructions perform correct computations		
All VSX instructions	No exception occurs	Correct result of operation

8.1.8. Rounding Modes

Instructions tested	Compliance conditions	Expected result
I-7.4.Round.1 Round to Nearest mode		
Representative of *VSX potentially fraction rounding instructions*	RN = 00 No exception occurs Result is numeric Numeric operands	Rounding is performed according to Round to Nearest mode
I-7.4.Round.2 Round toward Zero mode		
Representative of *VSX potentially fraction rounding instructions*	RN = 01 No exception occurs Result is numeric Numeric operands	Rounding is performed according to Round toward Zero mode
I-7.4.Round.3 Round toward +Infinity mode		
Representative of *VSX potentially fraction rounding instructions*	RN = 10 No exception occurs Result is numeric Numeric operands	Rounding is performed according to Round toward +Infinity mode
I-7.4.Round.4 Round toward Infinity mode		
Representative of *VSX potentially fraction rounding instructions*	RN = 11	Rounding is performed according to Round toward -Infinity mode

Instructions tested	Compliance conditions	Expected result
	No exception occurs Result is numeric Numeric operands	

8.2. VSX Floating-Point Exceptions

Architecture sections:

- I-7.4 VSX Floating-Point Exceptions

8.2.1. Invalid Operation Exception

Scenario groups:

- Setting exception bits
- Keeping exception bits unchanged
- Actions taken when the exception is enabled
- Action taken when the exception is disabled

Condition for Invalid Operation Exception **SNaN**: Any floating-point operation on a Signaling NaN.

Condition for Invalid Operation Exception **InfinityInfinity**: Magnitude subtraction of infinities occurs.

Condition for Invalid Operation Exception **InfinityInfinity**: Floating-point division of infinity by infinity occurs.

Condition for Invalid Operation Exception **ZeroZero**: Floating-point division of zero by zero occurs.

Condition for Invalid Operation Exception **Infinity Zero**: Floating-point multiplication of infinity by zero occurs.

Condition for Invalid Operation Exception **Invalid Compare**: Floating-point ordered comparison involving a NaN.

Condition for Invalid Operation Exception **Invalid Square Root**: Floating-point square root or reciprocal square root of a nonzero negative number.

Condition for Invalid Operation Exception **Invalid Integer Convert**: Floating-point-to-integer convert involving a number too large in magnitude to be represented in the target format, or involving an infinity or a NaN.

Condition for Invalid Operation Exception **Software Defined**: An **mtfsfi, mtfsf,** or **mtfsb1** instruction is executed that sets VXSOFT to 1

8.2.1.1. Setting exception bits

VSX potential-SNaN-operand instructions: xsredp xsresp xsrsqrtedp xsrsqrtesp xscvdpqp xscvhpdp xscvspdp, *VSX Scalar BFP Elementary Arithmetic Instructions, VSX Vector BFP Elementary Arithmetic Instructions, VSX Scalar BFP Multiply-Add-class Instructions,* xvredp xvrsqrtedp xvresp xvrsqrtesp, *VSX Vector BFP Multiply-Add-class Instructions, VSX Scalar BFP Compare Instructions, VSX Scalar BFP Maximum/Minimum Instructions, VSX Scalar BFP Predicate Compare Instructions, VSX Vector BFP Predicate Compare Instructions, VSX Vector BFP Maximum/Minimum Instructions,*

xscvdphp xscvdpsp xscvqpdp[o], *VSX Vector BFP Convert to Shorter Precision Instructions, VSX Scalar BFP Convert to Integer Instructions, VSX Vector BFP Convert to Integer Instructions, VSX Scalar BFP Round to Integral Instructions, VSX Vector BFP Round to Integral Instructions, VSX Scalar BFP Round to Shorter Precision Instructions, VSX Vector BFP Convert to Longer Precision Instructions*

VSX addition/subtraction instructions: xsadddp xssubdp xsaddsp xssubsp xsaddqp[o] xssubqp[o], *VSX Scalar BFP Multiply-Add-class Instructions,* xvadddp xvsubdp xvaddsp xvsubsp, *VSX Vector BFP Multiply-Add-class Instructions*

VSX multiplication instructions: *VSX Vector BFP Multiply-Add-class Instructions,* xvmuldp xvmulsp xsmuldp xsmulqp[o] xsmulsp, *VSX Scalar BFP Multiply-Add-class Instructions*

VSX conversion to integer instructions: *VSX Scalar BFP Convert to Integer Instructions, VSX Vector BFP Convert to Integer Instructions*

Instructions tested	**Compliance conditions**	**Expected result**
I-7.4.1.Invalid.1 VXSNAN is set to 1 when one of the operands is an SNaN		
VSX potential-SNaN-operand instructions	*Condition for Invalid Operation Exception* ***SNaN*** occurs	VXSNAN = 1
I-7.4.1.Invalid.2 Add/subtract operations set VXISI to 1 when magnitude subtraction of infinities occurs		
VSX addition/subtraction instructions	*Condition for Invalid Operation Exception* ***Infinity-Infinity*** occurs	VXISI = 1
I-7.4.1.Invalid.3 Division operations set VXIDI to 1 when division of infinity by infinity occurs		
xsdivdp xsdivsp xsdivqp[o] xvdivdp xvdivsp	*Condition for Invalid Operation Exception* ***Infinity÷Infinity*** occurs	VXIDI = 1
I-7.4.1.Invalid.4 Division operations set VXZDZ to 1 when division of zero by zero occurs		
xsdivdp xsdivsp xsdivqp[o] xvdivdp xvdivsp	*Condition for Invalid Operation Exception* ***Zero÷Zero*** occurs	VXZDZ = 1
I-7.4.1.Invalid.5 Multiplication operations set VXIMZ to 1 when multiplication of infinity by zero occurs		
VSX multiplication instructions	*Condition for Invalid Operation Exception* ***Infinity x Zero*** occurs	VXIMZ = 1
I-7.4.1.Invalid.6 Ordered comparison sets VXVC to 1 when the comparison involves a NaN		
xscmpgedp xscmpgtdp xscmpodp xscmpoqp xvcmpgedp[.] xvcmpgtdp[.] xvcmpgesp[.] xvcmpgtsp[.]	*Condition for Invalid Operation Exception* ***Invalid Compare*** occurs	VXVC = 1
I-7.4.1.Invalid.7 Square root operations set VXSQRT to 1 when applied to a negative nonzero number		
xsrsqrtedp xssqrtdp xssqrtqp[o] xsrsqrtesp xssqrtsp xvrsqrtedp xvsqrtdp xvrsqrtesp xvsqrtsp	*Condition for Invalid Operation Exception* ***Invalid Square Root*** occurs	VXSQRT = 1
I-7.4.1.Invalid.8 Integer conversion operations set VXCVI to 1 when it involves a too large number or infinity or a NaN		
VSX conversion to integer instructions	*Condition for Invalid Operation Exception* ***Invalid Integer Convert*** occurs	VXCVI = 1
I-7.4.1.Invalid.9 An Invalid Operation exception occurs when an **mtfsfi, mtfsf,** or **mtfsb1** sets VXSOFT to 1 while executing		
mtfsfi mtfsf mtfsb1	*Condition for Invalid Operation Exception* ***Software Defined*** occurs	VXSOFT = 1
I-7.4.1.Invalid.10 VX is set to 1 if any Invalid Operation Exception occurs		
Representative of the instructions tested in I-7.4.1.Invalid.1 to I-7.4.1.Invalid.9	VXSNAN | VXISI | VXIDI | VXZDZ | VXIMZ | VXVC | VXSQRT | VXCVI | VXSOFT = 1	VX = 1
I-7.4.1.Invalid.11 VX is not sticky		
Representative of the instructions tested in I-7.4.1.Invalid.1 to I-7.4.1.Invalid.9	VX = 1 before instruction execution	VX=0

Instructions tested	Compliance conditions	Expected result
	No Invalid Operation exception occurs	
I-7.4.1.Invalid.12 VX stays zero when no Invalid Operation exception occurs		
Representative of the instructions tested in I-7.4.1.Invalid.1 to I-7.4.1.Invalid.9	VX = 0 before instruction execution No Invalid Operation exception occurs	VX=0

8.2.1.2. Keeping exception bits unchanged

Instructions tested	Compliance conditions	Expected result
I-7.4.1.InvalidConst.1 VXSNAN is sticky		
Representative of *VSX potential-SNaN-operand instructions*	VXSNAN=1 before instruction execution *Condition for Invalid Operation Exception* ***SNaN*** occurs does **not** occur.	VXSNAN = 1
I-7.4.1.InvalidConst.2 VXSNAN stays zero when exception condition **does not occur**		
Representative of *VSX potential-SNaN-operand instructions*	VXSNAN=0 before instruction execution *Condition for Invalid Operation Exception* ***SNaN*** occurs does **not** occur.	VXSNAN = 0
I-7.4.1.InvalidConst.3 VXISI is sticky		
Representative of *VSX addition/subtraction instructions*	VXISI = 1 before instruction execution *Condition for Invalid Operation Exception* ***Infinity-Infinity*** does **not** occur	VXISI = 1
I-7.4.1.InvalidConst.4 VXISI stays zero when exception condition **does not occur**		
Representative of *VSX addition/subtraction instructions*	VXISI = 0 before instruction execution *Condition for Invalid Operation Exception* ***Infinity-Infinity*** does **not** occur	VXISI = 0
I-7.4.1.InvalidConst.5 VXIDI is sticky		
Representative of xsdivdp xsdivsp xsdivqp[o] xvdivdp xvdivsp	VXIDI = 1 before instruction execution *Condition for Invalid Operation Exception* ***Infinity÷Infinity*** does **not** occur	VXIDI = 1
I-7.4.1.InvalidConst.6 VXIDI stays zero when exception condition **does not occur**		
Representative of: xsdivdp xsdivsp xsdivqp[o] xvdivdp xvdivsp	VXIDI = 0 before instruction execution *Condition for Invalid Operation Exception* ***Infinity÷Infinity*** does **not** occur	VXIDI = 0
I-7.4.1.InvalidConst.7 VXZDZ is sticky		
Representative of: xsdivdp xsdivsp xsdivqp[o] xvdivdp xvdivsp	VXZDZ = 1 before instruction execution *Condition for Invalid Operation Exception* ***Zero÷Zero*** does **not** occur	VXZDZ = 1
I-7.4.1.InvalidConst.8 VXZDZ stays zero when exception condition **does not occur**		
Representative of: xsdivdp xsdivsp xsdivqp[o] xvdivdp xvdivsp	VXZDZ = 0 before instruction execution *Condition for Invalid Operation Exception* ***Zero÷Zero*** does **not** occur	VXZDZ = 0
I-7.4.1.InvalidConst.9 VXIMZ is sticky		
Representative of *VSX multiplication instructions*	VXIMZ = 1 before instruction execution *Condition for Invalid Operation Exception* ***Infinity x Zero*** does **not** occur	VXIMZ = 1
I-7.4.1.InvalidConst.10 VXIMZ stays zero when exception condition **does not occur**		

Instructions tested	Compliance conditions	Expected result
Representative of *VSX multiplication instructions*	VXIMZ = 0 before instruction execution *Condition for Invalid Operation Exception* ***Infinity x Zero*** does **not** occur	VXIMZ = 0
I-7.4.1.InvalidConst.11 VXVC is sticky		
Representative of xscmpgedp xscmpgt-dp xscmpodp xscmpoqp xvcmpgedp[.] xvcmpgtdp[.] xvcmpgesp[.] xvcmpgtsp[.]	VXVC = 1 before instruction execution *Condition for Invalid Operation Exception* ***Invalid Compare*** does **not** occur	VXVC = 1
I-7.4.1.InvalidConst.12 VXVC stays zero when exception condition **does not occur**		
Representative of xscmpgedp xscmpgt-dp xscmpodp xscmpoqp xvcmpgedp[.] xvcmpgtdp[.] xvcmpgesp[.] xvcmpgtsp[.]	VXVC = 0 before instruction execution *Condition for Invalid Operation Exception* ***Invalid Compare*** does **not** occur	VXVC = 0
I-7.4.1.InvalidConst.13 VXSQRT is sticky		
Representative of xsrsqrtedp xssqrtdp xsrsqrtesp xssqrtsp xssqrtqp[o] xvrsqrtedp xvsqrtdp xvrsqrtesp xvsqrtsp	VXSQRT = 1 before instruction execution *Condition for Invalid Operation Exception* ***Invalid Square Root*** does **not** occur	VXSQRT = 1
I-7.4.1.InvalidConst.14 VXSQRT stays zero when exception condition **does not occur**		
Representative of xsrsqrtedp xssqrtdp xsrsqrtesp xssqrtsp xssqrtqp[o] xvrsqrtedp xvsqrtdp xvrsqrtesp xvsqrtsp	VXSQRT = 0 before instruction execution *Condition for Invalid Operation Exception* ***Invalid Square Root*** does **not** occur	VXSQRT = 0
I-7.4.1.InvalidConst.15 VXCVI is sticky		
Representative of *VSX conversion to integer instructions*	VXCVI = 1 before instruction execution *Condition for Invalid Operation Exception* ***Invalid Integer Convert*** does **not** occur	VXCVI = 1
I-7.4.1.InvalidConst.16 VXCVI stays zero when exception condition **does not occur**		
Representative of *VSX conversion to integer instructions*	VXCVI = 0 before instruction execution *Condition for Invalid Operation Exception* ***Invalid Integer Convert*** does **not** occur	VXCVI = 0
I-7.4.1.InvalidConst.17 VXSOFT is sticky		
mtfsfi mtfsf mtfsb1	VXSOFT = 1 before instruction execution *Condition for Invalid Operation Exception* ***Software Defined*** does **not** occur	VXSOFT = 1
I-7.4.1.InvalidConst.18 VXSOFT stays zero when exception condition **does not occur**		
mtfsfi mtfsf mtfsb1	VXSOFT = 0 before instruction execution *Condition for Invalid Operation Exception* ***Software Defined*** does **not** occur	VXSOFT = 0

8.2.1.3. Actions taken when the exception is enabled

VSX Scalar Floating-Point Arithmetic instructions: xsadddp xsdivdp xsmuldp xssubdp xsaddsp xsdivsp xsmulsp xssubsp xsmaddadp xsmaddmdp xsmsubadp xsmsubmdp xsnmaddadp xsnmaddmdp xsnmsubadp xsnmsubmdp xsmaddasp xsmaddmsp xsmsubasp xsmsubmsp xsnmaddasp xsnmaddmsp xsnmsubasp xsnmsubmsp

VSX Scalar DP-SP Conversion instructions: xscvdpsp xscvspdp

VSX Scalar Convert Floating-Point Double-Precision to Integer instructions: xscvdpsxds xscvdpsxws xscvdpuxds xscvdpuxws

VSX Scalar Round to Floating-Point Double-Precision Integer instructions: xsrdpi xsrdpic xsrdpim xsrdpip xsrdpiz

VSX Scalar Floating-Point Compare Double-Precision instructions: xscmpodp xscmpudp

VSX Vector Floating-Point Arithmetic instructions: xvadddp xvdivdp xvmuldp xvsubdp xvaddsp xvdivsp xvmulsp xvsubsp xvmaddadp xvmaddmdp xvmsubadp xvmsubmdp xvnmaddadp xvnmaddmdp xvnmsubadp xvnmsubmdp xvmaddasp xvmaddmsp xvmsubasp xvmsubmsp xvnmaddasp xvnmaddmsp xvnmsubasp xvnmsubmsp

VSX Vector Floating-Point Compare instructions: xvcmpeqdp[.] xvcmpgedp[.] xvcmpgtdp[.] xvcmpeqsp[.] xvcmpgesp[.] xvcmpgtsp[.]

VSX Vector Maximum/Minimum instructions: xvmaxdp xvmindp xvmaxsp xvminsp

VSX Vector DP-SP Conversion instructions: xvcvdpsp xvcvspdp

VSX Vector Convert Floating-Point to Integer instructions: xvcvdpsxds xvcvdpsxws xvcvdpuxds xvcvdpuxws xvcvspsxds xvcvspsxws xvcvspuxds xvcvspuxws

VSX Vector Round to Floating-Point Integer instructions: xvrdpi xvrdpic xvrdpim xvrdpip xvrdpiz xvrspi xvrspic xvrspim xvrspip xvrspiz

VSX Scalar Quad-Precision Arithmetic instructions: xsaddqp[o] xsdivqp[o] xsmulqp[o] xssqrtqp[o] xssubqp[o] xsmaddqp[o] xsmsubqp[o] xsnmaddqp[o] xsnmsubqp[o]

VSX Scalar Quad-Precision Convert to Integer instructions: xscvqpsdz xscvqpswz xscvqpudz xscvqpuwz

VSX Scalar Round Quad-Precision to Double-Extended-Precision instruction: xsrqpxp

VSX Scalar Round to Quad-Precision Integer instruction: xsrqpi

VSX Scalar Convert with round Quad-Precision to Double-Precision format [using round to Odd] instructions: xscvqpdp[o]

Instructions tested	**Compliance conditions**	**Expected result**
I-7.4.1.VE1.1 The actions that will be taken when the Invalid Operation Exception is enabled for floating-point scalar instructions		
VSX Scalar Floating-Point Arithmetic instructions *VSX Scalar DP-SP Conversion Instructions* *VSX Scalar Convert Floating-Point Double-Precision to Integer Instructions* *VSX Scalar Round to Floating-Point Double-Precision Integer Instructions*	VE = 1 VXSNAN | VXISI | VXIDI | VXZDZ | VXIMZ | VXSQRT | VXCVI = 1 *Condition for Invalid Operation Exception occurs*	Update of VSR[XT] is suppressed FR = 0 FI = 0 FPRF is unchanged
I-7.4.1.VE1.2 The actions that will be taken when the Invalid Operation Exception is enabled for floating-point scalar compare instructions		
VSX Scalar Floating-Point Compare Double-Precision Instructions	VE = 1 VXSNAN | VXVC = 1 *Condition for Invalid Operation Exception occurs*	FR is unchanged FI is unchanged C is unchanged FPCC is set to reflect unordered
I-7.4.1.VE1.3 The actions that will be taken when the Invalid Operation Exception is enabled for floating-point vector instructions		

Instructions tested	Compliance conditions	Expected result
VSX Vector Floating-Point Arithmetic instructions *VSX Vector Floating-Point Compare Instructions* *VSX Vector Maximum/Minimum Instructions* *VSX Vector DP-SP Conversion Instructions* *VSX Vector Convert Floating-Point to Integer Instructions* *VSX Vector Round to Floating-Point Integer Instructions*	VE = 1 VXSNAN | VXISI | VXIDI | VXZDZ | VXIMZ | VXSQRT | VXCVI | VXVC = 1 *Condition for Invalid Operation Exception occurs*	Update of VSR[XT] is suppressed for all vector elements FR is unchanged FI is unchanged FPRF is unchanged
I-7.4.1.VE1.4 The actions that will be taken when the Invalid Operation Exception is enabled for VSX Scalar Quad-Precision instructions		
VSX Scalar Quad-Precision Arithmetic instructions *VSX Scalar Quad-Precision Convert to Integer Instructions* *VSX Scalar Round Quad-Precision to Double-Extended-Precision Instructions* *VSX Scalar Round to Quad-Precision Integer Instructions* *VSX Scalar Convert with round Quad-Precision to Double-Precision format Instructions*	VE = 1 VXSNAN | VXISI | VXIDI | VXZDZ | VXIMZ | VXSQRT | VXCVI = 1 *Condition for Invalid Operation Exception occurs*	VSR[VRT+32] is not modified FR = 0 FI = 0 FPRF is not modified
I-7.4.1.VE1.5 The actions that will be taken when the Invalid Operation Exception is enabled for VSX Scalar Quad-Precision Compare instructions		
VSX Scalar Compare Ordered Quad-Precision instruction (xscmpoqp) *VSX Scalar Compare Unordered Quad-Precision Instruction* (xscmpuqp)	VE = 1 VXSNAN | VXVC = 1 *Condition for Invalid Operation Exception occurs*	FR, FI, and C are not modified FPCC is set to reflect unordered
I-7.4.1.VE1.6 The actions that will be taken when the Invalid Operation Exception is enabled for VSX Scalar Half-Precision Compare instructions		
VSX Scalar Convert Half-Precision to Double-Precision format instruction (xscvhpdp) *VSX Scalar Convert with round Double-Precision to Half-Precision format Instruction* (xscvdphp)	VE = 1 VXSNAN = 1 *Condition for Invalid Operation Exception occurs*	VSR[XT] is not modified FR and FI are set to 0 FPRF is not modified
I-7.4.1.VE1.7 The actions that will be taken when the Invalid Operation Exception is enabled for VSX Scalar Convert Half-Precision instructions		
VSX Vector Convert Half-Precision to Single-Precision format instruction (xvcvhpsp) *VSX Vector Convert with round Single-Precision to Half-Precision format Instruction* (xvcvsphp)	VE = 1 VXSNAN = 1 *Condition for Invalid Operation Exception occurs*	VSR[XT] is not modified FR and FI are not modified FPRF is not modified

8.2.1.4. Action taken when the exception is disabled

VSX Vector Single-Precision Arithmetic instructions: xvaddsp xvdivsp xvmulsp xvsubsp xvmaddasp xvmaddmsp xvmsubasp xvmsubmsp xvnmaddasp xvnmaddmsp xvnmsubasp xvnmsubmsp

VSX Vector Single-Precision Maximum/Minimum instructions: xvmaxsp xvminsp

VSX Vector Round to Single-Precision Integer instructions: xvrspi xvrspic xvrspim xvrspip xvrspiz

VSX Scalar Double-Precision Arithmetic instructions: xsadddp xsdivdp xsmuldp xssubdp xsmaddadp xsmaddmdp xsmsubadp xsmsubmdp xsnmaddadp xsnmaddmdp xsnmsubadp xsnmsubmdp

VSX Scalar Double-Precision Maximum/Minimum instructions: xsmaxcdp xsmaxdp xsmaxjdp xsmincdp xsmindp xsminjdp

VSX Scalar Round to Double-Precision Integer instructions: xsrdpi xsrdpic xsrdpim xsrdpip xsrdpiz

VSX Vector Double-Precision Arithmetic instructions: xvadddp xvdivdp xvmuldp xvsubdp xvmaddadp xvmaddmdp xvmsubadp xvmsubmdp xvnmaddadp xvnmaddmdp xvnmsubadp xvnmsubmdp

VSX Vector Double-Precision Maximum/Minimum instructions: xvmaxdp xvmindp

VSX Vector Round to Double-Precision Integer instructions: xvrdpi xvrdpic xvrdpim xvrdpip xvrdpiz

Instructions tested	**Compliance conditions**	**Expected result**
I-7.4.1.VE0.1 The actions that will be taken when the Invalid Operation Exception is disabled for VSX Scalar Convert with round Double-Precision to Single-Precision format instruction		
xscvdpsp	VE = 0 VXSNAN = 1 *Condition for Invalid Operation Exception occurs*	VSR[XT].word[0] = single-precision representation of a QNaN FR = 0 FI = 0 FPRF is set to indicate the class of the result QNaN
I-7.4.1.VE0.2 The actions that will be taken when the Invalid Operation Exception is disabled for vector single-precision instructions		
VSX Vector Single-Precision Arithmetic instructions *VSX Vector Single-Precision Maximum/ Minimum Instructions* xvcvdpsp *VSX Vector Round to Single-Precision Integer Instructions*	VE = 0 VXSNAN | VXISI | VXIDI | VXZDZ | VXIMZ | VXSQRT = 1 *Condition for Invalid Operation Exception occurs*	VSR[XT].word["respective"] = single-precision QNaN FR is not modified FI is not modified FPRF is not modified
I-7.4.1.VE0.3 The actions that will be taken when the Invalid Operation Exception is disabled for scalar double-precision instructions		
VSX Scalar Double-Precision Arithmetic instructions *VSX Scalar Double-Precision Maximum/ Minimum Instructions* xscvspdp *VSX Scalar Round to Double-Precision Integer Instructions*	VE = 0 VXSNAN | VXISI | VXIDI | VXZDZ | VXIMZ | VXSQRT = 1 *Condition for Invalid Operation Exception occurs*	VSR[XT].word[0] = double-precision QNaN VSR[XT].word[1] is undefined FR = 0 FI = 0 FPRF is set to indicate the class of the result QNaN
I-7.4.1.VE0.4 The actions that will be taken when the Invalid Operation Exception is disabled for vector double-precision instructions		

Instructions tested	Compliance conditions	Expected result
VSX Vector Double-Precision Arithmetic instructions *VSX Vector Double-Precision Maximum/ Minimum Instructions* xvcvspdp *VSX Vector Round to Double-Precision Integer Instructions*	VE = 0 VXSNAN | VXISI | VXIDI | VXZDZ | VXIMZ | VXSQRT = 1 *Condition for Invalid Operation Exception occurs*	VSR[XT].word["respective"] = double-precision QNaN FR is not modified FI is not modified FPRF is not modified
I-7.4.1.VE0.5 The actions that will be taken when the Invalid Operation Exception is disabled when Double-precision operand in VSR[XB].doubleword[0] is positive or +Infinity		
xscvdpsxd	VE = 0 VXSNAN | VXCVI = 1 Double-precision operand in VSR[XB].doubleword[0] is positive or +Infinity *Condition for Invalid Operation Exception occurs*	VSR[XT].doubleword[0] = 0x7FFF_FFFF_FFFF_FFFF VSR[XT].doubleword[1] is undefined FR = 0 FI = 0 FPRF is undefined
I-7.4.1.VE0.6 The actions that will be taken when the Invalid Operation Exception is disabled when Double-precision operand in VSR[XB].doubleword[0] is negative or -Infinity or NaN		
xscvdpsxd	VE = 0 VXSNAN | VXCVI = 1 Double-precision operand in VSR[XB].doubleword[0] is negative or -Infinity or NaN *Condition for Invalid Operation Exception occurs*	VSR[XT].doubleword[0] = 0x8000_0000_0000_0000 VSR[XT].doubleword[1] is undefined FR = 0 FI = 0 FPRF is undefined
I-7.4.1.VE0.7 The actions that will be taken when the Invalid Operation Exception is disabled when Double-precision operand in VSR[XB].doubleword[0] is positive or +Infinity for xscvdpuxd instruction		
xscvdpuxd	VE = 0 VXSNAN | VXCVI = 1 Double-precision operand in VSR[XB].doubleword[0] is positive or +Infinity *Condition for Invalid Operation Exception occurs*	VSR[XT].doubleword[0] = 0xFFFF_FFFF_FFFF_FFFF VSR[XT].doubleword[1] is undefined FR = 0 FI = 0 FPRF is undefined
I-7.4.1.VE0.8 The actions that will be taken when the Invalid Operation Exception is disabled when Double-precision operand in VSR[XB].doubleword[0] is negative or -Infinity or NaN for xscvdpuxd instruction		
xscvdpuxd	VE = 0 VXSNAN | VXCVI = 1 Double-precision operand in VSR[XB].doubleword[0] is negative or -Infinity or NaN *Condition for Invalid Operation Exception occurs*	VSR[XT].doubleword[0] = 0x0000_0000_0000_0000 VSR[XT].doubleword[1] is undefined FR = 0 FI = 0 FPRF is undefined
I-7.4.1.VE0.9 The actions that will be taken when the Invalid Operation Exception is disabled when Double-precision operand in VSR[XB].doubleword[0] is positive or +Infinity for xscvdpsxw instruction		
xscvdpsxw	VE = 0 VXSNAN | VXCVI = 1	VSR[XT].word[1] = 0x7FFF_FFFF VSR[XT].word[0] is undefined

Instructions tested	Compliance conditions	Expected result
	Double-precision operand in VSR[XB].doubleword[0] is positive or +Infinity *Condition for Invalid Operation Exception occurs*	VSR[XT].word[2] is undefined VSR[XT].word[3] is undefined FR = 0 FI = 0 FPRF is undefined
I-7.4.1.VE0.10 The actions that will be taken when the Invalid Operation Exception is disabled when Double-precision operand in VSR[XB].doubleword[0] is is negative or -Infinity or NaN for xscvdpsxw instruction		
xscvdpsxw	VE = 0 VXSNAN | VXCVI = 1 Double-precision operand in VSR[XB].doubleword[0] is negative or -Infinity or NaN *Condition for Invalid Operation Exception occurs*	VSR[XT].word[1] = 0x8000_0000 VSR[XT].word[0] is undefined VSR[XT].word[2] is undefined VSR[XT].word[3] is undefined FR = 0 FI = 0 FPRF is undefined
I-7.4.1.VE0.11 The actions that will be taken when the Invalid Operation Exception is disabled when Double-precision operand in VSR[XB].doubleword[0] is positive or +Infinity for xscvdpuxw instruction		
xscvdpuxw	VE = 0 VXSNAN | VXCVI = 1 Double-precision operand in VSR[XB].doubleword[0] is positive or +Infinity *Condition for Invalid Operation Exception occurs*	VSR[XT].word[1] = 0xFFFF_FFFF VSR[XT].word[0] is undefined VSR[XT].word[2] is undefined VSR[XT].word[3] is undefined FR = 0 FI = 0 FPRF is undefined
I-7.4.1.VE0.12 The actions that will be taken when the Invalid Operation Exception is disabled when Double-precision operand in VSR[XB].doubleword[0] is negative or -Infinity or NaN for xscvdpuxw instruction		
xscvdpuxw	VE = 0 VXSNAN | VXCVI = 1 Double-precision operand in VSR[XB].doubleword[0] is negative or -Infinity or NaN *Condition for Invalid Operation Exception occurs*	VSR[XT].word[1] = 0x0000_0000 VSR[XT].word[0] is undefined VSR[XT].word[2] is undefined VSR[XT].word[3] is undefined FR = 0 FI = 0 FPRF is undefined
I-7.4.1.VE0.13 The actions that will be taken when the Invalid Operation Exception is disabled when Double-precision operand in VSR[XB].doubleword[i] is positive or +Infinity for xvcvdpsxd instruction		
xvcvdpsxd	VE = 0 VXSNAN | VXCVI = 1 Double-precision operand in VSR[XB].doubleword[i] is positive or +Infinity *Condition for Invalid Operation Exception occurs*	VSR[XT].doubleword[i] = 0x7FFF_FFFF_FFFF_FFFF FR is not modified FI is not modified FPRF is not modified

Instructions tested	Compliance conditions	Expected result
I-7.4.1.VE0.14 The actions that will be taken when the Invalid Operation Exception is disabled when Double-precision operand in VSR[XB].doubleword[i] is negative or -Infinity or NaN for xvcvdpsxd instruction		
xvcvdpsxd	VE = 0 VXSNAN | VXCVI = 1 Double-precision operand in VSR[XB].doubleword[i] is negative or -Infinity or NaN *Condition for Invalid Operation Exception occurs*	VSR[XT].doubleword[i] = 0x8000_0000_0000_0000 FR is not modified FI is not modified FPRF is not modified
I-7.4.1.VE0.15 The actions that will be taken when the Invalid Operation Exception is disabled when Double-precision operand in VSR[XB].doubleword[i] is positive or +Infinity for xvcvdpuxd instruction		
xvcvdpuxd	VE = 0 VXSNAN | VXCVI = 1 Double-precision operand in VSR[XB].doubleword[i] is positive or +Infinity *Condition for Invalid Operation Exception occurs*	VSR[XT].doubleword[i] = 0xFFFF_FFFF_FFFF_FFFF FR is not modified FI is not modified FPRF is not modified
I-7.4.1.VE0.16 The actions that will be taken when the Invalid Operation Exception is disabled when Double-precision operand in VSR[XB].doubleword[i] is negative or -Infinity or NaN for xvcvdpuxd instruction		
xvcvdpuxd	VE = 0 VXSNAN | VXCVI = 1 Double-precision operand in VSR[XB].doubleword[i] is negative or -Infinity or NaN *Condition for Invalid Operation Exception occurs*	VSR[XT].doubleword[i] = 0x0000_0000_0000_0000 FR is not modified FI is not modified FPRF is not modified
I-7.4.1.VE0.17 The actions that will be taken when the Invalid Operation Exception is disabled when Double-precision operand in VSR[XB].doubleword[i] is positive or +Infinity for xvcvdpsxw instruction		
xvcvdpsxw	VE = 0 VXSNAN | VXCVI = 1 Double-precision operand in VSR[XB].doubleword[i] is positive or +Infinity *Condition for Invalid Operation Exception occurs*	VSR[XT].word[2i] = 0x7FFF_FFFF VSR[XT].word[2i+1] is undefined FR is not modified FI is not modified FPRF is not modified
I-7.4.1.VE0.18 The actions that will be taken when the Invalid Operation Exception is disabled when Double-precision operand in VSR[XB].doubleword[i] is negative or -Infinity or NaN for xvcvdpsxw instruction		
xvcvdpsxw	VE = 0 VXSNAN | VXCVI = 1 Double-precision operand in VSR[XB].doubleword[i] is negative or -Infinity or NaN *Condition for Invalid Operation Exception occurs*	VSR[XT].word[2i] = 0x8000_0000 VSR[XT].word[2i+1] is undefined FR is not modified FI is not modified FPRF is not modified
I-7.4.1.VE0.19 The actions that will be taken when the Invalid Operation Exception is disabled when Double-precision operand in VSR[XB].doubleword[i] is positive or +Infinity for xvcvdpuxw instruction		
xvcvdpuxw	VE = 0	VSR[XT].word[2i] = 0xFFFF_FFFF

Instructions tested	Compliance conditions	Expected result
	VXSNAN | VXCVI = 1 Double-precision operand in VSR[XB].doubleword[i] is positive or +Infinity *Condition for Invalid Operation Exception occurs*	VSR[XT].word[2i+1] is undefined FR is not modified FI is not modified FPRF is not modified
I-7.4.1.VE0.20 The actions that will be taken when the Invalid Operation Exception is disabled when Double-precision operand in VSR[XB].doubleword[i] is negative or -Infinity or NaN for xvcvdpuxw instruction		
xvcvdpuxw	VE = 0 VXSNAN | VXCVI = 1 Double-precision operand in VSR[XB].doubleword[i] is negative or -Infinity or NaN *Condition for Invalid Operation Exception occurs*	VSR[XT].word[2i] = 0x0000_0000 VSR[XT].word[2i+1] is undefined FR is not modified FI is not modified FPRF is not modified
I-7.4.1.VE0.21 The actions that will be taken when the Invalid Operation Exception is disabled when single-precision operand in VSR[XB].word[2i] is positive or +Infinity for xvcvspsxd instruction		
xvcvspsxd	VE = 0 VXSNAN | VXCVI = 1 single-precision operand in VSR[XB].word[2i] is positive or +Infinity *Condition for Invalid Operation Exception occurs*	VSR[XT].doubleword[i] = 0x7FFF_FFFF_FFFF_FFFF FR is not modified FI is not modified FPRF is not modified
I-7.4.1.VE0.22 The actions that will be taken when the Invalid Operation Exception is disabled when single-precision operand in VSR[XB].word[2i] is negative or -Infinity or NaN for xvcvspsxd instruction		
xvcvspsxd	VE = 0 VXSNAN | VXCVI = 1 single-precision operand in VSR[XB].word[2i] is negative or -Infinity or NaN *Condition for Invalid Operation Exception occurs*	VSR[XT].doubleword[i] = 0x8000_0000_0000_0000 FR is not modified FI is not modified FPRF is not modified
I-7.4.1.VE0.23 The actions that will be taken when the Invalid Operation Exception is disabled when single-precision operand in VSR[XB].word[2i] is positive or +Infinity for xvcvspuxd instruction		
xvcvspuxd	VE = 0 VXSNAN | VXCVI = 1 single-precision operand in VSR[XB].word[2i] is positive or +Infinity *Condition for Invalid Operation Exception occurs*	VSR[XT].doubleword[i] = 0xFFFF_FFFF_FFFF_FFFF FR is not modified FI is not modified FPRF is not modified
I-7.4.1.VE0.24 The actions that will be taken when the Invalid Operation Exception is disabled when single-precision operand in VSR[XB].word[2i] is negative or -Infinity or NaN for xvcvspuxd instruction		
xvcvspuxd	VE = 0 VXSNAN | VXCVI = 1 single-precision operand in VSR[XB].word[2i] is negative or -Infinity or NaN	VSR[XT].doubleword[i] = 0x0000_0000_0000_0000 FR is not modified FI is not modified FPRF is not modified

Instructions tested	Compliance conditions	Expected result
	Condition for Invalid Operation Exception occurs	
I-7.4.1.VE0.25 The actions that will be taken when the Invalid Operation Exception is disabled when single-precision operand in VSR[XB].word[i] is positive or +Infinity for xvcvspsxw instruction		
xvcvspsxw	VE = 0 VXSNAN | VXCVI = 1 single-precision operand in VSR[XB].word[i] is positive or +Infinity *Condition for Invalid Operation Exception occurs*	VSR[XT].word[i] = 0x7FFF_FFFF FR is not modified FI is not modified FPRF is not modified
I-7.4.1.VE0.26 The actions that will be taken when the Invalid Operation Exception is disabled when single-precision operand in VSR[XB].word[i] is negative or -Infinity or NaN for xvcvspsxw instruction		
xvcvspsxw	VE = 0 VXSNAN | VXCVI = 1 single-precision operand in VSR[XB].word[i] is negative or -Infinity or NaN *Condition for Invalid Operation Exception occurs*	VSR[XT].word[i] = 0x8000_0000 FR is not modified FI is not modified FPRF is not modified
I-7.4.1.VE0.27 The actions that will be taken when the Invalid Operation Exception is disabled when single-precision operand in VSR[XB].word[i] is positive or +Infinity for xvcvspuxw instruction		
xvcvspuxw	VE = 0 VXSNAN | VXCVI = 1 single-precision operand in VSR[XB].word[i] is positive or +Infinity *Condition for Invalid Operation Exception occurs*	VSR[XT].word[i] = 0xFFFF_FFFF FR is not modified FI is not modified FPRF is not modified
I-7.4.1.VE0.28 The actions that will be taken when the Invalid Operation Exception is disabled when single-precision operand in VSR[XB].word[i] is negative or -Infinity or NaN for xvcvspuxw instruction		
xvcvspuxw	VE = 0 VXSNAN | VXCVI = 1 single-precision operand in VSR[XB].word[i] is negative or -Infinity or NaN *Condition for Invalid Operation Exception occurs*	VSR[XT].word[i] = 0x0000_0000 FR is not modified FI is not modified FPRF is not modified
I-7.4.1.VE0.29 The actions that will be taken when the Invalid Operation Exception is disabled for scalar compare double-precision instructions		
VSX Scalar Compare Double-Precision Instructions *VSX Scalar Double-Precision Maximum/ Minimum Instructions*	VE = 0 VXSNAN | VXCVI = 1 *Condition for Invalid Operation Exception occurs*	FR is unchanged FI is unchanged C is unchanged FPCC is set to reflect unordered
I-7.4.1.VE0.30 The actions that will be taken when the Invalid Operation Exception is disabled for *Vector Compare Single-Precision Instructions*		
VSX Vector Compare Single-Precision Instructions	VE = 0 VXSNAN | VXCVI = 1	VSR[XT].word[respective] = 0x0000_0000 FR is not modified

Instructions tested	Compliance conditions	Expected result
	Condition for Invalid Operation Exception occurs	FI is not modified FPRF is not modified
I-7.4.1.VE0.31 The actions that will be taken when the Invalid Operation Exception is disabled for Vector Compare double-Precision Instructions		
VSX Vector Compare Double-Precision Instructions *VSX Vector Double-Precision Maximum/ Minimum Instructions*	VE = 0 VXSNAN | VXCVI = 1 *Condition for Invalid Operation Exception occurs*	VSR[XT].doubleword[respective] = 0x0000_0000_0000_0000 FR is not modified FI is not modified FPRF is not modified
I-7.4.1.VE0.32 The actions that will be taken when the Invalid Operation Exception is disabled for *VSX Scalar Quad-Precision Instructions* and *VSX Scalar Round Quad-Precision to Integer Instructions*		
xsaddqp[o] xsdivqp[o] xsmulqp[o] xssqrtqp[o] xssubqp[o] xsmaddqp[o] xsmsubqp[o] xsnmaddqp[o] xsnmsubqp[o] xsrqpi	VE = 0 VXSNAN | VXISI | VXIDI | VXZDZ | VXIMZ | VXSQRT = 1 *Condition for Invalid Operation Exception occurs*	VSR[VRT+32] = Quiet NaN in quad-precision representation FR and FI are set to 0 FPRF is set to indicate the class of the result (Quiet NaN)
I-7.4.1.VE0.33 The actions that will be taken when the Invalid Operation Exception is disabled for *VSX Scalar Round Quad-Precision to Double-Extended_Precision Instruction*		
xsrqpxp	VE = 0 VXSNAN = 1 *Condition for Invalid Operation Exception occurs*	VSR[VRT+32] = Quiet NaN in quad-precision representation FR and FI are set to 0 FPRF is set to indicate the class of the result (Quiet NaN)
I-7.4.1.VE0.34 The actions that will be taken when the Invalid Operation Exception is disabled for *VSX Scalar Compare (Ordered, Unordered) Quad-Precision Instructions*		
xscmpoqp xscmpuqp	VE = 0 VXSNAN = 1 (if SNaN) VXVC = 1 (if Invalid Compare) *Condition for Invalid Operation Exception occurs*	FR, FI and C are unchanged FPCC is set to reflect unordered
I-7.4.1.VE0.35 The actions that will be taken when the Invalid Operation Exception is disabled for *VSX Scalar Convert with Round Quad-Precision to Double-Precision format [using round to Odd] Instructions*		
xscvqpdp[o]	VE = 0 VXSNAN = 1 *Condition for Invalid Operation Exception occurs*	VSR[VRT+32] doubleword[0] = Quiet NaN in double-precision representation VSR[VRT+32] doubleword [1] = 0x0000_0000_0000_0000 FR and FI are set to 0 FPRF is set to indicate the class of the result (Quiet NaN)
I-7.4.1.VE0.36 The actions that will be taken when the Invalid Operation Exception is disabled for *VSX Scalar Convert with Round to zero Quad-Precision to Signed Doubleword Instruction*		
xscvqpsdz	VE = 0 VXSNAN = 1 (if SNaN) VXCVI = 1 (if Invalid Integer Convert) *Condition for Invalid Operation Exception occurs*	VSR[VRT+32] doubleword[0] = 0x7FFF_FFFF_FFFF_FFFF if the quad-precision operand in VSR[VRB+32] is a positive number or +Infinity VSR[VRT+32] doubleword[0] = 0x8000_0000_0000_0000 if the quad-

Instructions tested	Compliance conditions	Expected result
		precision operand in VSR[VRB+32] is a negative number, -Infinity, or NaN VSR[VRT+32] doubleword [1] = 0x0000_0000_0000_0000 FR and FI are set to 0 FPRF is undefined
I-7.4.1.VE0.37 The actions that will be taken when the Invalid Operation Exception is disabled for *VSX Scalar Convert with Round to zero Quad-Precision to Signed Word Instruction*		
xscvqpswz	VE = 0 VXSNAN = 1 (if SNaN) VXCVI = 1 (if Invalid Integer Convert) *Condition for Invalid Operation Exception occurs*	VSR[VRT+32] word[1] = 0x7FFF_FFFF if the quad-precision operand in VSR[VRB +32] is a positive number or +Infinity VSR[VRT+32] word[1] = 0x8000_0000 if the quad-precision operand in VSR[VRB +32] is a negative number, -Infinity, or NaN VSR[VRT+32] word [0] = 0x0000_0000 VSR[VRT+32] word [2] = 0x0000_0000 VSR[VRT+32] word [3] = 0x0000_0000 FR and FI are set to 0 FPRF is undefined
I-7.4.1.VE0.38 The actions that will be taken when the Invalid Operation Exception is disabled for *VSX Scalar Convert with Round to zero Quad-Precision to Unsigned Doubleword Instruction*		
xscvqpudz	VE = 0 VXSNAN = 1 (if SNaN) VXCVI = 1 (if Invalid Integer Convert) *Condition for Invalid Operation Exception occurs*	VSR[VRT+32] doubleword[0] = 0xFFFF_FFFF_FFFF_FFFF if the quad-precision operand in VSR[VRB+32] is a positive number or +Infinity VSR[VRT+32] doubleword[0] = 0x0000_0000_0000_0000 if the quad-precision operand in VSR[VRB+32] is a negative number, -Infinity, or NaN VSR[VRT+32] doubleword [1] = 0x0000_0000_0000_0000 FR and FI are set to 0 FPRF is undefined
I-7.4.1.VE0.39 The actions that will be taken when the Invalid Operation Exception is disabled for *VSX Scalar Convert with Round to zero Quad-Precision to Unsigned Word Instruction*		
xscvqpuwz	VE = 0 VXSNAN = 1 (if SNaN) VXCVI = 1 (if Invalid Integer Convert) *Condition for Invalid Operation Exception occurs*	VSR[VRT+32] word[1] = 0xFFFF_FFFF if the quad-precision operand in VSR[VRB +32] is a positive number or +Infinity VSR[VRT+32] word[1] = 0x0000_0000 if the quad-precision operand in VSR[VRB +32] is a negative number, -Infinity, or NaN VSR[VRT+32] word [0] = 0x0000_0000 VSR[VRT+32] word [2] = 0x0000_0000 VSR[VRT+32] word [3] = 0x0000_0000 FR and FI are set to 0 FPRF is undefined

Instructions tested	Compliance conditions	Expected result
I-7.4.1.VE0.40 The actions that will be taken when the Invalid Operation Exception is disabled for *VSX Scalar Convert with Round Double-Precision to Half-Precision format Instruction*		
xscvdphp	VE = 0 VXSNAN = 1 *Condition for Invalid Operation Exception occurs*	VSR[XT] rightmost halfword of double-word[0] = Quiet NaN in half-precision representation VSR[XT] leftmost 3 halfwords of double-word[0] are set to 0 VSR[XT] doubleword[1] is undefined FR and FI are set to 0 FPRF is set to indicate the class of the result (Quiet NaN)
I-7.4.1.VE0.41 The actions that will be taken when the Invalid Operation Exception is disabled for *VSX Scalar Convert with Half-Precision to Double-Precision format Instruction*		
xscvhpdp	VE = 0 VXSNAN = 1 *Condition for Invalid Operation Exception occurs*	VSR[XT] doubleword[0] = Quiet NaN in double-precision representation VSR[XT] doubleword[1] is undefined FR and FI are set to 0 FPRF is set to indicate the class of the result (Quiet NaN)
I-7.4.1.VE0.42 The actions that will be taken when the Invalid Operation Exception is disabled for *VSX Vector Convert with Round Single-Precision to Half-Precision format Instruction*		
xvcvsphp	VE = 0 VXSNAN = 1 *Condition for Invalid Operation Exception occurs*	VSR[XT] rightmost halfword of respective word element = Quiet NaN in half-precision representation VSR[XT] leftmost halfword of respective word element is set to 0 FR and FI are not modified FPRF is not modified
I-7.4.1.VE0.43 The actions that will be taken when the Invalid Operation Exception is disabled for *VSX Vector Convert with Half-Precision to Single-Precision format Instruction*		
xvcvhpsp	VE = 0 VXSNAN = 1 *Condition for Invalid Operation Exception occurs*	VSR[XT] respective word element = Quiet NaN in single-precision representation FR and FI are not modified FPRF is not modified

8.2.2. Zero Divide Exception

Scenario groups:

- Setting exception bit ZX
- Actions taken when the exception is enabled
- Action taken when the exception is disabled

Condition for Zero Divide Exception for divide instructions: zero divisor value and a finite nonzero dividend value.

Condition for Zero Divide Exception for reciprocal estimate instructions: operand value of zero

8.2.2.1. Setting exception bit ZX

Instructions tested	Compliance conditions	Expected result
I-7.4.2.ZeroDiv.1 VSX Floating-Point Divide instructions set ZX to 1 when division by zero occurs		
xsdivqp[o] xsdivdp xsdivsp xvdivdp xvdivsp	*Condition for Zero Divide Exception for divide instructions* occurs	ZX = 1
I-7.4.2.ZeroDiv.2 ZX is sticky divide instructions		
Representative of: xsdivqp[o] xsdivdp xsdivsp xvdivdp xvdivsp	ZX=1 before instruction execution *Condition for Zero Divide Exception for divide instructions* does **not** occur	ZX = 1
I-7.4.2.ZeroDiv.3 ZX stays zero when exception condition **does not occur** divide instructions		
Representative of: xsdivqp[o] xsdivdp xsdivsp xvdivdp xvdivsp	ZX=0 before instruction execution *Condition for Zero Divide Exception for divide instructions* does **not** occur	ZX = 0
I-7.4.2.ZeroDiv.4 VSX Floating-Point Reciprocal Estimate instruction or a VSX Floating-Point Reciprocal Square Root Estimate instruction set ZX to 1 when operand is zero		
xsredp xsrsqrtedp xsresp xsrsqrtesp xvredp xvrsqrtedp xvresp xvrsqrtesp	*Condition for Zero Divide Exception for reciprocal estimate instructions* occurs	ZX = 1
I-7.4.2.ZeroDiv.5 ZX is sticky reciprocal estimate instructions		
Representative of: xsredp xsrsqrtedp xsresp xsrsqrtesp xvredp xvrsqrtedp xvresp xvrsqrtesp	ZX=1 before instruction execution *Condition for Zero Divide Exception for reciprocal estimate instructions* does **not** occur	ZX = 1
I-7.4.2.ZeroDiv.6 ZX stays zero when exception condition **does not occur** reciprocal estimate instructions		
Representative of: xsredp xsrsqrtedp xsresp xsrsqrtesp xvredp xvrsqrtedp xvresp xvrsqrtesp	ZX=0 before instruction execution *Condition for Zero Divide Exception for reciprocal estimate instructions* does **not** occur	ZX = 0

8.2.2.2. Actions taken when the exception is enabled

Instructions tested	Compliance conditions	Expected result
I-7.4.2.ZE1.1 The actions that will be taken when the Zero Divide Exception is enabled and operation scalar floating-point causing Zero Divide Exception instructions		
xsdivdp xsdivsp	ZE = 1 ZX = 1 *Condition for Zero Divide Exception occurs*	Update of VSR[XT] is suppressed FR = 0 FI = 0 FPRF is unchanged
I-7.4.2.ZE1.1b The actions that will be taken when the Zero Divide Exception is enabled and operation scalar floating-point reciprocal or square root causing Zero Divide Exception instructions		
xsredp xsresp xsrsqrtedp xsrsqrtesp	ZE = 1 ZX = 1 *Condition for Zero Divide Exception occurs*	Update of VSR[XT] is suppressed FR is undefined FI is undefined FPRF is unchanged
I-7.4.2.ZE1.2 The actions that will be taken when the Zero Divide Exception is enabled and operation vector floating-point causing Zero Divide Exception instructions		
xvdivdp xvdivsp xvredp xvresp xvrsqrtedp xvrsqrtesp	ZE = 1	Update of VSR[XT] is suppressed for all vector elements

Instructions tested	Compliance conditions	Expected result
	ZX = 1 *Condition for Zero Divide Exception occurs*	FI is unchanged FR is unchanged FPRF is unchanged
I-7.4.2.ZE1.3 The actions that will be taken when the Zero Divide Exception is enabled and operation vector floating-point causing Zero Divide Exception instructions		
xsdivqp[o]	ZE = 1 ZX = 1 *Condition for Zero Divide Exception occurs*	Update of VSR[VRT+32] is suppressed FI = 0 FR = 0 FPRF is unchanged

8.2.2.3. Action taken when the exception is disabled

Instructions tested	Compliance conditions	Expected result
I-7.4.2.ZE0.1 The actions that will be taken when the Zero Divide Exception is disabled for scalar floating-point divide instructions if the XOR of the signs of source operands = 0		
xsdivdp xsdivsp	ZE = 0 ZX = 1 XOR of the signs of source operands = 0 *Condition for Zero Divide Exception occurs*	VSR[XT].doubleword[0] = +Infinity in double-precision format VSR[XT].doubleword[1] undefined FR = 0 FI = 0 FPRF is set to indicate class of +Infinity
I-7.4.2.ZE0.2 The actions that will be taken when the Zero Divide Exception is disabled for scalar floating-point divide instructions if the XOR of the signs of source operands = 1		
xsdivdp xsdivsp	ZE = 0 ZX = 1 XOR of the signs of source operands = 1 *Condition for Zero Divide Exception occurs*	VSR[XT].doubleword[0]= -Infinity in double-precision format VSR[XT].doubleword[1] undefined FR = 0 FI = 0 FPRF is set to indicate class of -Infinity
I-7.4.2.ZE0.3 The actions that will be taken when the Zero Divide Exception is disabled for vector double-precision divide instruction if the XOR of the signs of source operands = 0		
xvdivdp	ZE = 0 ZX = 1 XOR of the signs of source operands = 0 Vector element i causing a zero divide exception	VSR[XT].doubleword[i] = +Infinity in double-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.2.ZE0.4 The actions that will be taken when the Zero Divide Exception is disabled for vector double-precision divide instruction if the XOR of the signs of source operands = 1		
xvdivdp	ZE = 0 ZX = 1 XOR of the signs of source operands = 1 Vector element i causing a zero divide exception	VSR[XT].doubleword[i] = -Infinity in double-precision format FR is not modified FI is not modified FPRF is not modified

Instructions tested	Compliance conditions	Expected result
I-7.4.2.ZE0.5 The actions that will be taken when the Zero Divide Exception is disabled for vector single-precision divide instruction if the XOR of the signs of source operands = 0		
xvdivsp	ZE = 0 ZX = 1 XOR of the signs of source operands = 0 Vector element i causing a zero divide exception	VSR[XT].word[i] = +Infinity in single-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.2.ZE0.6 The actions that will be taken when the Zero Divide Exception is disabled for vector single-precision divide instruction if the XOR of the signs of source operands = 1		
xvdivsp	ZE = 0 ZX = 1 XOR of the signs of source operands = 1 Vector element i causing a zero divide exception	VSR[XT].word[i] = -Infinity in single-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.2.ZE0.7 The actions that will be taken when the Zero Divide Exception is disabled for scalar floating-point reciprocal instructions if the sign of the source operand is 0		
xsredp xsresp xsrsqrtedp xsrsqrtesp	ZE = 0 ZX = 1 The sign of the source operand is 0 *Condition for Zero Divide Exception occurs*	VSR[XT].doubleword[0] = +Infinity in double-precision format VSR[XT].doubleword[1] undefined FR = 0 FI = 0 FPRF is set to indicate class of +Infinity
I-7.4.2.ZE0.8 The actions that will be taken when the Zero Divide Exception is disabled for scalar floating-point reciprocal instructions if the sign of the source operand is 1		
xsredp xsresp xsrsqrtedp xsrsqrtesp	ZE = 0 ZX = 1 The sign of the source operand is 1 *Condition for Zero Divide Exception occurs*	VSR[XT].doubleword[0]= -Infinity in double-precision format VSR[XT].doubleword[1] undefined FR = 0 FI = 0 FPRF is set to indicate class of -Infinity
I-7.4.2.ZE0.9 The actions that will be taken when the Zero Divide Exception is disabled for vector double-precision reciprocal instructions if the sign of the source operand is 0		
xvredp xvrsqrtedp	ZE = 0 ZX = 1 The sign of the source operand is 0 Vector element i causing a zero divide exception	VSR[XT].doubleword[i] = +Infinity in double-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.2.ZE0.10 The actions that will be taken when the Zero Divide Exception is disabled for vector double-precision reciprocal instructions if the sign of the source operand is 1		
xvredp xvrsqrtedp	ZE = 0 ZX = 1 The sign of the source operand is 1	VSR[XT].doubleword[i] = -Infinity in double-precision format FR is not modified FI is not modified

Instructions tested	Compliance conditions	Expected result
	Vector element i causing a zero divide exception	FPRF is not modified
I-7.4.2.ZE0.11 The actions that will be taken when the Zero Divide Exception is disabled for vector single-precision reciprocal instructions if the sign of the source operand is 0		
xvresp xvrsqrtesp	ZE = 0 ZX = 1 The sign of the source operand is 0 Vector element i causing a zero divide exception	VSR[XT].word[i] = +Infinity in single-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.2.ZE0.12 The actions that will be taken when the Zero Divide Exception is disabled for vector single-precision reciprocal instructions if the sign of the source operand is 1		
xvresp xvrsqrtesp	ZE = 0 ZX = 1 The sign of the source operand is 1 Vector element i causing a zero divide exception	VSR[XT].word[i] = -Infinity in single-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.2.ZE0.13 The actions that will be taken when the Zero Divide Exception is disabled for *VSX Scalar Divide Quad-Precision instructions* if the XOR of the signs of source operands = 0		
xsdivqp[o]	ZE = 0 ZX = 1 XOR of the signs of source operands = 0 *Condition for Zero Divide Exception occurs*	VSR[VRT+32] = +Infinity in quad-precision format FR = 0 FI = 0 FPRF is set to indicate the class of +Infinity
I-7.4.2.ZE0.14 The actions that will be taken when the Zero Divide Exception is disabled for *VSX Scalar Divide Quad-Precision instructions* if the XOR of the signs of source operands = 1		
xsdivqp[o]	ZE = 0 ZX = 1 XOR of the signs of source operands = 1 *Condition for Zero Divide Exception occurs*	VSR[VRT+32] = -Infinity in quad-precision format FR = 0 FI = 0 FPRF is set to indicate the class of -Infinity

8.2.3. Overflow Exception

Architecture sections:

- I-7.4.3 Floating-Point Overflow Exception

Scenario groups:

- Setting exception bit OX
- Actions taken when the exception is enabled
- Action taken when the exception is disabled

Condition for Overflow Exception: the magnitude of what would have been the rounded result if the exponent range were unbounded exceeds that of the largest finite number of the specified result precision

VSX Scalar Double-Precision Arithmetic instructions: xsadddp xsdivdp xsmuldp xssubdp xsmaddadp xsmaddmdp xsmsubadp xsmsubmdp xsnmaddadp xsnmaddmdp xsnmsubadp xsnmsubmdp

VSX Scalar Reciprocal Estimate Double-Precision instruction: xsredp

VSX Scalar Single-Precision Arithmetic instructions: xsaddsp xsdivsp xsmulsp xssubsp xsmaddasp xsmaddmsp xsmsubasp xsmsubmsp xsnmaddasp xsnmaddmsp xsnmsubasp xsnmsubmsp

VSX Scalar Reciprocal Estimate Single-Precision instruction: xsresp

VSX Vector Double-Precision Arithmetic instructions: xvadddp xvdivdp xvmuldp xvsubdp xvmaddadp xvmaddmdp xvmsubadp xvmsubmdp xvnmaddadp xvnmaddmdp xvnmsubadp xvnmsubmdp

VSX Vector Reciprocal Estimate Double-Precision instruction: xvredp

VSX Vector Single-Precision Arithmetic instructions: xvaddsp xvdivsp xvmulsp xvsubsp xvmaddasp xvmaddmsp xvmsubasp xvmsubmsp xvnmaddasp xvnmaddmsp xvnmsubasp xvnmsubmsp

VSX Vector Reciprocal Estimate Single-Precision instruction: xvresp

VSX Scalar Quad-Precision Arithmetic instructions: xsaddqp[o] xsdivqp[o] xsmulqp[o] xssqrtqp[o] xssubqp[o] xsmaddqp[o] xsmsubqp[o] xsnmaddqp[o] xsnmsubqp[o]

VSX Scalar Round Quad-Precision to Double-Extended-Precision instruction: xsrqpxp

VSX Scalar Convert with round Quad-Precision to Double-Precision format instructions: xscvqpdp[o]

VSX Scalar Convert with round Double-Precision to Half-Precision format instruction: xscvdphp

VSX Vector Convert with round Single-Precision to Half-Precision format instruction: xvcvsphp

8.2.3.1. Setting exception bit OX

VSX potentially overflowing/ underflowing instructions: *VSX Scalar Double-Precision Arithmetic Instructions, VSX Scalar Reciprocal Estimate Double-Precision instruction, VSX Scalar Single-Precision Arithmetic instructions, VSX Scalar Reciprocal Estimate Single-Precision instruction, VSX Vector Double-Precision Arithmetic instructions, VSX Vector Reciprocal Estimate Double-Precision instruction, VSX Vector Single-Precision Arithmetic instructions, VSX Vector Reciprocal Estimate Single-Precision instruction, VSX Scalar Quad-Precision Arithmetic instructions, VSX Scalar Round Quad-Precision to Double-Extended-Precision instruction, VSX Scalar Convert with round Quad-Precision to Double-Precision format instruction, VSX Scalar Convert with round Double-Precision to Half-Precision format instruction, VSX Vector Convert with round Single-Precision to Half-Precision format instruction,* xscvdpsp xvcvdpsp xssqrtsp xsrsp xsrsqrtesp

Instructions tested	**Compliance conditions**	**Expected result**
I-7.4.3.Over.1 OX is set to 1 when overflow occurs		
VSX potentially overflowing/ underflowing instructions	*Condition for Overflow Exception* occurs	OX = 1
I-7.4.3.Over.2 OX is sticky		
Representative of *VSX potentially overflowing/ underflowing instructions*	OX=1 before instruction execution *Condition for Overflow Exception* does **not** occur	OX = 1
I-7.4.3.Over.3 OX stays zero when exception condition **does not occur**		
Representative of *VSX potentially overflowing/ underflowing instructions*	OX=0 before instruction execution	OX = 0

Instructions tested	Compliance conditions	Expected result
	Condition for Overflow Exception does **not** occur	

8.2.3.2. Actions taken when the exception is enabled

Instructions tested	Compliance conditions	Expected result
I-7.4.3.OE1.1 The actions that will be taken when the Overflow Exception is enabled for the xscvdpsp instruction		
xscvdpsp	OE = 1 OX = 1 The unbiased exponent of the normalized intermediate result is less than or equal to Emax+192 *Condition for Overflow Exception occurs*	The exponent is adjusted by subtracting 192 VSR[XT].word[0] = the adjusted rounded result in single-precision format VSR[XT].word[1] is undefined VSR[XT].word[2] is undefined VSR[XT].word[3] is undefined FPRF is set to indicate ±Normal Number
I-7.4.3.OE1.2 The actions that will be taken when the Overflow Exception is enabled for the *VSX Scalar Double-Precision Arithmetic instructions* and *VSX Scalar Reciprocal Estimate Double-Precision instruction*		
VSX Scalar Double-Precision Arithmetic instructions *VSX Scalar Reciprocal Estimate Double-Precision instruction*	OE = 1 OX = 1 *Condition for Overflow Exception occurs*	The result of the normalized intermediate result adjusted by subtracting 1536. VSR[XT].doubleword[0] = the rounded result in double-precision format VSR[XT].doubleword[1] is undefined FPRF is set to indicate ±Normal Number
I-7.4.3.OE1.3 The actions that will be taken when the Overflow Exception is enabled for the *VSX Scalar Single-Precision Arithmetic instructions* and *VSX Scalar Reciprocal Estimate Single-Precision instruction*		
VSX Scalar Single-Precision Arithmetic instructions *VSX Scalar Reciprocal Estimate Single-Precision instruction*	OE = 1 OX = 1 *Condition for Overflow Exception occurs*	The exponent is adjusted by subtracting 192 VSR[XT].doubleword[0] = the rounded result in double-precision format VSR[XT].doubleword[1] is undefined FPRF is set to indicate ±Normal Number
I-7.4.3.OE1.4 The actions that will be taken when the Overflow Exception is enabled for the VSX Vector Double-Precision and VSX Vector Single-Precision relevant instructions		
VSX Vector Double-Precision Arithmetic instructions *VSX Vector Reciprocal Estimate Double-Precision instruction* *VSX Vector Single-Precision Arithmetic instructions* *VSX Vector Reciprocal Estimate Single-Precision instruction* xvcvdpsp	OE = 1 OX = 1 *Condition for Overflow Exception occurs*	Update of VSR[XT] is suppressed for all vector elements FR is not modified FI is not modified FPRF is not modified
I-7.4.3.OE1.5 The actions that will be taken when the Overflow Exception is enabled for *VSX Scalar Quad-Precision Arithmetic instructions* and *VSX Scalar Round Quad-Precision to Double-Extended-Precision instruction*		
xsaddqp[o], xsdivqp[o], xsmulqp[o], xssqrtqp[o], xssubqp[o], xsmaddqp[o],	OE = 1 OX = 1	The exponent is adjusted by subtracting 24576

Instructions tested	Compliance conditions	Expected result
xsmsubqp[o], xsnmaddqp[o], xsnmsubqp[o], xsrqpxp	*Condition for Overflow Exception occurs*	VSR[VRT+32] = the adjusted, rounded result in quad-precision format FPRF is set to indicate ±Normal Number (unless the result is undefined)
I-7.4.3.OE1.6 The actions that will be taken when the Overflow Exception is enabled for *VSX Scalar Convert with round Quad-Precision to Double-Precision format [using round to Odd] instructions*		
xscvqpdp[o]	OE = 1 OX = 1 *Condition for Overflow Exception occurs*	The exponent is adjusted by subtracting 1536 If the adjusted exponent is greater than +1023 (Emax), the result is undefined VSR[VRT+32].doubleword[0] = the adjusted, rounded result in double-precision format VSR[VRT+32].doubleword[1] = 0x0000_0000_0000_0000 FPRF is set to indicate ±Normal Number (unless the result is undefined)
I-7.4.3.OE1.7 The actions that will be taken when the Overflow Exception is enabled for *VSX Scalar Convert with round Double-Precision to Half-Precision format instruction*		
xscvdphp	OE = 1 OX = 1 *Condition for Overflow Exception occurs*	The exponent is adjusted by subtracting 24 If the adjusted exponent is greater than +15 (Emax), the result is undefined VSR[XT] rightmost halfword of doubleword[0] = the adjusted, rounded result in half-precision format VSR[XT] leftmost 3 halfwords of doubleword[0] are set to 0 VSR[XT] doubleword[1] is undefined FPRF is set to indicate ±Normal Number (unless the result is undefined)
I-7.4.3.OE1.8 The actions that will be taken when the Overflow Exception is enabled for *VSX Vector Convert with round Single-Precision to Half-Precision format instruction*		
xvcvsphp	OE = 1 OX = 1 *Condition for Overflow Exception occurs*	VSR[XT] is not modified FR and FI are not modified FPRF is not modified

8.2.3.3. Action taken when the exception is disabled

Instructions tested	Compliance conditions	Expected result
I-7.4.3.OE0.1 The actions that will be taken when the Overflow Exception is disabled for the xscvdpsp instruction when it is Negative overflow with Round to Nearest Even mode		
xscvdpsp	OE = 0 OX = 1 Round to Nearest Even Negative overflow	XX = 1 VSR[XT].word[0] = -Infinity in single-precision format VSR[XT].word[1] is undefined VSR[XT].word[2] is undefined VSR[XT].word[3] is undefined

Instructions tested	Compliance conditions	Expected result
		FR is undefined FI = 1 FPRF is set to indicate -Infinity
I-7.4.3.OE0.2 The actions that will be taken when the Overflow Exception is disabled for the xscvdpsp instruction when it is Positive overflow with Round to Nearest Even mode		
xscvdpsp	OE = 0 OX = 1 Round to Nearest Even Positive overflow	XX = 1 VSR[XT].word[0] = +Infinity in single-precision format VSR[XT].word[1] is undefined VSR[XT].word[2] is undefined VSR[XT].word[3] is undefined FR is undefined FI = 1 FPRF is set to indicate +Infinity
I-7.4.3.OE0.3 The actions that will be taken when the Overflow Exception is disabled for the xscvdpsp instruction when it is Negative overflow with Round toward Zero mode		
xscvdpsp	OE = 0 OX = 1 Round toward Zero Negative overflow	XX = 1 VSR[XT].word[0] = single-precision formats most negative finite number VSR[XT].word[1] is undefined VSR[XT].word[2] is undefined VSR[XT].word[3] is undefined FR is undefined FI = 1 FPRF is set to indicate -Normal Number
I-7.4.3.OE0.4 The actions that will be taken when the Overflow Exception is disabled for the xscvdpsp instruction when it is Positive overflow with Round toward Zero mode		
xscvdpsp	OE = 0 OX = 1 Round toward Zero Positive overflow	XX = 1 VSR[XT].word[0] = single-precision formats most positive finite number VSR[XT].word[1] is undefined VSR[XT].word[2] is undefined VSR[XT].word[3] is undefined FR is undefined FI = 1 FPRF is set to indicate +Normal Number
I-7.4.3.OE0.5 The actions that will be taken when the Overflow Exception is disabled for the xscvdpsp instruction when it is Negative overflow with Round toward +Infinity mode		
xscvdpsp	OE = 0 OX = 1 Round toward +Infinity	XX = 1 VSR[XT].word[0] = single-precision formats most negative finite number

Instructions tested	Compliance conditions	Expected result
	Negative overflow	VSR[XT].word[1] is undefined VSR[XT].word[2] is undefined VSR[XT].word[3] is undefined FR is undefined FI = 1 FPRF is set to indicate -Normal Number
I-7.4.3.OE0.6 The actions that will be taken when the Overflow Exception is disabled for the xscvdpsp instruction when it is Positive overflow with Round toward +Infinity mode		
xscvdpsp	OE = 0 OX = 1 Round toward +Infinity Positive overflow	XX = 1 VSR[XT].word[0] = +Infinity in single-precision format VSR[XT].word[1] is undefined VSR[XT].word[2] is undefined VSR[XT].word[3] is undefined FR is undefined FI = 1 FPRF is set to indicate +Infinity
I-7.4.3.OE0.7 The actions that will be taken when the Overflow Exception is disabled for the xscvdpsp instruction when it is Negative overflow with Round toward -Infinity mode		
xscvdpsp	OE = 0 OX = 1 Round toward -Infinity Negative overflow	XX = 1 VSR[XT].word[0] = -Infinity in single-precision format VSR[XT].word[1] is undefined VSR[XT].word[2] is undefined VSR[XT].word[3] is undefined FR is undefined FI = 1 FPRF is set to indicate -Infinity
I-7.4.3.OE0.8 The actions that will be taken when the Overflow Exception is disabled for the xscvdpsp instruction when it is Positive overflow with Round toward -Infinity mode		
xscvdpsp	OE = 0 OX = 1 Round toward -Infinity Positive overflow	XX = 1 VSR[XT].word[0] = single-precision formats most positive finite number VSR[XT].word[1] is undefined VSR[XT].word[2] is undefined VSR[XT].word[3] is undefined FR is undefined FI = 1 FPRF is set to indicate +Normal Number

Instructions tested	Compliance conditions	Expected result
I-7.4.3.OE0.9 The actions that will be taken when the Overflow Exception is disabled for the scalar arithmetic instructions and scalar reciprocal estimate instructions when it is Negative overflow with Round to Nearest Even mode		
VSX Scalar Single-Precision Arithmetic instructions *VSX Scalar Reciprocal Estimate Single-Precision instruction* *VSX Scalar Double-Precision Arithmetic instructions* *VSX Scalar Reciprocal Estimate Double-Precision instruction*	OE = 0 OX = 1 Round to Nearest Even Negative overflow	XX = 1 VSR[XT].doubleword[0]= -Infinity in double-precision format VSR[XT].doubleword[1] is undefined FR is undefined FI = 1 FPRF is set to indicate -Infinity
I-7.4.3.OE0.10 The actions that will be taken when the Overflow Exception is disabled for the scalar arithmetic instructions and scalar reciprocal estimate instructions when it is Positive overflow with Round to Nearest Even mode		
VSX Scalar Single-Precision Arithmetic instructions *VSX Scalar Reciprocal Estimate Single-Precision instruction* *VSX Scalar Double-Precision Arithmetic instructions* *VSX Scalar Reciprocal Estimate Double-Precision instruction*	OE = 0 OX = 1 Round to Nearest Even Positive overflow	XX = 1 VSR[XT].doubleword[0]= +Infinity in double-precision format VSR[XT].doubleword[1] is undefined FR is undefined FI = 1 FPRF is set to indicate +Infinity
I-7.4.3.OE0.11 The actions that will be taken when the Overflow Exception is disabled for the scalar arithmetic instructions and scalar reciprocal estimate instructions when it is Negative overflow with Round toward Zero mode		
VSX Scalar Single-Precision Arithmetic instructions *VSX Scalar Reciprocal Estimate Single-Precision instruction* *VSX Scalar Double-Precision Arithmetic instructions* *VSX Scalar Reciprocal Estimate Double-Precision instruction*	OE = 0 OX = 1 Round toward Zero Negative overflow	XX = 1 VSR[XT].doubleword[0]= most negative finite number in double-precision format VSR[XT].doubleword[1] is undefined FR is undefined FI = 1 FPRF is set to indicate -Normal Number
I-7.4.3.OE0.12 The actions that will be taken when the Overflow Exception is disabled for the scalar arithmetic instructions and scalar reciprocal estimate instructions when it is Positive overflow with Round toward Zero mode		
VSX Scalar Single-Precision Arithmetic instructions *VSX Scalar Reciprocal Estimate Single-Precision instruction* *VSX Scalar Double-Precision Arithmetic instructions* *VSX Scalar Reciprocal Estimate Double-Precision instruction*	OE = 0 OX = 1 Round toward Zero Positive overflow	XX = 1 VSR[XT].doubleword[0]= most positive finite number in double-precision format VSR[XT].doubleword[1] is undefined FR is undefined FI = 1 FPRF is set to indicate +Normal Number
I-7.4.3.OE0.13 The actions that will be taken when the Overflow Exception is disabled for the scalar arithmetic instructions and scalar reciprocal estimate instructions when it is Negative overflow with Round toward +Infinity mode		
VSX Scalar Single-Precision Arithmetic instructions *VSX Scalar Reciprocal Estimate Single-Precision instruction*	OE = 0 OX = 1 Round toward +Infinity Negative overflow	XX = 1 VSR[XT].doubleword[0]= most negative finite number in double-precision format VSR[XT].doubleword[1] is undefined

Instructions tested	Compliance conditions	Expected result
VSX Scalar Double-Precision Arithmetic instructions *VSX Scalar Reciprocal Estimate Double-Precision instruction*		FR is undefined FI = 1 FPRF is set to indicate -Normal Number
I-7.4.3.OE0.14 The actions that will be taken when the Overflow Exception is disabled for the scalar arithmetic instructions and scalar reciprocal estimate instructions when it is Positive overflow with Round toward +Infinity mode		
VSX Scalar Single-Precision Arithmetic instructions *VSX Scalar Reciprocal Estimate Single-Precision instruction* *VSX Scalar Double-Precision Arithmetic instructions* *VSX Scalar Reciprocal Estimate Double-Precision instruction*	OE = 0 OX = 1 Round toward +Infinity Positive overflow	XX = 1 VSR[XT].doubleword[0]= +Infinity in double-precision format VSR[XT].doubleword[1] is undefined FR is undefined FI = 1 FPRF is set to indicate +Infinity
I-7.4.3.OE0.15 The actions that will be taken when the Overflow Exception is disabled for the scalar arithmetic instructions and scalar reciprocal estimate instructions when it is Negative overflow with Round toward -Infinity mode		
VSX Scalar Single-Precision Arithmetic instructions *VSX Scalar Reciprocal Estimate Single-Precision instruction* *VSX Scalar Double-Precision Arithmetic instructions* *VSX Scalar Reciprocal Estimate Double-Precision instruction*	OE = 0 OX = 1 Round toward -Infinity Negative overflow	XX = 1 VSR[XT].doubleword[0]= -Infinity in double-precision format VSR[XT].doubleword[1] is undefined FR is undefined FI = 1 FPRF is set to indicate -Infinity
I-7.4.3.OE0.16 The actions that will be taken when the Overflow Exception is disabled for the scalar arithmetic instructions and scalar reciprocal estimate instructions when it is Positive overflow with Round toward -Infinity mode		
VSX Scalar Single-Precision Arithmetic instructions *VSX Scalar Reciprocal Estimate Single-Precision instruction* *VSX Scalar Double-Precision Arithmetic instructions* *VSX Scalar Reciprocal Estimate Double-Precision instruction*	OE = 0 OX = 1 Round toward -Infinity Positive overflow	XX = 1 VSR[XT].doubleword[0]= most positive finite number in double-precision format VSR[XT].doubleword[1] is undefined FR is undefined FI = 1 FPRF is set to indicate +Normal Number
I-7.4.3.OE0.17 The actions that will be taken when the Overflow Exception is disabled for the *VSX Vector Double-Precision Arithmetic instructions* and *VSX Vector Reciprocal Estimate Double-Precision instruction* when it is Negative overflow with Round to Nearest Even mode		
VSX Vector Double-Precision Arithmetic instructions *VSX Vector Reciprocal Estimate Double-Precision instruction*	OE = 0 OX = 1 Round to Nearest Even Negative overflow Vector element i is causing an Overflow exception	XX = 1 VSR[XT].doubleword[i] = -Infinity in double-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.3.OE0.18 The actions that will be taken when the Overflow Exception is disabled for the *VSX Vector Double-Precision Arithmetic instructions* and *VSX Vector Reciprocal Estimate Double-Precision instruction* when it is Positive overflow with Round to Nearest Even mode		

Instructions tested	Compliance conditions	Expected result
VSX Vector Double-Precision Arithmetic instructions *VSX Vector Reciprocal Estimate Double-Precision instruction*	OE = 0 OX = 1 Round to Nearest Even Positive overflow Vector element i is causing an Overflow exception	XX = 1 VSR[XT].doubleword[i] = +Infinity in double-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.3.OE0.19 The actions that will be taken when the Overflow Exception is disabled for the *VSX Vector Double-Precision Arithmetic instructions* and *VSX Vector Reciprocal Estimate Double-Precision instruction* when it is Negative overflow with Round toward Zero mode		
VSX Vector Double-Precision Arithmetic instructions *VSX Vector Reciprocal Estimate Double-Precision instruction*	OE = 0 OX = 1 Round toward Zero Negative overflow Vector element i is causing an Overflow exception	XX = 1 VSR[XT].doubleword[i] = most negative finite number in double-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.3.OE0.20 The actions that will be taken when the Overflow Exception is disabled for the *VSX Vector Double-Precision Arithmetic instructions* and *VSX Vector Reciprocal Estimate Double-Precision instruction* when it is Positive overflow with Round toward Zero mode		
VSX Vector Double-Precision Arithmetic instructions *VSX Vector Reciprocal Estimate Double-Precision instruction*	OE = 0 OX = 1 Round toward Zero Positive overflow Vector element i is causing an Overflow exception	XX = 1 VSR[XT].doubleword[i] = most positive finite number in double-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.3.OE0.21 The actions that will be taken when the Overflow Exception is disabled for the *VSX Vector Double-Precision Arithmetic instructions* and *VSX Vector Reciprocal Estimate Double-Precision instruction* when it is Negative overflow with Round toward +Infinity mode		
VSX Vector Double-Precision Arithmetic instructions *VSX Vector Reciprocal Estimate Double-Precision instruction*	OE = 0 OX = 1 Round toward +Infinity Negative overflow Vector element i is causing an Overflow exception	XX = 1 VSR[XT].doubleword[i] = most negative finite number in double-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.3.OE0.22 The actions that will be taken when the Overflow Exception is disabled for the *VSX Vector Double-Precision Arithmetic instructions* and *VSX Vector Reciprocal Estimate Double-Precision instruction* when it is Positive overflow with Round toward +Infinity mode		
VSX Vector Double-Precision Arithmetic instructions *VSX Vector Reciprocal Estimate Double-Precision instruction*	OE = 0 OX = 1 Round toward +Infinity Positive overflow Vector element i is causing an Overflow exception	XX = 1 VSR[XT].doubleword[i] = +Infinity in double-precision format FR is not modified FI is not modified FPRF is not modified

Instructions tested	Compliance conditions	Expected result
I-7.4.3.OE0.23 The actions that will be taken when the Overflow Exception is disabled for the *VSX Vector Double-Precision Arithmetic instructions* and *VSX Vector Reciprocal Estimate Double-Precision instruction* when it is Negative overflow with Round toward -Infinity mode		
VSX Vector Double-Precision Arithmetic instructions *VSX Vector Reciprocal Estimate Double-Precision instruction*	OE = 0 OX = 1 Round toward -Infinity Negative overflow Vector element i is causing an Overflow exception	XX = 1 VSR[XT].doubleword[i] = -Infinity in double-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.3.OE0.24 The actions that will be taken when the Overflow Exception is disabled for the *VSX Vector Double-Precision Arithmetic instructions* and *VSX Vector Reciprocal Estimate Double-Precision instruction* when it is Positive overflow with Round toward -Infinity mode		
VSX Vector Double-Precision Arithmetic instructions *VSX Vector Reciprocal Estimate Double-Precision instruction*	OE = 0 OX = 1 Round toward -Infinity Positive overflow Vector element i is causing an Overflow exception	XX = 1 VSR[XT].doubleword[i] = most positive finite number in double-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.3.OE0.25 The actions that will be taken when the Overflow Exception is disabled for the *VSX Vector Single-Precision Arithmetic instructions, VSX Vector Reciprocal Estimate Single-Precision instruction,* and *VSX Vector round and Convert Double-Precision to Single-Precision format instruction* when it is Negative overflow with Round to Nearest Even mode		
VSX Vector Single-Precision Arithmetic instructions *VSX Vector Reciprocal Estimate Single-Precision instruction* xvcvdpsp	OE = 0 OX = 1 Round to Nearest Even Negative overflow Vector element i is causing an Overflow exception	XX = 1 VSR[XT].word[i] = -Infinity in single-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.3.OE0.26 The actions that will be taken when the Overflow Exception is disabled for the *VSX Vector Single-Precision Arithmetic instructions, VSX Vector Reciprocal Estimate Single-Precision instruction,* and *VSX Vector round and Convert Double-Precision to Single-Precision format instruction* when it is Positive overflow with Round to Nearest Even mode		
VSX Vector Single-Precision Arithmetic instructions *VSX Vector Reciprocal Estimate Single-Precision instruction* xvcvdpsp	OE = 0 OX = 1 Round to Nearest Even Positive overflow Vector element i is causing an Overflow exception	XX = 1 VSR[XT].word[i] = +Infinity in single-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.3.OE0.27 The actions that will be taken when the Overflow Exception is disabled for the *VSX Vector Single-Precision Arithmetic instructions, VSX Vector Reciprocal Estimate Single-Precision instruction,* and *VSX Vector round and Convert Double-Precision to Single-Precision format instruction* when it is Negative overflow with Round toward Zero mode		
VSX Vector Single-Precision Arithmetic instructions *VSX Vector Reciprocal Estimate Single-Precision instruction* xvcvdpsp	OE = 0 OX = 1 Round toward Zero Negative overflow	XX = 1 VSR[XT].word[i] = most negative finite number in single-precision format FR is not modified FI is not modified

Instructions tested	Compliance conditions	Expected result
	Vector element i is causing an Overflow exception	FPRF is not modified
I-7.4.3.OE0.28 The actions that will be taken when the Overflow Exception is disabled for the *VSX Vector Single-Precision Arithmetic instructions, VSX Vector Reciprocal Estimate Single-Precision instruction,* and *VSX Vector round and Convert Double-Precision to Single-Precision format instruction* when it is Positive overflow with Round toward Zero mode		
VSX Vector Single-Precision Arithmetic instructions *VSX Vector Reciprocal Estimate Single-Precision instruction* xvcvdpsp	OE = 0 OX = 1 Round toward Zero Positive overflow Vector element i is causing an Overflow exception	XX = 1 VSR[XT].word[i] = most positive finite number in single-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.3.OE0.29 The actions that will be taken when the Overflow Exception is disabled for the *VSX Vector Single-Precision Arithmetic instructions, VSX Vector Reciprocal Estimate Single-Precision instruction,* and *VSX Vector round and Convert Double-Precision to Single-Precision format instruction* when it is Negative overflow with Round toward +Infinity mode		
VSX Vector Single-Precision Arithmetic instructions *VSX Vector Reciprocal Estimate Single-Precision instruction* xvcvdpsp	OE = 0 OX = 1 Round toward +Infinity Negative overflow Vector element i is causing an Overflow exception	XX = 1 VSR[XT].word[i] = most negative finite number in single-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.3.OE0.30 The actions that will be taken when the Overflow Exception is disabled for the *VSX Vector Single-Precision Arithmetic instructions, VSX Vector Reciprocal Estimate Single-Precision instruction,* and *VSX Vector round and Convert Double-Precision to Single-Precision format instruction* when it is Positive overflow with Round toward +Infinity mode		
VSX Vector Single-Precision Arithmetic instructions *VSX Vector Reciprocal Estimate Single-Precision instruction* xvcvdpsp	OE = 0 OX = 1 Round toward +Infinity Positive overflow Vector element i is causing an Overflow exception	XX = 1 VSR[XT].word[i] = +Infinity in single-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.3.OE0.31 The actions that will be taken when the Overflow Exception is disabled for the *VSX Vector Single-Precision Arithmetic instructions, VSX Vector Reciprocal Estimate Single-Precision instruction,* and *VSX Vector round and Convert Double-Precision to Single-Precision format instruction* when it is Negative overflow with Round toward -Infinity mode		
VSX Vector Single-Precision Arithmetic instructions *VSX Vector Reciprocal Estimate Single-Precision instruction* xvcvdpsp	OE = 0 OX = 1 Round toward -Infinity Negative overflow Vector element i is causing an Overflow exception	XX = 1 VSR[XT].word[i] = -Infinity in single-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.3.OE0.32 The actions that will be taken when the Overflow Exception is disabled for the *VSX Vector Single-Precision Arithmetic instructions, VSX Vector Reciprocal Estimate Single-Precision instruction,* and *VSX Vector round and Convert Double-Precision to Single-Precision format instruction* when it is Positive overflow with Round toward -Infinity mode		
VSX Vector Single-Precision Arithmetic instructions *VSX Vector Reciprocal Estimate Single-Precision instruction* xvcvdpsp	OE = 0 OX = 1 Round toward -Infinity Positive overflow	XX = 1 VSR[XT].word[i] = - most positive finite number in single-precision format FR is not modified

Instructions tested	Compliance conditions	Expected result
	Vector element i is causing an Overflow exception	FI is not modified FPRF is not modified
I-7.4.3.OE0.33 The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Quad-Precision Arithmetic instructions* and *VSX Scalar Round Quad-Precision to Double-Extended-Precision instruction* when it is Negative overflow with Round to Nearest Even mode		
xsaddqp[o] xsdivqp[o] xsmulqp[o] xssqrtqp[o] xssubqp[o] xsmaddqp[o] xsmsubqp[o] xsnmaddqp[o] xsnmsubqp[o] xsrqpxp	OE = 0 OX = 1 Round to Nearest Even Negative overflow	XX = 1 VSR[VRT+32] = -Infinity in quad-precision format FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (-Infinity)
I-7.4.3.OE0.33a The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Quad-Precision Arithmetic instructions* and *VSX Scalar Round Quad-Precision to Double-Extended-Precision instruction* when it is Positive overflow with Round to Nearest Even mode		
xsaddqp[o] xsdivqp[o] xsmulqp[o] xssqrtqp[o] xssubqp[o] xsmaddqp[o] xsmsubqp[o] xsnmaddqp[o] xsnmsubqp[o] xsrqpxp	OE = 0 OX = 1 Round to Nearest Even Positive overflow	XX = 1 VSR[VRT+32] = +Infinity in quad-precision format FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (+Infinity)
I-7.4.3.OE0.33b The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Quad-Precision Arithmetic instructions* and *VSX Scalar Round Quad-Precision to Double-Extended-Precision instruction* when it is Negative overflow with Round toward Zero mode		
xsaddqp[o] xsdivqp[o] xsmulqp[o] xssqrtqp[o] xssubqp[o] xsmaddqp[o] xsmsubqp[o] xsnmaddqp[o] xsnmsubqp[o] xsrqpxp	OE = 0 OX = 1 Round toward Zero Negative overflow	XX = 1 VSR[VRT+32] = most negative finite number in quad-precision format FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (-Normal Number)
I-7.4.3.OE0.33c The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Quad-Precision Arithmetic instructions* and *VSX Scalar Round Quad-Precision to Double-Extended-Precision instruction* when it is Positive overflow with Round toward Zero mode		
xsaddqp[o] xsdivqp[o] xsmulqp[o] xssqrtqp[o] xssubqp[o] xsmaddqp[o] xsmsubqp[o] xsnmaddqp[o] xsnmsubqp[o] xsrqpxp	OE = 0 OX = 1 Round toward Zero Positive overflow	XX = 1 VSR[VRT+32] = most positive finite number in quad-precision format FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (+Normal Number)
I-7.4.3.OE0.33d The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Quad-Precision Arithmetic instructions* and *VSX Scalar Round Quad-Precision to Double-Extended-Precision instruction* when it is Negative overflow with Round toward +Infinity mode		
xsaddqp[o] xsdivqp[o] xsmulqp[o] xssqrtqp[o] xssubqp[o] xsmaddqp[o]	OE = 0	XX = 1

Instructions tested	Compliance conditions	Expected result
xsmsubqp[o] xsnmaddqp[o] xsnmsubqp[o] xsrqpxp	OX = 1 Round toward +Infinity Negative overflow	VSR[VRT+32] = most negative finite number in quad-precision format FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (-Normal Number)
I-7.4.3.OE0.33e The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Quad-Precision Arithmetic instructions* and *VSX Scalar Round Quad-Precision to Double-Extended-Precision instruction* when it is Positive overflow with Round toward +Infinity mode		
xsaddqp[o] xsdivqp[o] xsmulqp[o] xssqrtqp[o] xssubqp[o] xsmaddqp[o] xsmsubqp[o] xsnmaddqp[o] xsnmsubqp[o] xsrqpxp	OE = 0 OX = 1 Round toward +Infinity Positive overflow	XX = 1 VSR[VRT+32] = +Infinity in quad-precision format FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (+Infinity)
I-7.4.3.OE0.33f The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Quad-Precision Arithmetic instructions* and *VSX Scalar Round Quad-Precision to Double-Extended-Precision instruction* when it is Negative overflow with Round toward -Infinity mode		
xsaddqp[o] xsdivqp[o] xsmulqp[o] xssqrtqp[o] xssubqp[o] xsmaddqp[o] xsmsubqp[o] xsnmaddqp[o] xsnmsubqp[o] xsrqpxp	OE = 0 OX = 1 Round toward -Infinity Negative overflow	XX = 1 VSR[VRT+32] = -Infinity in quad-precision format FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (-Infinity)
I-7.4.3.OE0.33g The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Quad-Precision Arithmetic instructions* and *VSX Scalar Round Quad-Precision to Double-Extended-Precision instruction* when it is Positive overflow with Round toward -Infinity mode		
xsaddqp[o] xsdivqp[o] xsmulqp[o] xssqrtqp[o] xssubqp[o] xsmaddqp[o] xsmsubqp[o] xsnmaddqp[o] xsnmsubqp[o] xsrqpxp	OE = 0 OX = 1 Round toward -Infinity Positive overflow	XX = 1 VSR[VRT+32] = most positive finite number in quad-precision format FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (+Normal Number)
I-7.4.3.OE0.34 The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Convert with round Quad-Precision to Double-Precision format instruction* when it is Negative overflow with Round to Nearest Even mode		
xscvqpdp	OE = 0 OX = 1 Round to Nearest Even Negative overflow	XX = 1 VSR[VRT+32].doubleword[0] = -Infinity in double-precision format VSR[VRT+32].doubleword[1] = 0x0000_0000_0000_0000 FR is undefined FI = 1

Instructions tested	Compliance conditions	Expected result
		FPRF is set to indicate class and sign of the result (-Infinity)
I-7.4.3.OE0.34a The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Convert with round Quad-Precision to Double-Precision format instruction* when it is Positive overflow with Round to Nearest Even mode		
xscvqpdp	OE = 0 OX = 1 Round to Nearest Even Positive overflow	XX = 1 VSR[VRT+32].doubleword[0] = +Infinity in double-precision format VSR[VRT+32].doubleword[1] = 0x0000_0000_0000_0000 FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (+Infinity)
I-7.4.3.OE0.34b The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Convert with round Quad-Precision to Double-Precision format instruction* when it is Negative overflow with Round toward Zero mode		
xscvqpdp	OE = 0 OX = 1 Round toward Zero Negative overflow	XX = 1 VSR[VRT+32].doubleword[0] = most negative finite number in double-precision format VSR[VRT+32].doubleword[1] = 0x0000_0000_0000_0000 FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (-Normal Number)
I-7.4.3.OE0.34c The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Convert with round Quad-Precision to Double-Precision format instruction* when it is Positive overflow with Round toward Zero mode		
xscvqpdp	OE = 0 OX = 1 Round toward Zero Positive overflow	XX = 1 VSR[VRT+32].doubleword[0] = most positive finite number in double-precision format VSR[VRT+32].doubleword[1] = 0x0000_0000_0000_0000 FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (+Normal Number)
I-7.4.3.OE0.34d The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Convert with round Quad-Precision to Double-Precision format instruction* when it is Negative overflow with Round toward +Infinity mode		
xscvqpdp	OE = 0 OX = 1 Round toward +Infinity Negative overflow	XX = 1 VSR[VRT+32].doubleword[0] = most negative finite number in double-precision format VSR[VRT+32].doubleword[1] = 0x0000_0000_0000_0000 FR is undefined

Instructions tested	Compliance conditions	Expected result
		FI = 1 FPRF is set to indicate class and sign of the result (-Normal Number)
I-7.4.3.OE0.34e The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Convert with round Quad-Precision to Double-Precision format instruction* when it is Positive overflow with Round toward +Infinity mode		
xscvqpdp	OE = 0 OX = 1 Round toward +Infinity Positive overflow	XX = 1 VSR[VRT+32].doubleword[0] = +Infinity in double-precision format VSR[VRT+32].doubleword[1] = 0x0000_0000_0000_0000 FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (+Infinity)
I-7.4.3.OE0.34f The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Convert with round Quad-Precision to Double-Precision format instruction* when it is Negative overflow with Round toward -Infinity mode		
xscvqpdp	OE = 0 OX = 1 Round toward -Infinity Negative overflow	XX = 1 VSR[VRT+32].doubleword[0] = -Infinity in double-precision format VSR[VRT+32].doubleword[1] = 0x0000_0000_0000_0000 FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (-Infinity)
I-7.4.3.OE0.34g The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Convert with round Quad-Precision to Double-Precision format instruction* when it is Positive overflow with Round toward -Infinity mode		
xscvqpdp	OE = 0 OX = 1 Round toward -Infinity Positive overflow	XX = 1 VSR[VRT+32].doubleword[0] = most positive finite number in double-precision format VSR[VRT+32].doubleword[1] = 0x0000_0000_0000_0000 FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (+Normal Number)
I-7.4.3.OE0.35 The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Convert with round Double-Precision to Half-Precision format instruction* when it is Negative overflow with Round to Nearest Even mode		
xscvdphp	OE = 0 OX = 1 Round to Nearest Even Negative overflow	XX = 1 VSR[XT] rightmost halfword of double-word[0] = -Infinity in half-precision format VSR[XT] leftmost 3 halfwords of double-word[0] are set to 0 VSR[XT] doubleword[1] is undefined

Instructions tested	Compliance conditions	Expected result
		FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (-Infinity)
I-7.4.3.OE0.35a The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Convert with round Double-Precision to Half-Precision format instruction* when it is Positive overflow with Round to Nearest Even mode		
xscvdphp	OE = 0 OX = 1 Round to Nearest Even Positive overflow	XX = 1 VSR[XT] rightmost halfword of double-word[0] = +Infinity in half-precision format VSR[XT] leftmost 3 halfwords of double-word[0] are set to 0 VSR[XT] doubleword[1] is undefined FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (+Infinity)
I-7.4.3.OE0.35b The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Convert with round Double-Precision to Half-Precision format instruction* when it is Negative overflow with Round toward Zero mode		
xscvdphp	OE = 0 OX = 1 Round toward Zero Negative overflow	XX = 1 VSR[XT] rightmost halfword of double-word[0] = most negative finite number in half-precision format VSR[XT] leftmost 3 halfwords of double-word[0] are set to 0 VSR[XT] doubleword[1] is undefined FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (-Normal Number)
I-7.4.3.OE0.35c The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Convert with round Double-Precision to Half-Precision format instruction* when it is Positive overflow with Round toward Zero mode		
xscvdphp	OE = 0 OX = 1 Round toward Zero Positive overflow	XX = 1 VSR[XT] rightmost halfword of double-word[0] = most positive finite number in half-precision format VSR[XT] leftmost 3 halfwords of double-word[0] are set to 0 VSR[XT] doubleword[1] is undefined FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (+Normal Number)
I-7.4.3.OE0.35d The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Convert with round Double-Precision to Half-Precision format instruction* when it is Negative overflow with Round toward +Infinity mode		
xscvdphp	OE = 0	XX = 1

Instructions tested	Compliance conditions	Expected result
	OX = 1 Round toward +Infinity Negative overflow	VSR[XT] rightmost halfword of double-word[0] = most negative finite number in half-precision format VSR[XT] leftmost 3 halfwords of double-word[0] are set to 0 VSR[XT] doubleword[1] is undefined FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (-Normal Number)
I-7.4.3.OE0.35e The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Convert with round Double-Precision to Half-Precision format instruction* when it is Positive overflow with Round toward +Infinity mode		
xscvdphp	OE = 0 OX = 1 Round toward +Infinity Positive overflow	XX = 1 VSR[XT] rightmost halfword of double-word[0] = +Infinity in half-precision format VSR[XT] leftmost 3 halfwords of double-word[0] are set to 0 VSR[XT] doubleword[1] is undefined FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (+Infinity)
I-7.4.3.OE0.35f The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Convert with round Double-Precision to Half-Precision format instruction* when it is Negative overflow with Round toward -Infinity mode		
xscvdphp	OE = 0 OX = 1 Round toward -Infinity Negative overflow	XX = 1 VSR[XT] rightmost halfword of double-word[0] = -Infinity in half-precision format VSR[XT] leftmost 3 halfwords of double-word[0] are set to 0 VSR[XT] doubleword[1] is undefined FR is undefined FI = 1 FPRF is set to indicate class and sign of the result (-Infinity)
I-7.4.3.OE0.35g The actions that will be taken when the Overflow Exception is disabled for *VSX Scalar Convert with round Double-Precision to Half-Precision format instruction* when it is Positive overflow with Round toward -Infinity mode		
xscvdphp	OE = 0 OX = 1 Round toward -Infinity Positive overflow	XX = 1 VSR[XT] rightmost halfword of double-word[0] = most positive finite number in half-precision format VSR[XT] leftmost 3 halfwords of double-word[0] are set to 0 VSR[XT] doubleword[1] is undefined FR is undefined

Instructions tested	Compliance conditions	Expected result
		FI = 1 FPRF is set to indicate class and sign of the result (+Normal Number)
I-7.4.3.OE0.36 The actions that will be taken when the Overflow Exception is disabled for *VSX Vector Convert with round Single-Precision to Half-Precision format instruction* when it is Negative overflow with Round to Nearest Even mode		
xvcvsphp	OE = 0 OX = 1 Round to Nearest Even Negative overflow	XX = 1 VSR[XT] rightmost halfword of respective word element = -Infinity in half-precision representation VSR[XT] leftmost halfword of respective word element is set to 0 FR and FI are not modified FPRF is not modified
I-7.4.3.OE0.36a The actions that will be taken when the Overflow Exception is disabled for *VSX Vector Convert with round Single-Precision to Half-Precision format instruction* when it is Positive overflow with Round to Nearest Even mode		
xvcvsphp	OE = 0 OX = 1 Round to Nearest Even Positive overflow	XX = 1 VSR[XT] rightmost halfword of respective word element = +Infinity in half-precision representation VSR[XT] leftmost halfword of respective word element is set to 0 FR and FI are not modified FPRF is not modified
I-7.4.3.OE0.36b The actions that will be taken when the Overflow Exception is disabled for *VSX Vector Convert with round Single-Precision to Half-Precision format instruction* when it is Negative overflow with Round toward Zero mode		
xvcvsphp	OE = 0 OX = 1 Round toward Zero Negative overflow	XX = 1 VSR[XT] rightmost halfword of respective word element = most negative finite number in half-precision representation VSR[XT] leftmost halfword of respective word element is set to 0 FR and FI are not modified FPRF is not modified
I-7.4.3.OE0.36c The actions that will be taken when the Overflow Exception is disabled for *VSX Vector Convert with round Single-Precision to Half-Precision format instruction* when it is Positive overflow with Round toward Zero mode		
xvcvsphp	OE = 0 OX = 1 Round toward Zero Positive overflow	XX = 1 VSR[XT] rightmost halfword of respective word element = most positive finite number in half-precision representation VSR[XT] leftmost halfword of respective word element is set to 0 FR and FI are not modified FPRF is not modified
I-7.4.3.OE0.36d The actions that will be taken when the Overflow Exception is disabled for *VSX Vector Convert with round Single-Precision to Half-Precision format instruction* when it is Negative overflow with Round toward +Infinity mode		
xvcvsphp	OE = 0	XX = 1

Instructions tested	Compliance conditions	Expected result
	OX = 1 Round toward +Infinity Negative overflow	VSR[XT] rightmost halfword of respective word element = most negative finite number in half-precision representation VSR[XT] leftmost halfword of respective word element is set to 0 FR and FI are not modified FPRF is not modified
I-7.4.3.OE0.36e The actions that will be taken when the Overflow Exception is disabled for *VSX Vector Convert with round Single-Precision to Half-Precision format instruction* when it is Positive overflow with Round toward +Infinity mode		
xvcvsphp	OE = 0 OX = 1 Round toward +Infinity Positive overflow	XX = 1 VSR[XT] rightmost halfword of respective word element = +Infinity in half-precision representation VSR[XT] leftmost halfword of respective word element is set to 0 FR and FI are not modified FPRF is not modified
I-7.4.3.OE0.36f The actions that will be taken when the Overflow Exception is disabled for *VSX Vector Convert with round Single-Precision to Half-Precision format instruction* when it is Negative overflow with Round toward -Infinity mode		
xvcvsphp	OE = 0 OX = 1 Round toward -Infinity Negative overflow	XX = 1 VSR[XT] rightmost halfword of respective word element = -Infinity in half-precision representation VSR[XT] leftmost halfword of respective word element is set to 0 FR and FI are not modified FPRF is not modified
I-7.4.3.OE0.36g The actions that will be taken when the Overflow Exception is disabled for *VSX Vector Convert with round Single-Precision to Half-Precision format instruction* when it is Positive overflow with Round toward -Infinity mode		
xvcvsphp	OE = 0 OX = 1 Round toward -Infinity Positive overflow	XX = 1 VSR[XT] rightmost halfword of respective word element = most positive finite number in half-precision representation VSR[XT] leftmost halfword of respective word element is set to 0 FR and FI are not modified FPRF is not modified

8.2.4. Underflow Exception

Architecture sections:

- I-7.4.4 Floating-Point Underflow Exception

Scenario groups:

- Setting exception bit UX
- Actions taken when the exception is enabled

- Actions taken when the exception is disabled

Condition for enabled Underflow Exception: intermediate result is Tiny (detected before rounding, when a nonzero intermediate result would be less in magnitude than the smallest normalized number)

Condition for disabled Underflow Exception: intermediate result is Tiny (see above) and there is Loss of Accuracy (the delivered result value differs from the intermediate result)

VSX Scalar Double-Precision Arithmetic instructions: xsadddp xsdivdp xsmuldp xssubdp xsmaddadp xsmaddmdp xsmsubadp xsmsubmdp xsnmaddadp xsnmaddmdp xsnmsubadp xsnmsubmdp

VSX Scalar Reciprocal Estimate Double-Precision instruction: xsredp

VSX Scalar Single-Precision Arithmetic instructions: xsaddsp xsdivsp xsmulsp xssubsp xsmaddasp xsmaddmsp xsmsubasp xsmsubmsp xsnmaddasp xsnmaddmsp xsnmsubasp xsnmsubmsp

VSX Scalar Reciprocal Estimate Single-Precision instruction: xsresp

VSX Scalar Floating-Point Arithmetic instructions: *VSX Scalar Double-Precision Arithmetic instructions, VSX Scalar Single-Precision Arithmetic instructions*

VSX Vector Double-Precision Arithmetic instructions: xvadddp xvdivdp xvmuldp xvsubdp xvmaddadp xvmaddmdp xvmsubadp xvmsubmdp xvnmaddadp xvnmaddmdp xvnmsubadp xvnmsubmdp

VSX Vector Reciprocal Estimate Double-Precision instruction: xvredp

VSX Vector Single-Precision Arithmetic instructions: xvaddsp xvdivsp xvmulsp xvsubsp xvmaddasp xvmaddmsp xvmsubasp xvmsubmsp xvnmaddasp xvnmaddmsp xvnmsubasp xvnmsubmsp

VSX Vector Reciprocal Estimate Single-Precision instruction: xvresp

VSX Vector Floating-Point Arithmetic instructions: *VSX Vector Double-Precision Arithmetic instructions, VSX Vector Single-Precision Arithmetic instructions*

VSX Scalar Quad-Precision Arithmetic instructions: xsaddqp[o] xsdivqp[o] xsmulqp[o] xssqrtqp[o] xssubqp[o] xsmaddqp[o] xsmsubqp[o] xsnmaddqp[o] xsnmsubqp[o]

VSX Scalar Round Quad-Precision to Double-Extended-Precision instruction: xsrqpxp

VSX Scalar Convert with round Quad-Precision to Double-Precision format instructions: xscvqpdp[o]

VSX Scalar Convert with round Double-Precision to Half-Precision format instruction: xscvdphp

VSX Vector Convert with round Single-Precision to Half-Precision format instruction: xvcvsphp

8.2.4.1. Setting exception bit UX

VSX potentially overflowing/ underflowing instructions: *VSX Scalar Double-Precision Arithmetic instructions, VSX Scalar Reciprocal Estimate Double-Precision instruction, VSX Scalar Single-Precision Arithmetic instructions, VSX Scalar Reciprocal Estimate Single-Precision instruction, VSX Vector Double-Precision Arithmetic instructions, VSX Vector Reciprocal Estimate Double-Precision instruction, VSX Vector Single-Precision Arithmetic instructions, VSX Vector Reciprocal Estimate Single-Precision instruction, VSX Scalar Quad-Precision Arithmetic instructions, VSX Scalar Round Quad-Precision to Double-Extended-Precision instruction, VSX Scalar Convert with round Quad-Precision to Double-Precision format instruction, VSX Scalar Convert with round Double-Precision to*

Half-Precision format instruction, VSX Vector Convert with round Single-Precision to Half-Precision format instruction, xscvdpsp xvcvdpsp xssqrtsp xsrsp xsrsqrtesp

Instructions tested	Compliance conditions	Expected result
I-7.4.4.Under.1 UX is set to 1 when underflow occurs and Underflow Exception is enabled		
VSX potentially overflowing/ underflowing instructions	UE = 1 *Condition for enabled Underflow Exception* occurs	UX = 1
I-7.4.4.Under.2 UX is sticky enabled Underflow Exception		
Representative of *VSX potentially overflowing/ underflowing instructions*	UE = 1 UX=1 before instruction execution *Condition for enabled Underflow Exception* does **not** occur	UX = 1
I-7.4.4.Under.3 UX stays zero when exception condition **does not occur** enabled Underflow Exception		
Representative of *VSX potentially overflowing/ underflowing instructions*	UE = 1 UX=0 before instruction execution *Condition for enabled Underflow Exception* does **not** occur	UX = 0
I-7.4.4.Under.4 UX is set to 1 when underflow occurs and Underflow Exception is disabled		
VSX potentially overflowing/ underflowing instructions	UE = 0 *Condition for disabled Underflow Exception* occurs	UX = 1
I-7.4.4.Under.5 UX is sticky disabled Underflow exception		
Representative of *VSX potentially overflowing/ underflowing instructions*	UE = 0 UX=1 before instruction execution *Condition for disabled Underflow Exception* does **not** occur	UX = 1
I-7.4.4.Under.6 UX stays zero when exception condition **does not occur** disabled Underflow exception		
Representative of *VSX potentially overflowing/ underflowing instructions*	UE = 0 UX=0 before instruction execution *Condition for disabled Underflow Exception* does **not** occur	UX = 0
I-7.4.4.Under.7 If Underflow Exception is enabled, UX is set to 1 when result is Tiny, even if no loss of accuracy occurs		
Representative of *VSX potentially overflowing/ underflowing instructions*	UE = 1 Intermediate result is Tiny The delivered result is the same as the intermediate result	UX = 1

8.2.4.2. Actions taken when the exception is enabled

Instructions tested	Compliance conditions	Expected result
I-7.4.4.UE1.1 The actions that will be taken when the Underflow Exception is enabled for the xscvdpsp instruction		
xscvdpsp	UE = 1 UX = 1 The unbiased exponent of the normalized intermediate result is greater than or equal to Emin-192	The exponent is adjusted by adding 192 VSR[XT].word[0] = the rounded result in single-precision format VSR[XT].word[1] is undefined

Instructions tested	Compliance conditions	Expected result
	Condition for Underflow Exception occurs	VSR[XT].word[2] is undefined VSR[XT].word[3] is undefined FPRF is set to indicate Normal Number
I-7.4.4.UE1.2 The actions that will be taken when the Underflow Exception is enabled for the VSX Scalar Double-Precision Arithmetic instructions and VSX Scalar Reciprocal Estimate Double-Precision instruction		
VSX Scalar Double-Precision Arithmetic instructions xsredp	UE = 1 UX = 1 *Condition for Underflow Exception occurs*	The exponent of the normalized intermediate result is adjusted by adding 1536 VSR[XT].doubleword[0]= the adjusted rounded result VSR[XT].doubleword[1] is undefined FPRF is set to indicate Normal Number
I-7.4.4.UE1.3 The actions that will be taken when the Underflow Exception is enabled for the VSX Scalar Single-Precision Arithmetic instructions and VSX Scalar Reciprocal Estimate Single-Precision instruction		
VSX Scalar Single-Precision Arithmetic instructions xsresp	UE = 1 UX = 1 *Condition for Underflow Exception occurs*	The exponent of the normalized intermediate result is adjusted by adding 192 VSR[XT].doubleword[0]= the adjusted rounded result VSR[XT].doubleword[1] is undefined FPRF is set to indicate Normal Number
I-7.4.4.UE1.4 The actions that will be taken when the Underflow Exception is enabled for the VSX Vector Floating-Point Arithmetic instructions, VSX Vector Floating-Point Reciprocal Estimate instructions, and VSX Vector round and Convert Double-Precision to Single-Precision format instruction		
VSX Vector Floating-Point Arithmetic instructions xvredp xvresp xvcvdpsp	UE = 1 UX = 1 *Condition for Underflow Exception occurs*	Update of VSR[XT] is suppressed for all vector elements FR is not modified FI is not modified FPRF is not modified
I-7.4.4.UE1.5 The actions that will be taken when the Underflow Exception is enabled for *VSX Scalar Quad-Precision Arithmetic instructions* and *VSX Scalar Round Quad-Precision to Double-Extended-Precision instruction*		
xsaddqp[o] xsdivqp[o] xsmulqp[o] xssubqp[o] xsmaddqp[o] xsmsubqp[o] xsnmaddqp[o] xsnmsubqp[o] xsrqpxp	UE = 1 UX = 1 *Condition for Underflow Exception occurs*	Exponent of the normalized intermediate result is adjusted by adding 24576 VSR[VRT+32] = the adjusted, rounded result in quad-precision format FPRF is set to indicate Normal Number (unless the result is undefined)
I-7.4.4.UE1.6 The actions that will be taken when the Underflow Exception is enabled for *VSX Scalar Convert with round Quad-Precision to Double-Precision format [using round to Odd] instructions*		
xscvqpdp[o]	UE = 1 UX = 1 *Condition for Underflow Exception occurs*	Exponent of the normalized intermediate result is adjusted by adding 1536 If the adjusted exponent is less than -1022, the result is undefined VSR[VRT+32].doubleword[0] = the adjusted, rounded result in double-precision format VSR[VRT+32].doubleword[1] = 0x0000_0000_0000_0000

Instructions tested	Compliance conditions	Expected result
		FPRF is set to indicate Normal Number (unless the result is undefined)
I-7.4.4.UE1.7 The actions that will be taken when the Underflow Exception is enabled for *VSX Scalar Convert with round Double-Precision to Half-Precision format instruction*		
xscvdphp	UE = 1 UX = 1 *Condition for Underflow Exception occurs*	Exponent of the normalized intermediate result is adjusted by adding 24 If the adjusted exponent is less than -14, the adjusted, rounded result is undefined VSR[XT] rightmost halfword of double-word[0] = result in half-precision format VSR[XT] leftmost 3 halfwords of double-word[0] are set to 0 VSR[XT] doubleword[1] is undefined FPRF is set to indicate Normal Number (unless the result is undefined)
I-7.4.4.UE1.8 The actions that will be taken when the Underflow Exception is enabled for *VSX Vector Convert with round Single-Precision to Half-Precision format instruction*		
xvcvsphp	UE = 1 UX = 1 *Condition for Underflow Exception occurs*	VSR[XT] is not modified FR and FI are not modified FPRF is not modified

8.2.4.3. Actions taken when the exception is disabled

Instructions tested	Compliance conditions	Expected result
I-7.4.4.UE0.1 The actions that will be taken when the Underflow Exception is disabled for xscvdpsp instruction		
xscvdpsp	UE = 0 UX = 1 *Condition for Underflow Exception occurs*	VSR[XT].word[0] = the result in single-precision format VSR[XT].word[1] is undefined VSR[XT].word[2] is undefined VSR[XT].word[3] is undefined FPRF is set to indicate the class and sign of the result
I-7.4.4.UE0.2 The actions that will be taken when the Underflow Exception is disabled for VSX Scalar Floating-Point Arithmetic instructions		
VSX Scalar Floating-Point Arithmetic instructions xsredp xsresp	UE = 0 UX = 1 *Condition for Underflow Exception occurs*	VSR[XT].doubleword[0]= the result VSR[XT].doubleword[1] is undefined FPRF is set to indicate the class and sign of the result
I-7.4.4.UE0.3 The actions that will be taken when the Underflow Exception is disabled for VSX Vector Double-Precision Arithmetic instructions		
VSX Vector Double-Precision Arithmetic instructions *xvredp*	UE = 0 UX = 1 Vector element i is causing an Underflow exception	VSR[XT].doubleword[i]= the result in double-precision format FR is not modified FI is not modified FPRF is not modified

Instructions tested	Compliance conditions	Expected result
I-7.4.4.UE0.4 The actions that will be taken when the Underflow Exception is disabled for VSX Vector Single-Precision Arithmetic instructions		
VSX Vector Single-Precision Arithmetic instructions xvresp xvcvdpsp	UE = 0 UX = 1 Vector element i is causing an Underflow exception	VSR[XT].word[i]= the result in single-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.4.UE0.5 The actions that will be taken when the Underflow Exception is disabled for *VSX Scalar Quad-Precision Arithmetic instructions* and *VSX Scalar Round Quad-Precision to Double-Extended-Precision instruction*		
xsaddqp[o] xsdivqp[o] xsmulqp[o] xssubqp[o] xsmaddqp[o] xsmsubqp[o] xsnmaddqp[o] xsnmsubqp[o] xsrqpxp	UE = 0 UX = 1 *Condition for Underflow Exception occurs*	VSR[VRT+32] = result in quad-precision format FPRF is set to indicate class and sign of the result
I-7.4.4.UE0.6 The actions that will be taken when the Underflow Exception is disabled for *VSX Scalar Convert with round Quad-Precision to Double-Precision format instructions*		
xscvqpdp[o]	UE = 0 UX = 1 *Condition for Underflow Exception occurs*	VSR[VRT+32].doubleword[0] = result in double-precision format VSR[VRT+32].doubleword[1] = 0x0000_0000_0000_0000 FPRF is set to indicate class and sign of the result
I-7.4.4.UE0.7 The actions that will be taken when the Underflow Exception is disabled for *VSX Scalar Convert with round Double-Precision to Half-Precision format instruction*		
xscvdphp	UE = 0 UX = 1 *Condition for Underflow Exception occurs*	VSR[XT] rightmost halfword of double-word[0] = result in half-precision format VSR[XT] leftmost 3 halfwords of double-word[0] are set to 0 VSR[XT] doubleword[1] is undefined FPRF is set to indicate class and sign of the result
I-7.4.4.UE0.8 The actions that will be taken when the Underflow Exception is disabled for *VSX Vector Convert with round Single-Precision to Half-Precision format instruction*		
xvcvsphp	UE = 0 UX = 1 Vector element i is causing an Underflow exception	VSR[XT] rightmost halfword of word[i] = result in half-precision format VSR[XT] leftmost halfword of word[i] is set to 0 FR and FI are not modified FPRF is not modified

8.2.5. Inexact Exception

Architecture sections:

- I-7.4.5 Floating-Point Inexact Exception

Scenario groups:

- Setting exception bit XX
- Actions taken when the exception is enabled

- Actions taken when the exception is disabled

Condition for Inexact Exception due to rounding:

- The rounded result differs from the intermediate result
- No enabled Overflow or Underflow exception occurs

Condition for Inexact Exception due to overflow:

- The rounded result overflows
- Overflow Exception is disabled

VSX Vector round and Convert Double-Precision to Single-Precision format instruction: xscvdpsp

VSX Scalar Double-Precision Arithmetic instructions: xsadddp xsdivdp xsmuldp xssubdp xsmaddadp xsmaddmdp xsmsubadp xsmsubmdp xsnmaddadp xsnmaddmdp xsnmsubadp xsnmsubmdp

VSX Scalar Double-Precision Square Root instruction: xssqrtdp

VSX Scalar Single-Precision Arithmetic instructions: xsaddsp xsdivsp xsmulsp xssubsp xsmaddasp xsmaddmsp xsmsubasp xsmsubmsp xsnmaddasp xsnmaddmsp xsnmsubasp xsnmsubmsp

VSX Scalar Single-Precision Square Root instruction: xssqrtsp

VSX Scalar Floating-Point Arithmetic instructions: *VSX Scalar Double-Precision Arithmetic instructions, VSX Scalar Single-Precision Arithmetic instructions*

VSX Scalar Round to Single-Precision instruction: xsrsp

VSX Scalar Round to Double-Precision Integer Exact using Current rounding mode instruction: xsrdpic

VSX Scalar Integer to Double-Precision Format Conversion instructions: xscvsxddp xscvuxddp

VSX Scalar Integer to Single-Precision Format Conversion instructions: xscvsxdsp xscvuxdsp

VSX Scalar Integer to Floating-Point Format Conversion instructions: *VSX Scalar Integer to Double-Precision Format Conversion instructions, VSX Scalar Integer to Single-Precision Format Conversion instructions*

VSX Scalar Convert with round to zero Double-Precision to Signed Word format instruction: xscvdpsxws

VSX Scalar Convert with round to zero Double-Precision to Unsigned Word format instruction: xscvdpuxws

VSX Scalar Convert Double-Precision Floating-Point to Integer Word instructions: *VSX Scalar Convert with round to zero Double-Precision to Signed Word format instruction, VSX Scalar Convert with round to zero Double-Precision to Unsigned Word format instruction*

VSX Vector Double-Precision Arithmetic instructions: xvadddp xvsubdp xvmuldp xvdivdp xvmaddadp xvmaddmdp xvmsubadp xvmsubmdp xvnmaddadp xvnmaddmdp xvnmsubadp xvnmsubmdp

VSX Vector Double-Precision Square Root instruction: xvsqrtdp

VSX Vector Single-Precision Arithmetic instructions: xvaddsp xvsubsp xvmulsp xvdivsp xvmaddasp xvmaddmsp xvmsubasp xvmsubmsp xvnmaddasp xvnmaddmsp xvnmsubasp xvnmsubmsp

VSX Vector Single-Precision Square Root instruction: xvsqrtsp

VSX Vector Floating-Point Arithmetic instructions: *VSX Vector Double-Precision Arithmetic instructions, VSX Vector Single-Precision Arithmetic instructions*

VSX Vector Floating-Point Reciprocal Estimate instructions: xvredp xvresp

VSX Vector round and Convert Double-Precision to Single-Precision format instruction: xvcvdpsp

VSX Vector Double-Precision to Integer Format Conversion instructions: xvcvdpsxds xvcvdpsxws xvcvdpuxds xvcvdpuxws

VSX Vector Single-Precision to Integer Format Conversion instructions: xvcvspsxds xvcvspsxws xvcvspuxds xvcvspuxws

VSX Vector Integer to Floating-Point Format Conversion instructions: xvcvsxddp xvcvuxddp xvcvsxdsp xvcvuxdsp xvcvsxwsp xvcvuxwsp

VSX Scalar Quad-Precision Arithmetic instructions: xsaddqp[o] xsdivqp[o] xsmulqp[o] xssubqp[o] xsmaddqp[o] xsmsubqp[o] xsnmaddqp[o] xsnmsubqp[o]

VSX Scalar Quad-Precision Square Root instructions: xssqrtqp[o]

VSX Scalar Round Quad-Precision to Double-Extended-Precision instruction: xsrqpi

VSX Scalar Round to Quad-Precision Integer instruction: xsrqpxp

VSX Scalar Round Quad-Precision instructions: *VSX Scalar Round Quad-Precision to Double-Extended-Precision instruction, VSX Scalar Round to Quad-Precision Integer instruction*

VSX Scalar Convert with round Quad-Precision to Double-Precision format instructions: xscvqpdp[o]

VSX Scalar truncate & Convert Quad-Precision to Signed Doubleword instruction: xscvqpsdz

VSX Scalar truncate & Convert Quad-Precision to Signed Word instruction: xscvqpswz

VSX Scalar truncate & Convert Quad-Precision to Unsigned Doubleword instruction: xscvqpudz

VSX Scalar truncate & Convert Quad-Precision to Unsigned Word instruction: xscvqpuwz

VSX Scalar Convert with round Double-Precision to Half-Precision format instruction: xscvdphp

VSX Vector Convert with round Single-Precision to Half-Precision format instruction: xvcvsphp

8.2.5.1. Setting exception bit XX

VSX potentially inexact instructions: *VSX Vector round and Convert Double-Precision to Single-Precision format instruction, VSX Scalar Double-Precision Arithmetic instructions, VSX Scalar Single-Precision Arithmetic instructions, VSX Scalar Round to Single-Precision instruction, VSX Scalar Round to Double-Precision Integer Exact using Current rounding mode instruction, VSX Scalar Integer to Double-Precision Format Conversion instructions, VSX Scalar Integer to Single-Precision Format Conversion instructions, VSX Scalar Convert with round to zero Double-Precision to Signed Word format instruction, VSX Scalar Convert with round to zero Double-Precision to Unsigned Word format instruction, VSX Vector Double-Precision Arithmetic instructions, VSX Vector Double-Precision Square Root instruction, VSX Vector Single-Precision Arithmetic instructions, VSX Vector Single-Precision Square Root instruction, VSX Vector Floating-Point Reciprocal Estimate*

instructions, VSX Vector round and Convert Double-Precision to Single-Precision format instruction, VSX Vector Double-Precision to Integer Format Conversion instructions, VSX Vector Integer to Floating-Point Format Conversion instructions,VSX Scalar Quad-Precision Arithmetic instructions, VSX Scalar Quad-Precision Square Root instruction, VSX Scalar Round Quad-Precision to Double-Extended-Precision instruction, VSX Scalar Round to Quad-Precision Integer instruction, VSX Scalar Convert with round Quad-Precision to Double-Precision instruction, VSX Scalar truncate & Convert Quad-Precision to Signed Doubleword instruction, VSX Scalar truncate & Convert Quad-Precision to Signed Word instruction, VSX Scalar truncate & Convert Quad-Precision to Unsigned Doubleword instruction, VSX Scalar truncate & Convert Quad-Precision to Unsigned Word instruction, VSX Scalar Convert with round Double-Precision to Half-Precision truncate instruction, VSX Vector Convert with round Single-Precision to Half-Precision format instruction

Instructions tested	**Compliance conditions**	**Expected result**
I-7.4.5.Inexact.1 XX is set to 1 when rounded result differs from intermediate result		
VSX potentially inexact instructions	*Condition for Inexact Exception due to rounding* occurs	XX = 1
I-7.4.5.Inexact.2 XX is set to 1 when rounded result overflows and Overflow Exception is disabled		
VSX potentially inexact instructions	*Condition for Inexact Exception due to overflow* occurs	XX = 1
I-7.4.5.Inexact.3 XX is sticky		
Representative of *VSX potentially inexact instructions*	XX=1 before instruction execution *Condition for Inexact Exception due to rounding* does **not** occur *Condition for Inexact Exception due to overflow* does **not** occur	XX = 1
I-7.4.5.Inexact.4 XX stays zero when exception condition **does not occur**		
Representative of *VSX potentially inexact instructions*	XX=0 before instruction execution *Condition for Inexact Exception due to rounding* does **not** occur *Condition for Inexact Exception due to overflow* does **not** occur	XX = 0

8.2.5.2. Actions taken when the exception is enabled

Instructions tested	**Compliance conditions**	**Expected result**
I-7.4.5.XE1.1 The actions that will be taken when the Inexact Exception is enabled for the xscvdpsp instruction		
xscvdpsp	XE = 1 XX = 1 *Condition for Inexact Exception occurs*	VSR[XT].word[0] = the result in single-precision format VSR[XT].word[1] is undefined VSR[XT].word[2] is undefined VSR[XT].word[3] is undefined FPRF is set to indicate the class and sign of the result
I-7.4.5.XE1.2 The actions that will be taken when the Inexact Exception is enabled for VSX Scalar Floating-Point Arithmetic instructions, VSX Scalar Round to Double-Precision Integer Exact using Current rounding mode (xsrdpic), and VSX Scalar Integer to Floating-Point Format Conversion instructions		
VSX Scalar Floating-Point Arithmetic instructions xsrdpic	XE = 1 XX = 1 *Condition for Inexact Exception occurs*	VSR[XT].doubleword[0]= the result in double-precision format VSR[XT].doubleword[1] is undefined

Instructions tested	Compliance conditions	Expected result
xscvsxddp xscvuxddp xscvsxdsp xscvuxdsp		FPRF is set to indicate the class and sign of the result
I-7.4.5.XE1.3 The actions that will be taken when the Inexact Exception is enabled for VSX Scalar Convert Double-Precision Floating-Point to Integer Word instructions		
xscvdpsxws xscvdpuxws	XE = 1 XX = 1 *Condition for Inexact Exception occurs*	VSR[XT].word[1] = the result VSR[XT].word[0] is undefined VSR[XT].word[2] is undefined VSR[XT].word[3] is undefined FPRF is set to indicate the class and sign of the result
I-7.4.5.XE1.4 The actions that will be taken when the Inexact Exception is enabled for VSX Vector Floating-Point Arithmetic instructions, VSX Vector Floating-Point Reciprocal Estimate instructions, VSX Vector round and Convert Double-Precision to Single-Precision format (xvcvdpsp), VSX Vector Double-Precision to Integer Format Conversion instructions, VSX Vector Single-Precision to Integer Format Conversion instructions, and VSX Vector Integer to Floating-Point Format Conversion instructions		
VSX Vector Floating-Point Arithmetic instructions xvredp xvresp xvcvdpsp xvcvdpsxds xvcvdpsxws xvcvdpuxds xvcvdpuxws xvcvspsxds xvcvspsxws xvcvspuxds xvcvspuxws xvcvsxddp xvcvuxddp xvcvsxdsp xvcvuxdsp xvcvsxwsp xvcvuxwsp	XE = 1 XX = 1 *Condition for Inexact Exception occurs*	Update of VSR[XT] is suppressed for all vector elements FR is not modified FI is not modified FPRF is not modified
I-7.4.5.XE1.5 The actions that will be taken when the Inexact Exception is enabled for *VSX Scalar Quad-Precision Arithmetic instructions, VSX Scalar Quad-Precision Square Root instruction, and VSX Scalar Round Quad-Precision instructions*		
xsaddqp[o] xsdivqp[o] xsmulqp[o] xssubqp[o] xsmaddqp[o] xsmsubqp[o] xsnmaddqp[o] xsnmsubqp[o] xssqrtqp[o] xsrqpi xsrqpxp	XE = 1 XX = 1 *Condition for Inexact Exception occurs*	VSR[VRT+32] = result in quad-precision format FR is set to indicate if the rounded result was incremented FI is set to 1 FPRF is set to indicate class and sign of the result
I-7.4.5.XE1.6 The actions that will be taken when the Inexact Exception is enabled for *VSX Scalar Convert with round Quad-Precision to Double-Precision format instruction*		
xscvqpdp	XE = 1 XX = 1 *Condition for Inexact Exception occurs*	VSR[VRT+32].doubleword[0] = result in double-precision format VSR[VRT+32].doubleword[1] = 0x0000_0000_0000_0000 FR is set to indicate if the rounded result was incremented FI is set to 1 FPRF is set to indicate class and sign of the result
I-7.4.5.XE1.7 The actions that will be taken when the Inexact Exception is enabled for *VSX Scalar truncate and Convert Quad-Precision to Signed Doubleword instruction*		
xscvqpsdz	XE = 1	VSR[XT].doubleword[0] = result in signed integer format

Instructions tested	Compliance conditions	Expected result
	XX = 1 *Condition for Inexact Exception occurs*	VSR[VRT+32].doubleword[1] = 0x0000_0000_0000_0000 FR is set to 0 FI is set to 1 FPRF is undefined
I-7.4.5.XE1.8 The actions that will be taken when the Inexact Exception is enabled for *VSX Scalar truncate and Convert Quad-Precision to Signed Word instruction*		
xscvqpswz	XE = 1 XX = 1 *Condition for Inexact Exception occurs*	VSR[XT].word[1] = result in signed integer format VSR[VRT+32].word[0] = 0x0000_0000 VSR[VRT+32].word[2] = 0x0000_0000 VSR[VRT+32].word[3] = 0x0000_0000 FR is set to 0 FI is set to 1 FPRF is undefined
I-7.4.5.XE1.9 The actions that will be taken when the Inexact Exception is enabled for *VSX Scalar truncate and Convert Quad-Precision to Unsigned Doubleword instruction*		
xscvqpudz	XE = 1 XX = 1 *Condition for Inexact Exception occurs*	VSR[XT].doubleword[0] = result in unsigned integer format VSR[VRT+32].doubleword[1] = 0x0000_0000_0000_0000 FR is set to 0 FI is set to 1 FPRF is undefined
I-7.4.5.XE1.10 The actions that will be taken when the Inexact Exception is enabled for *VSX Scalar truncate and Convert Quad-Precision to Unsigned Word instruction*		
xscvqpuwz	XE = 1 XX = 1 *Condition for Inexact Exception occurs*	VSR[XT].word[1] = result in unsigned integer format VSR[VRT+32].word[0] = 0x0000_0000 VSR[VRT+32].word[2] = 0x0000_0000 VSR[VRT+32].word[3] = 0x0000_0000 FR is set to 0 FI is set to 1 FPRF is undefined
I-7.4.5.XE1.11 The actions that will be taken when the Inexact Exception is enabled for *VSX Scalar Convert with round Double-Precision to Half-Precision format instruction*		
xscvdphp	XE = 1 XX = 1 *Condition for Inexact Exception occurs*	VSR[XT] rightmost halfword of double-word[0] = result in half-precision format VSR[XT] leftmost 3 halfwords of double-word[0] are set to 0 VSR[XT] doubleword[1] is undefined FR is set to indicate if the rounded result was incremented

Instructions tested	Compliance conditions	Expected result
		FI is set to 1 FPRF is set to indicate class and sign of the result
I-7.4.5.XE1.12 The actions that will be taken when the Inexact Exception is enabled for *VSX Vector Convert with round Single-Precision to Half-Precision format instruction*		
xvcvsphp	XE = 1 XX = 1 *Condition for Inexact Exception occurs*	VSR[XT] is not modified FR and FI are not modified FPRF is not modified

8.2.5.3. Actions taken when the exception is disabled

Instructions tested	Compliance conditions	Expected result
I-7.4.5.XE0.1 The actions that will be taken when the Inexact Exception is disabled for VSX Scalar round and Convert Double-Precision to Single-Precision format (xscvdpsp)		
xscvdpsp	XE = 0 XX = 1 *Condition for Inexact Exception occurs*	VSR[XT].word[0] = the result in single-precision format VSR[XT].word[1] is undefined VSR[XT].word[2] is undefined VSR[XT].word[3] is undefined FPRF is set to indicate the class and sign of the result
I-7.4.5.XE0.2 The actions that will be taken when the Inexact Exception is disabled for VSX Scalar Double-Precision Arithmetic instructions, VSX Scalar Single-Precision Arithmetic instructions, VSX Scalar Double-Precision Square Root instruction, VSX Scalar Single-Precision Square Root instruction, VSX Scalar Round to Single-Precision (xsrsp), the VSX Scalar Round to Double-Precision Integer Exact using Current rounding mode (xsrdpic), and VSX Scalar Integer to Double-Precision Format Conversion instructions		
VSX Scalar Double-Precision Arithmetic instructions *VSX Scalar Single-Precision Arithmetic instructions* *VSX Scalar Double-Precision Square Root instruction* *VSX Scalar Double-Precision Square Root instruction* xsrsp xsrdpic xscvsxddp xscvuxddp	XE = 0 XX = 1 *Condition for Inexact Exception occurs*	VSR[XT].doubleword[0]= the result in double-precision format VSR[XT].doubleword[1] is undefined FPRF is set to indicate the class and sign of the result
I-7.4.5.XE0.3 The actions that will be taken when the Inexact Exception is disabled for VSX Scalar Convert Double-Precision Floating-Point to Integer Word instructions		
xscvdpsxws xscvdpuxws	XE = 0 XX = 1 *Condition for Inexact Exception occurs*	VSR[XT].word[1] = the result VSR[XT].word[0] is undefined VSR[XT].word[2] is undefined VSR[XT].word[3] is undefined FPRF is set to indicate the class and sign of the result
I-7.4.5.XE0.4 The actions that will be taken when the Inexact Exception is disabled for *VSX Vector Double-Precision Arithmetic instructions* and *VSX Vector Double-Precision Square Root instruction*		

Instructions tested	Compliance conditions	Expected result
VSX Vector Double-Precision Arithmetic instructions *VSX Vector Double-Precision Square Root instruction*	XE = 0 XX = 1 Vector element i is causing an Inexact exception	VSR[XT].doubleword[i]= the result in double-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.5.XE0.5 The actions that will be taken when the Inexact Exception is disabled for *VSX Vector Single-Precision Arithmetic instructions* and *VSX Vector Single-Precision Square Root instruction*		
VSX Vector Single-Precision Arithmetic instructions *VSX Vector Single-Precision Square Root instruction*	XE = 0 XX = 1 Vector element i is causing an Inexact exception	VSR[XT].word[i]= the result in single-precision format FR is not modified FI is not modified FPRF is not modified
I-7.4.5.XE0.5a The actions that will be taken when the Inexact Exception is disabled for *VSX Scalar Convert with round Double-Precision to Half-Precision format instruction*		
xscvdphp	XE = 0 XX = 1 *Condition for Inexact Exception occurs*	VSR[XT] rightmost halfword of double-word[0] = result in half-precision format VSR[XT] leftmost three halfwords of doubleword[0] = 0x0000_0000_0000 VSR[XT].doubleword[1] is undefined FR is set to indicate if the rounded result was incremented FI is set to indicate the result is inexact FPRF is set to indicate class and sign of the result
I-7.4.5.XE0.6 The actions that will be taken when the Inexact Exception is disabled for *VSX Scalar Quad-Precision Arithmetic instructions, VSX Scalar Quad-Precision Square Root instruction* and *VSX Scalar Round Quad-Precision instructions*		
xsaddqp[o] xsdivqp[o] xsmulqp[o] xssubqp[o] xsmaddqp[o] xsmsubqp[o] xsnmaddqp[o] xsnmsubqp[o] xssqrtqp[o] xsrqpi xsrqpxp	XE = 0 XX = 1 *Condition for Inexact Exception occurs*	VSR[VRT+32] = result in quad-precision format FR is set to indicate if the rounded result was incremented FI is set to 1 FPRF is set to indicate class and sign of the result
I-7.4.5.XE0.7 The actions that will be taken when the Inexact Exception is disabled for *VSX Scalar Convert with round Quad-Precision to Double-Precision format [using round to Odd] instructions*		
xscvqpdp[o]	XE = 0 XX = 1 *Condition for Inexact Exception occurs*	VSR[VRT+32].doubleword[0] = result in double-precision format VSR[VRT+32].doubleword[1] = 0x0000_0000_0000_0000 FR is set to indicate if the rounded result was incremented FI is set to indicate the result is inexact FPRF is set to indicate class and sign of the result
I-7.4.5.XE0.8 The actions that will be taken when the Inexact Exception is disabled for *VSX Scalar truncate and Convert Quad-Precision to Signed Doubleword instruction* and *VSX Scalar truncate and Convert Quad-Precision to Signed Word instruction*		

Instructions tested	Compliance conditions	Expected result
xscvqpsdz xscvqpswz	XE = 0 XX = 1 *Condition for Inexact Exception occurs*	VSR[VRT+32].doubleword[0] = result in signed integer format VSR[VRT+32].doubleword[1] = 0x0000_0000_0000_0000 FR is set to 0 FI is set to 1 FPRF is undefined
I-7.4.5.XE0.9 The actions that will be taken when the Inexact Exception is disabled for *VSX Scalar truncate and Convert Quad-Precision to Unsigned Doubleword instruction* and *VSX Scalar truncate and Convert Quad-Precision to Unsigned Word instruction*		
xscvqpudz xscvqpuwz	XE = 0 XX = 1 *Condition for Inexact Exception occurs*	VSR[VRT+32].doubleword[0] = result in unsigned integer format VSR[VRT+32].doubleword[1] = 0x0000_0000_0000_0000 FR is set to 0 FI is set to 1 FPRF is undefined
I-7.4.5.XE0.10 The actions that will be taken when the Inexact Exception is disabled for *VSX Vector Convert with round Single-Precision to Half-Precision format instruction*		
xvcvsphp	XE = 0 XX = 1 Vector element i is causing an Inexact exception	VSR[XT] rightmost halfword of word[i] = result in half-precision format VSR[XT] leftmost halfword of word[i] is set to 0 FR is not modified FI is not modified FPRF is not modified

8.2.6. Combinations of exceptions

Architecture sections:

- I-7.4 VSX Floating-Point Exceptions

Scenario groups:

- Cases where two exceptions can occur
- Setting exception bit XX when enabled Overflow/Underflow Exception occurs

8.2.6.1. Cases where two exceptions can occur

Instructions tested	Compliance conditions	Expected result
I-7.4.Combine.1 Multiply-Add instructions may set both Invalid Operation Exception (SNaN) and Invalid Operation Exception (∞x0)		
Representative of *VSX multiplication instructions*	*Condition for Invalid Operation Exception* ***SNaN*** occurs *Condition for Invalid Operation Exception* ***Infinity x Zero*** occurs	VXSNAN = 1 VXIMZ = 1

Instructions tested	Compliance conditions	Expected result
I-7.4.Combine.2 Compare Ordered instructions may set both Invalid Operation Exception (SNaN) and Invalid Operation Exception (Invalid Compare)		
Representative of: xscmpodp xvcmpgedp[.] xvcmpgtdp[.] xvcmpgesp[.] xvcmpgtsp[.]	*Condition for Invalid Operation Exception* ***SNaN*** occurs *Condition for Invalid Operation Exception* ***Invalid Compare*** occurs	VXSNAN = 1 VXVC = 1
I-7.4.Combine.3 Convert to Integer instructions may set both Invalid Operation Exception (SNaN) and Invalid Operation Exception (Invalid Integer Convert)		
Representative of *VSX conversion to integer instructions*	*Condition for Invalid Operation Exception* ***SNaN*** occurs *Condition for Invalid Operation Exception* ***Invalid Integer Convert*** occurs	VXSNAN = 1 VXCVI = 1

8.2.6.2. Setting Inexact Exception when enabled Overflow/Underflow Exception occurs

Instructions tested	Compliance conditions	Expected result
I-7.4.CombineInexact.1 Inexact Exception with enabled Overflow Exception, when rounding changes the significand		
Representative of VSX potentially overflowing/ underflowing and inexact instructions	Significands of the rounded and intermediate results differ *Condition for Overflow Exception* occurs OE=1	XX=1
I-7.4.CombineInexact.2 Inexact Exception with enabled Overflow Exception, when rounding does not change the significand		
Representative of VSX potentially overflowing/ underflowing and inexact instructions	The rounded and intermediate results differ The significands of the rounded and intermediate results are equal *Condition for Overflow Exception* occurs OE=1	XX=0
I-7.4.CombineInexact.3 Inexact Exception with enabled Underflow Exception, when rounding changes the significand		
Representative of *VSX* VSX potentially overflowing/ underflowing and inexact instructions	Significands of the rounded and intermediate results differ *Condition for enabled Underflow Exception* occurs UE=1	XX=1
I-7.4.CombineInexact.4 Inexact Exception with enabled Underflow Exception, when rounding does not change the significand		
Representative of VSX potentially overflowing/ underflowing and inexact instructions	The rounded and intermediate results differ The significands of the rounded and intermediate results are equal *Condition for enabled Underflow Exception* occurs UE=1	XX=0
I-7.4.CombineInexact.5 Inexact Exception with disabled Overflow Exception		
Representative of VSX potentially overflowing/ underflowing and inexact instructions	*Condition for Overflow Exception* occurs OE = 0	XX = 1

8.2.7. Setting the exception summary bits

Architecture sections:

- I-7.2.2 Floating-Point Status and Control Register

VSX floating-point instruction: instructions that may cause an exception, all the instructions used in the scenarios of sections 8.2.1 through 8.2.6

Instructions tested	Compliance conditions	Expected result
I-7.2.2.ExSum.1 FX is set to 1 if any exception occurs		
Representative VSX floating-point instruction	Any exception occurs (one or more)	FX=1
I-7.2.2.ExSum.2 FX is sticky		
Representative VSX floating-point instruction	FX = 1 before instruction execution No exception occurs	FX=1
I-7.2.2.ExSum.3 FX stays zero when no exceptions occur		
Representative VSX floating-point instruction	FX = 0 before instruction execution No exception occurs	FX=0
I-7.2.2.ExSum.4 FEX is set to 1 if any enabled exception occurs		
Representative VSX floating-point instruction	Any enabled exception occurs (one or more)	FEX=1
I-7.2.2.ExSum.5 FEX is not sticky		
Representative VSX floating-point instruction	FEX = 1 before instruction execution No enabled exception occurs	FEX=0
I-7.2.2.ExSum.6 FEX stays zero when no enabled exceptions occur		
Representative VSX floating-point instruction	FEX = 0 before instruction execution No enabled exception occurs Any disabled exception occurs (one or more)	FEX=0

8.2.8. Floating-point exception modes

Architecture sections:

- I-7.4 Floating-Point Exceptions

Instructions tested	Compliance conditions	Expected result
I-7.4.ExMode.1 Floating-point exception mode Ignore Exceptions		
Representative VSX floating-point instruction	MSR_{FE0} = 0 MSR_{FE1} = 0 Some enabled floating-point exception occurs	Ignore Exceptions Mode: the system floating-point enabled error handler is not invoked
I-7.4.ExMode.2 Floating-point exception mode Imprecise Nonrecoverable Mode		
Representative VSX floating-point instruction	MSR_{FE0} = 0 MSR_{FE1} = 1	Imprecise Nonrecoverable Mode: The system floating-point enabled exception error handler is invoked at some point at or beyond the instruction that caused

Instructions tested	Compliance conditions	Expected result
	Some enabled floating-point exception occurs	the enabled exception, in nonrecoverable mode
I-7.4.ExMode.3 Floating-point exception mode Imprecise Recoverable Mode		
Representative VSX floating-point instruction	MSR_{FE0} = 1 MSR_{FE1} = 0 Some enabled floating-point exception occurs	Imprecise Recoverable Mode: the system floating-point enabled exception error handler is invoked at some point at or beyond the instruction that caused the enabled exception, in recoverable mode
I-7.4.ExMode.4 Floating-point exception mode Precise Mode		
Representative VSX floating-point instruction	MSR_{FE0} = 1 MSR_{FE1} = 1 Some enabled floating-point exception occurs	Precise Mode: the system floating-point enabled exception error handler is invoked precisely at the instruction that caused the enabled exception.

Part II. Scenarios for Compliance Testing - Virtual Environment (Book II) and Related Supervisor Instructions (Book III)

This Part describes the scenarios required for compliance testing the Virtual Environment (Book II) and related Supervisor Instructions (Book III). The methodology and guidelines specified in Chapter 2, “Power ISA - OpenPOWER Profile Test Harness and Test Suite” [2] apply to all of the scenarios in this Part.

9. Storage Model (Chapter II.1)

Table of Contents

9.1. Single-Copy Atomicity (Section II.1.4)

Below are the scenarios that verify correct implementation of the Single-Copy Atomicity rules as described in the Power ISA Book II Section 1.4. All the scenarios should be implemented in BE/LE addressing modes.

Alignment requirements of storage accesses are defined in Section 1.11.1 of Book I.

Instructions from the following two instruction groups are used. Each scenario should be verified for all storage instructions accessing a quadword operand (lq, lqarx, stq, stqcx.) and a pair of any other instruction from the groups.

Load : Instructions from Sections 3.3.2, 4.6.2 of Book I, lhbrx, lwbrx, ldbrx, lq, lbarx, lharx, lwarx, ldarx, lqarx

Store : Instructions from Sections 3.3.3, 4.6.3 of Book I, sthbrx, stwbrx, stdbrx, stq, stbcx., sthcx., stwcx., stdcx., stqcx.

Instruction sequence		Observability preconditions	Expected results
II-1.4.Atomicity.1 Two processors (P1 and P2) perform atomic stores to non-overlapping memory locations (A, B).			
P1 Store to A	P2 Store to B	A, B are aligned. Initial values of corresponding bytes of the source registers used by Store instructions and of locations A and B should be different.	The contents of A, B are the same as if the two stores were performed by a single processor
II-1.4.Atomicity.2 Two processors (P1 and P2) perform atomic stores to the same storage location (A)			
P1 Store to A	P2 Store to A	A is aligned. Initial values of corresponding bytes of the source registers used by Store instructions and of location A should be different.	The contents of A is the result stored by one of the processors.
II-1.4.Atomicity.3 Two processors (P1 and P2) perform atomic store and atomic load to/from the same storage location (A)			
P1 Store to A	P2 Load from A	A is aligned. Initial values of corresponding bytes of the source register used by Store instruction and of location A should be different.	The value returned by Load is the contents of A before Store or after it.

9.2. Cache Model and Cache Management Instructions (Section II.1.5)

9.3. Storage Control Attributes (Section II.1.6)

Below are the scenarios that verify correct implementation of the Storage Control Attributes as described in the Power ISA Book II Section 1.6.

Alignment requirements of storage accesses are defined in Section 1.11.1 of Book I.

Instructions from the following two instruction groups are used. Each scenario should be verified for a pair of any instruction from the groups.

Load : Instructions from Sections 3.3.2, 4.6.2 of Book I, lhbrx, lwbrx, ldbrx, lq, lbarx, lharx, lwarx, ldarx, lqarx

Store : Instructions from Sections 3.3.3, 4.6.3 of Book I, sthbrx, stwbrx, stdbrx, stq, stbcx., sthcx., stwcx., stdcx., stqcx.

Instruction sequence		Observability preconditions	Expected results
II-1.6.Control.1 Two processors (P1 and P2) perform atomic stores and atomic loads to/from the same coherent storage location (A)			
P1 Store to A x 10	P2 Load from A x 10	A is aligned. Storage Attributes of the A memory location are set as follows: • not Write Through Required • not Caching Inhibited • not Guarded • Memory Coherence Required Source registers used by all Store instructions should be initialized to different values	P2 can never load a newer value first and then, later, load an older value stored by P1. Values are stored by P1 to A in Program Order.
II-1.6.Control.2 Two processors (P1 and P2) perform atomic stores and atomic loads to/from the same cache inhibited storage location (A)			
P1 Store to A x 10	P2 Load from A x 10	A is aligned. Storage Attributes of the A memory location are set as follows: • Caching Inhibited Source registers used by all Store instructions should be initialized to different values	P2 can never load a newer value first and then, later, load an older value stored by P1. Values are stored by P1 to A in Program Order.

9.4. Shared Storage (Section II.1.7)

9.4.1. Storage Access Ordering (Section II.1.7.1)

Below are the scenarios that verify correct implementation of the Storage Access Ordering rules as described in the Power ISA Book II Section 1.7.1.

Instructions from the following two instruction groups are used. Each scenario should be verified for a pair of any instruction from the groups.

Load : Instructions from Sections 3.3.2, 4.6.2 of Book I, lhbrx, lwbrx, ldbrx, lq, lbarx, lharx, lwarx, ldarx, lqarx

Store : Instructions from Sections 3.3.3, 4.6.3 of Book I, sthbrx, stwbrx, stdbrx, stq, stbcx., sthcx., stwcx., stdcx., stqcx.

The **Synchronize** instruction group includes the sync and eieio instructions.

Instruction sequence			Observability preconditions	Expected results
II-1.7.1.Access.1 Two processors (P1 and P2) perform stores and loads to/from two non-overlapping Caching Inhibited and Guarded storage locations (A, B)				
P1 Store to A Store to B	P2 Load from B Load from A		Storage Attributes of the A and B memory locations are set as follows: • Caching Inhibited • Guarded Initial values of corresponding bytes of the source registers used by Stores and of locations A and B should be different.	If P2 loads from B a new value stored by P1, then P2 must load also a new value from A
II-1.7.1.Access.2 Two processors (P1 and P2) perform stores and loads to/from two non-overlapping storage locations (A, B) and there is data dependency between the Loads				
P1 Store to A Synchronize Store to B	P2 Load from B Load from A		Storage Attributes of the A and B memory locations are set as follows: • not Caching Inhibited • not Guarded Initial values of corresponding bytes of the source registers used by Stores and of locations A and B should be different. Target register of the first Load is used as the base register of the second Load executed by P2	If P2 loads from B a new value stored by P1, then P2 must load also a new value from A
II-1.7.1.Access.3 Three processors (P1, P2 and P3) perform stores and loads to/from two non-overlapping storage locations (A, B)				
P1 Store to A	P2 Load from A Syncronize Store to B	P3 Load from B Syncronize Load from A	Storage Attributes of the A and B memory locations are set as follows: • Memory Coherence Required Initial values of corresponding bytes of the source registers used by Stores and of locations A and B should be different.	If P2 loads from A a new value stored by P1 and P3 loads from B a new value stored by P2, then P3 must load also a new value from A
II-1.7.1.Access.4 Three processors (P1, P2 and P3) perform stores and loads to/from three non-overlapping storage locations (A, B, C)				
P1 Store to A Synchronize Store to B	P2 Load from B isync Store to C	P3 Load from C Synchronize Load from A	Storage Attributes of the A, B and C memory locations are set as follows: -- Memory Coherence Required Initial values of corresponding bytes of the source registers used by Stores and of locations A, B and C should be different.	If P2 loads from B a new value stored by P1 and P3 loads from C a new value stored by P2, then P3 must load also a new value from A

9.4.2. Copy/Paste-Initiated Data Transfers (Section II.1.7.2)

Below are the scenarios that verify correct implementation of the Copy/Paste-Initated Data Transfers as described in the Power ISA Book II Section 1.7.2.

Alignment requirements of storage accesses are defined in Section 1.11.1 of Book I.

Copy/Paste Instructions (from Section 4.4 of Book II): copy, paste., cpabort

Instruction sequence	Observability preconditions	Expected results
II-1.7.2.CP.1 Copy and paste. operations with correct alignment		
P1 cpabort copy paste.	Copy source EA and paste. target EA are both 128 byte aligned Storage Attributes of the copy and paste. memory location are set as follows: • not Write Through Required • not Caching Inhibited • not Guarded • Memory Coherence Required	First three bits of CR0=001 Target location contents = source location contents
II-1.7.2.CP.2 Copy and paste. operations with copy source EA has incorrect alignment		
P1 cpabort copy paste.	Copy source EA is not 128 byte aligned Paste. target EA is 128 byte aligned Storage Attributes of the copy and paste. memory location are set as follows: • not Write Through Required • not Caching Inhibited • not Guarded • Memory Coherence Required	First three bits of CR0=000 Target location is not changed System alignment error
II-1.7.2.CP.3 Copy and paste. operations with paste. target EA has incorrect alignment		
P1 cpabort copy paste.	Copy source EA is 128 byte aligned Paste. target EA is not 128 byte aligned Storage Attributes of the copy and paste. memory location are set as follows: • not Write Through Required • not Caching Inhibited • not Guarded • Memory Coherence Required	First three bits of CR0=000 Target location is not changed System alignment error
II-1.7.2.CP.4 Copy and paste. operations with copy source EA block is Caching Inhibited		
P1 cpabort copy paste.	Copy source EA and paste. target EA are both 128 byte aligned Storage Attributes of the copy source memory location is set as follows: • Caching Inhibited	First three bits of CR0=000 Target location is not changed System data storage error
II-1.7.2.CP.5 Copy and paste. operations with paste. target EA block is Caching Inhibited		
P1 cpabort copy	Copy source EA and paste. target EA are both 128 byte aligned Storage Attributes of the paste. target memory location is set as follows:	First three bits of CR0=000 Target location is not changed System data storage error

Instruction sequence	Observability preconditions	Expected results
paste.	• Caching Inhibited	

9.4.3. Atomic Update and Reservations (lwarx and stwcx) (Subsection II.1.7.4)

Below are the scenarios that verify correct implementation of the Atomic Update and Reservations rules as described in the Power ISA Book II Section 1.7.4.

Alignment requirements of storage accesses are defined in Section 1.11.1 of Book I.

Instructions from the following two instruction groups are used. Each scenario should be verified for a pair of any two instruction accessing the same operand size (byte, halfword, word, doubleword, and quadword) from the groups.

Load And Reserve : lbarx, lharx, lwarx, ldarx, lqarx

Store Conditional : stbcx., sthcx., stwcx., stdcx., stqcx.

Also the following instruction groups are used in scenarios:

Load : Instructions from Sections 3.3.2, 4.6.2 of Book I.

Store : Instructions from Sections 3.3.3, 4.6.3 of Book I, dcbz

Instruction sequence		Observability preconditions	Expected results
II-1.7.4.Update.1 Two processors (P1 and P2) perform Load And Reserve and Store Conditional from/to storage location A			
P1 Load And Reserve from A Store Conditional to A	P2 Load And Reserve from A Store Conditional to A	A is aligned. Storage Attributes of the storage location A are set as follows: • Memory Coherence Required Initial values of corresponding bytes of the source registers used by Stores and of location A should be different.	If both Store Conditional succeed, then either P1 loads the value stored by P2 or vice versa.
II-1.7.4.Update.2 Processor P1 performs Load And Reserve and Store Conditional from/to non-overlapping storage locations A and B belonging to the same reservation granule			
P1 Load And Reserve from A Store Conditional to B isync Store Conditional to A		A and B are aligned. A and B belong to the same reservation granule. Storage Attributes of the storage locations A and B are set as follows: • Memory Coherence Required Initial values of corresponding bytes of the source registers used by Stores and of locations A and B should be different.	Store Conditional to A fails and the value of A is not updated
II-1.7.4.Update.3 Processors P1 and P2 perform Load and Reserve and Store from/to non-overlapping storage locations A and B belonging to the same reservation granule			
P1 Load And Reserve from A	P2 Store to B	A and B are aligned. A and B belong to the same reservation granule.	If part of Loads from memory location B executed by P1 return initial value of B and part of them return the new value stored by

Instruction sequence	Observability preconditions	Expected results
isync Load from B x 5 Store Conditional to A	Storage Attributes of the A and B memory locations are set as follows: • Memory Coherence Required Initial values of corresponding bytes of the source registers used by Store Conditional, Store and of locations A and B should be different.	P2, then Store Conditional executed by P1 fails and the value of A is not updated
II-1.7.4.Update.4 Processors P1 performs Load and Reserve from Cache inhibited storage location A		
P1 Load And Reserve from A	A is aligned. Storage Attributes of the A memory locations are set as follows: • Cache Inhibited	The system data storage error handler is invoked
II-1.7.4.Update.5 Processors P1 performs Load and Reserve from Write Through storage location A		
P1 Load And Reserve from A	A is aligned. Storage Attributes of the A memory locations are set as follows: • Write Through Required	The system data storage error handler is invoked

9.5. Instruction Storage (Section II.1.9)

9.5.1. Instruction and data storage synchronization

Below are the scenarios that verify correct implementation of instruction and data storage synchronization as described in the Power ISA Book II Section 1.9.

Instruction sequence	Compliance conditions	Expected results
II-1.9.Sync.1 Single-process instruction and data storage synchronization		
P1 executes the instruction sequence specified in Case 1 of the programming note in Section 1.9	The conditions listed in Case 1 of the programming note in Section 1.9	Instruction storage is consistent with data storage
II-1.9.Sync.2 Cross-process instruction and data storage synchronization		
P1 and P2 execute the instruction sequences specified in Case 2 of the programming note in Section 1.9	The conditions listed in Case 2 of the programming note in Section 1.9	Instruction storage is consistent with data storage

10. Management of Shared Resources (Chapter II.3)

Table of Contents

10.1. Program Priority Registers (Section II.3.1)

The following scenarios verify setting of the Program Priority Register (PPR) as described in the Power ISA Book II Section 3.1.

Instruction sequence	Observability preconditions	Expected results
II-3.1.PPR.1 Processors performs the *or Rx, Rx, Rx* instruction		
P1 or R31, R31, R31	PRI field of the PPR register is initialized to any value different from 001	PRI field of the PPR register is set to 001
II-3.1.PPR.2 Processors performs the *or Rx, Rx, Rx* instruction		
P1 or R1, R1, R1	PRI field of the PPR register is initialized to any value different from 010	PRI field of the PPR register is set to 010
II-3.1.PPR.3 Processors performs the *or Rx, Rx, Rx* instruction		
P1 or R6, R6, R6	PRI field of the PPR register is initialized to any value different from 011	PRI field of the PPR register is set to 011
II-3.1.PPR.4 Processors performs the *or Rx, Rx, Rx* instruction		
P1 or R2, R2, R2	PRI field of the PPR register is initialized to any value different from 100	PRI field of the PPR register is set to 100
II-3.1.PPR.5 Processors performs the *or. Rx, Rx, Rx* instruction		
P1 or. R31, R31, R31	PRI field of the PPR register is initialized to any value different from 001	PRI field of the PPR register is not changed

11. Storage Control Instructions (Chapter II.4)

Table of Contents

- Instruction-driven scenarios: sections II.4.3, II.4.6, III.5.9

11.1. Cache Management Instructions (Section II.4.3)

The following scenarios verify correct implementation of the Cache Management Instructions as described in the Power ISA Book II Section 4.3.

Instruction sequence	Observability preconditions	Expected results
II-4.3.Cache.1 Processors P1 performs icbi instruction to Memory Coherent storage locations A		
P1 Icbi A	Address translation is ON Storage Attributes of the A memory location are set as follows: • Memory Coherence Required Block containing A is in the instruction cache of P1 and any other processor of the system	Reference and change recording of the page accessed by the instruction are not updated The effective address of A is translated using translation resources that are used for data accesses
II-4.3.Cache.2 Processors P1 performs icbi instruction to not Memory Coherent storage locations A		
P1 Icbi A	Address translation is ON Storage Attributes of the A memory location are set as follows: • not Memory Coherence Required Block containing A is in the instruction cache of P1 and any other processor of the system	Reference and change recording of the page accessed by the instruction are not updated The effective address of A is translated using translation resources that are used for data accesses
II-4.3.Cache.3 Processors P1 performs icbt instruction to memory location A with valid translation path		
P1 Icbt A	Address translation is ON There is valid translation path for the effective address of A	Reference and change recording of the page accessed by the instruction are not updated The instruction effective address is translated using translation resources that are used for data accesses
II-4.3.Cache.4 Processors P1 performs icbt instruction to memory location A with invalid translation path		
P1 Icbt A	Address translation is ON	The instruction effective address is translated using translation resources that are used for data accesses

Instruction sequence	Observability preconditions	Expected results
	There is no valid translation path for the effective address of A	System data storage error handler is not invoked
II-4.3.Cache.5 Processors P1 performs dcba instruction to memory location A		
P1 dcba A		No architectural resources are changed
II-4.3.Cache.6 Processors P1 performs dcbt instruction to memory location A		
P1 dcbt A		No architectural resources are changed
II-4.3.Cache.7 Processors P1 performs dcbtst instruction to memory location A		
P1 dcbtst A		No architectural resources are changed
II-4.3.Cache.8 Processors P1 performs dcbz instruction to memory location A		
P1 dcbz A		All bytes in the data cache block containing the memory location A are set to zero. Data cache block size is a design dependent value that should be configured for each specific processor design.
II-4.3.Cache.9 Processors P1 performs dcbst instruction to memory location A with valid translation path		
P1 dcbst A	Address translation is ON There is valid translation path for the effective address of A	Reference and change recording of the page accessed by the instruction are not updated
II-4.3.Cache.10 Processors P1 performs dcbf instruction to memory location A with valid translation path		
P1 dcbf A	Address translation is ON There is valid translation path for the effective address of A	Reference and change recording of the page accessed by the instruction are not updated
II-4.3.Cache.11 Processors P1 performs dst instruction to memory location A		
P1 dst A		The operation must be treated as no-op
II-4.3.Cache.12 Processors P1 performs dstst instruction to memory location A		
P1 dstst A		The operation must be treated as no-op
II-4.3.Cache.13 Processors P1 performs dss instruction to memory location A		
P1 dss A		The operation must be treated as no-op

11.2. Atomic Memory Operations (Section II.4.5)

Load Atomic Instructions : lwat, ldat

Store Atomic Instructions : stwat, stdat

Load Atomic Function Codes : 00000 Fetch and Add, 00001 Fetch and XOR, 00010 Fetch and OR, 00011 Fetch and AND, 00100 Fetch and Maximum Unsigned, 00101 Fetch and Maximum

Signed, 00110 Fetch and Minimum Unsigned, 00111 Fetch and Minimum Signed, 01000 Swap, 10000 Compare and Swap Not Equal, 11000 Fetch and Increment Bounded, 11001 Fetch and Increment Equal, 11100 Fetch and Decrement Bounded

Note

Load Function Codes not listed are considered invalid

Store Atomic Function Codes : 00000 Store Add, 00001 Store XOR, 00010 Store OR, 00011 Store AND, 00100 Store Maximum Unsigned, 00101 Store Maximum Signed, 00110 Store Minimum Unsigned, 00111 Store Minimum Signed, 11000 Store Twin

Note

Store Function Codes not listed are considered invalid

Instruction sequence	Observability preconditions	Expected results
II-4.5.AMO.1 Correct computation of all of the valid function codes for Atomic Memory Operations with correct alignment		
Load Atomic Instructions Store Atomic Instructions	Memory accessed is contained within an aligned 32-byte block of storage Valid Function Code	Correct result of operation
II-4.5.AMO.2 System data storage error when using invalid function codes for Atomic Memory Operations		
Load Atomic Instructions Store Atomic Instructions	Memory accessed is contained within an aligned 32-byte block of storage Invalid Function Code	Target location is not changed System data storage error
II-4.5.AMO.3 Alignment storage error when portion of memory accessed by the instruction is not contained within an aligned 32-byte block of storage for Atomic Memory Operations		
Load Atomic Instructions Store Atomic Instructions	Memory accessed is not contained within an aligned 32-byte block of storage	Target location is not changed System alignment error

11.3. Synchronization and Memory Barrier Instructions (Section II.4.6)

The following scenarios are used to verify correct implementation of the Synchronization and Memory Barrier Instructions as described in the Power ISA Book II Section 4.6.

Load and Reserve and Store Conditional instruction scenarios are covered by Section 4 of this document.

Synchronize and Enforce In-order Execution instruction (sync and eieio) scenarios are covered by Section 3 of this document.

12. Transactional Memory Facility (Chapter II.5)

Below are the scenarios that verify correct implementation of the Transactional Memory Facility as described in the Power ISA Book II Chapter 5.

The following instruction groups are used in scenarios:

Load : Instructions from Sections 3.3.2, 4.6.2 of Book I.

Store : Instructions from Sections 3.3.3, 4.6.3 of Book I, dcbz

Transaction Abort : tabort., tabortdc., tabortdci., tabortwc., tabortwci.

Non-checkpointed SPRs: EBBHR, EBBRR, BESCR, Performance Monitor registers

Disallowed Instructions (in transactional state): icbi, copy, paste., cpabort, lwat, ldat, stwat, stdat, dcbf, dcbi, dcbst, rfscv, rfid, hrfid, rfebb, mtmsr, mtmsrd, msgsnd, msgsndp, msgslr, msgslrp, slbie, slbieg, slbia, slbmte, slbfee, stop, tlbie, tlbiel, mtspr to a non-checkpointed SPR

Cache Block Invalidation : dcbf, dcbi, icbi

In all scenarios tbegin. instructions are immediately followed by a beq instruction that branches to the main body of the transaction failure handler.

By default, memory attributed of all memory accesses in the following scenarios are set to the following values:

- Not Write Through
- Not Cache Inhibited
- Memory Coherent

Transaction Conflict Granule size is a design dependent value that should be configured for each specific processor design.

Instruction sequence		Observability preconditions	Expected results
II-5.Mem.1 Processors P1 and P2 perform Loads and Stores from/to memory location A in transactional mode			
P1 tbegin. 0 beq Store to A x 10 Load from A tend. 0	P2 tbegin. 0 beq Load from A tend. 0	Initial state of P1 and P2 is Non-transactional. Initial values of the source registers used by Stores are all different. Initial values of the source registers used by Stores and of location A are different.	If both transactions succeed then the Load of P1 loads from A the value stored by its last Store instruction, while P2 loads from A either its initial value or the value stored by the last Store of P1. If only the transaction of P1 succeeds then it loads from A the value stored by its last Store. If only the transaction of P2 succeeds then it loads from A its initial value.
II-5.Mem.2 Processors P1 and P2 perform Loads and Stores from/to not overlapping memory locations A and B belonging to the same Transaction Conflict Granule in transactional mode.			
P1 tbegin. 0 beq	P2 tbegin. 0 beq	Initial state of P1 and P2 is Non-transactional. Initial values of the source registers used by Stores are all different.	If both transactions succeed then the Load of P1 loads from A the value stored by its last Store instruction, while P2 loads from B the value stored by its Store instruction.

Instruction sequence		Observability preconditions	Expected results
Store to A x 10 Load from A tend. 0	Store to B Load from B tend. 0	Initial values of the source registers used by Stores and of locations A and B are different.	If only the transaction of P1 succeeds then it loads from A the value stored by its last Store. The value of B is not changed. If only the transaction of P2 succeeds then it loads from B the value stored by its Store instruction. The value of A is not changed.
II-5.Mem.3 Processors P1 and P2 perform Loads and Stores from/to not overlapping memory locations A and B belonging to the same Transaction Conflict Granule. P1 performs memory accesses in transactional mode.			
P1 tbegin. 0 beq Store to A x 10 Load from A tend. 0	P2 Store to B Load from B	Initial state of P1 and P2 is Non-transactional. Initial values of the source registers used by Stores are all different. Initial values of the source registers used by Stores and of locations A and B are different.	If transaction of P1 succeeds then it loads from A the value stored by its last Store. P2 loads from B the value stored by its Store instruction. If transaction of P1 fails, the value of A is not changed. P2 loads from B the value stored by its Store instruction.
II-5.Mem.4 Processors P1 and P2 perform Loads and Stores from/to not overlapping memory locations A and B belonging to the same Transaction Conflict Granule. P1 performs memory accesses in Rollback-only Transactional mode (ROT).			
P1 tbegin. 1 beq Store to A x 10 Load from A tend. 0	P2 Store to B Load from B	Initial state of P1 and P2 is Non-transactional. Initial values of the source registers used by Stores are all different. Initial values of the source registers used by Stores and of locations A and B are different.	If transaction of P1 succeeds then it loads from A the value stored by its last Store. P2 loads from B the value stored by its Store instruction. If transaction of P1 fails, the value of A is not changed. P2 loads from B the value stored by its Store instruction.
II-5.Mem.5 Processors P1 and P2 perform Loads and Stores from/to not overlapping memory locations A and B belonging to the same Transaction Conflict Granule. P1 performs memory accesses in Rollback-only Transactional mode (ROT).			
P1 tbegin. 1 beq Load from A tend. 0	P2 Store to B Load from B	Initial state of P1 and P2 is Non-transactional. Initial values of the source registers used by Store and of location B are different.	Transaction of P1 always succeeds. It loads from A its initial value. P2 loads from B the value stored by its Store instruction.
II-5.Mem.6 Processor P1 performs Load and Stores from/to memory location A in transactional mode			
P1 tbegin. 0 beq Store to A x 10 Load from A Transaction Abort tend. 0		Initial state of P1 is Non-transactional. Initial values of the source registers used by Store and of location A are different. If conditional version of Transaction Abort instruction is used its condition should be met and the instruction is successfully executed.	Transaction of P1 always fails. Value of A is not changed.
II-5.Mem.7 Processor P1 performs Load and Stores from/to memory location A in Rollback-only transactional mode			
P1 tbegin. 1 beq		Initial state of P1 is Non-transactional. Initial values of the source registers used by Store and of location A are different.	Transaction of P1 always fails. Value of A is not changed.

Instruction sequence	Observability preconditions	Expected results
Store to A x 10 Load from A Transaction Abort tend. 0	If conditional version of Transaction Abort instruction is used its condition should be met and the instruction is successfully executed.	
II-5.Mem.8 Processor P1 performs Load and Stores from/to memory location A in transactional mode. Processor P2 performs tlbie that invalidates translation path of A.		
P1 P2 tbegin. 0 tlbie of A beq Store to A x 10 Load from A tend. 0	Initial state of P1 is Non-transactional. Initial values of the source registers used by Store and of location A are different.	If transaction of P1 succeeds then it loads from A the value stored by its last Store, otherwise the value of A is not changed.
II-5.Mem.9 Processor P1 performs Load and Stores from/to memory location A in Rollback-only transactional mode. Processor P2 performs tlbie that invalidates translation path of A.		
P1 P2 tbegin. 1 tlbie of A beq Store to A x 10 Load from A tend. 0	Initial state of P1 is Non-transactional. Initial values of the source registers used by Store and of location A are different.	If transaction of P1 succeeds then it loads from A the value stored by its last Store, otherwise the value of A is not changed.
II-5.Mem.10 Processor P1 performs Load and Stores from/to memory location A in transactional mode		
P1 tbegin. 0 beq Store to A x 10 Load from A treclaim. tend. 0	Initial state of P1 is Non-transactional. Initial values of the source registers used by Store and of location A are different.	Transaction of P1 always fails. Value of A is not changed.
II-5.Mem.11 Processor P1 performs Load and Stores from/to memory location A in Rollback-only transactional mode		
P1 tbegin. 1 beq Store to A x 10 Load from A treclaim. tend. 0	Initial state of P1 is Non-transactional. Initial values of the source registers used by Store and of location A are different.	Transaction of P1 always fails. Value of A is not changed.
II-5.Mem.12 Processor P1 performs Loads and Stores from/to memory location A in transactional mode. The number of nested transactions exceeds its maximum value supported by the design.		
P1	Initial state of P1 is Non-transactional.	Transaction of P1 always fails. Value of A is not changed.

Instruction sequence	Observability preconditions	Expected results
tbegin. 0 beq Store to A x 10 Load from A tbegin. 0 beq Store to A x 10 Load from A tbegin. 0 beq Store to A x 10 Load from A : : tbegin. 0 beq Store to A x 10 Load from A tend. 0	Initial values of the source registers used by Store and of location A are different.	
II-5.Mem.13 Processor P1 performs Loads and Stores from/to memory location A in Rollback-only transactional mode. The number of nested transactions exceeds its maximum value supported by the design.		
P1 tbegin. 1 beq Store to A x 10 Load from A tbegin. 1 beq Store to A x 10 Load from A tbegin. 0 beq Store to A x 10 Load from A : : tbegin. 1	Initial state of P1 is Non-transactional. Initial values of the source registers used by Store and of location A are different.	Transaction of P1 always fails. Value of A is not changed.

Instruction sequence	Observability preconditions	Expected results
beq Store to A x 10 Load from A tend. 0		
II-5.Mem.14 Processor P1 performs Loads and Stores from/to memory locations A, B and C in transactional mode.		
P1 tbegin. 0 beq Store to A x 10 Load from A tbegin. 0 beq Store to B x 10 Load from B tend. 0 tbegin. 0 beq Store to C x 10 Load from C tend. 1	Initial state of P1 is Non-transactional. Initial values of the source registers used by Stores and of locations A, B and C are different.	P1 loads from A, B and C the values stored by the Store instruction immediately preceding the corresponding Load.
II-5.Mem.15 Processor P1 performs Loads and Stores from/to memory locations A, B and C in transactional mode.		
P1 tbegin. 0 beq Store to A x 10 Load from A tbegin. 0 beq Store to B x 10 Load from B tend. 1 tbegin. 0 beq Store to C x 10 Load from C tend. 1	Initial state of P1 is Non-transactional. Initial values of the source registers used by Stores and of locations A, B and C are different.	P1 loads from A and B the values stored by the Store instruction immediately preceding the corresponding Load. Value of memory location C is unchanged.

Instruction sequence	Observability preconditions	Expected results
II-5.Mem.16 Processor P1 performs Loads and Stores from/to memory locations in transactional mode. The transaction fails due to footprint overflow.		
P1 tbegin. 0 beq Store to A Store to B Store to C : : : Store to D Load from A tend. 0	Initial state of P1 is Non-transactional. Initial values of the source registers used by Stores and of accessed memory locations are different. The number of memory accesses in transactional mode should exceed the footprint size of the current design and cause the transaction failure due to footprint overflow.	Values of all accessed memory locations is unchanged.
II-5.Mem.17 Processor P1 performs Load and Stores from/to memory location A in transactional mode		
P1 tbegin. 0 beq Store to A Load from A Disallowed Instruc- tion tend. 0	Initial state of P1 is Non-transactional. Initial values of the source register used by Store and of location A are different.	Transaction of P1 fails. Value of A is not changed.
II-5.Mem.18 Processor P1 performs Load and Stores from/to memory location A in transactional mode and in suspended mode		
P1 tbegin. 0 beq Store to A Load from A tsr. 0 Store to A tsr. 1 tend. 0	Initial state of P1 is Non-transactional. Initial values of the source registers used by Stores and of location A are different.	Transaction of P1 fails. Value of A is not changed.
II-5.Mem.19 Processor P1 performs Load and Stores from/to memory locations A and B in transactional mode and in suspended mode		
P1 tbegin. 0 beq Store to A	Initial state of P1 is Non-transactional. Initial values of the source registers used by Stores and of locations A and B are different.	Transaction of P1 succeeds. P1 loads from A and B new values stored by the preceding Stores.

Instruction sequence	Observability preconditions	Expected results
Load from A tsr. 0 Store to B Load from B tsr. 1 tend. 0	A and B memory location belongs to different Transaction Conflict Granule	
II-5.Mem.20 Processor P1 performs Load and Stores from/to memory location A in transactional mode. Translation path of A is invalidated while P1 is in suspended mode.		
P1 tbegin. 0 beq Store to A Load from A tsr. 0 tlbie tsr. 1 tend. 0	Initial state of P1 is Non-transactional. Initial values of the source register used by Store and of location A are different. tlbie invalidates translation path of A.	Transaction of P1 fails. Value of A is not changed.
II-5.Mem.21 Processor P1 performs Load and Stores from/to Cache Inhibited memory location A in transactional mode.		
P1 tbegin. 0 beq Store to A Load from A tend. 0	Initial state of P1 is Non-transactional. Initial values of the source register used by Store and of location A are different. Memory attributes of A are set to the following values: • Cache Inhibited	Transaction of P1 fails. Value of A is not changed.
II-5.Mem.22 Processor P1 performs Load and Stores from/to Write Through memory location A in transactional mode.		
P1 tbegin. 0 beq Store to A Load from A tend. 0	Initial state of P1 is Non-transactional. Initial values of the source register used by Store and of location A are different. Memory attributes of A are set to the following values • Write Through	Transaction of P1 fails. Value of A is not changed.
II-5.Mem.23 Processor P1 performs Load and Stores from/to Not Coherent memory location A in transactional mode.		
P1 tbegin. 0 beq Store to A Load from A tend. 0	Initial state of P1 is Non-transactional. Initial values of the source register used by Store and of location A are different. Memory attributes of A are set to the following values • Not Memory Coherent	Transaction of P1 fails. Value of A is not changed.

Instruction sequence	Observability preconditions	Expected results
II-5.Mem.24 Processor P1 performs Load and Stores from/to Cache Inhibited memory location A in suspended mode.		
P1 tbegin. 0 beq Store to B Load from B tsr. 0 Store to A Load from A tsr. 1 tend. 0	Initial state of P1 is Non-transactional. Initial values of the source register used by Stores and of locations A and B are different. Memory attributes of A are set to the following values: • Cache Inhibited Memory attributes of B are set to the following values: • Not Write Through • Not Cache Inhibited • Memory Coherent	Transaction of P1 succeeds. Values of A and B are updated by the corresponding Stores.
II-5.Mem.25 Processor P1 performs Load and Stores from/to Write Through memory location A in suspended mode.		
P1 tbegin. 0 beq Store to B Load from B tsr. 0 Store to A Load from A tsr. 1 tend. 0	Initial state of P1 is Non-transactional. Initial values of the source register used by Store and of location A are different. Memory attributes of A are set to the following values • Write Through Memory attributes of B are set to the following values: • Not Write Through • Not Cache Inhibited • Memory Coherent	Transaction of P1 succeeds. Values of A and B are updated by the corresponding Stores.
II-5.Mem.26 Processor P1 performs Load and Stores from/to Not Coherent memory location A in suspended mode.		
P1 tbegin. 0 beq Store to B Load from B tsr. 0 Store to A Load from A tsr. 1 tend. 0	Initial state of P1 is Non-transactional. Initial values of the source register used by Store and of location A are different. Memory attributes of A are set to the following values • Not Memory Coherent Memory attributes of B are set to the following values: • Not Write Through • Not Cache Inhibited • Memory Coherent	Transaction of P1 succeeds. Values of A and B are updated by the corresponding Stores.
II-5.Mem.27 Processor P1 performs Load from memory location A in transactional mode and a cache block invalidation instruction in suspended mode.		
P1 tbegin. 0 beq	Initial state of P1 is Non-transactional. Cache Block Invalidation instruction accesses the cache block containing A	Transaction of P1 fails.

Instruction sequence		Observability preconditions	Expected results
Load from A tsr. 0 Cache Block Invali-dation tsr. 1 tend. 0			
II-5.Mem.28 Processor P1 performs Load from memory location A in transactional mode. P2 performs a cache block invalidation instruction.			
P1 tbegin. 0 beq Load from A tend. 0	P2 Cache Block Invali-dation	Initial state of P1 is Non-transactional. Cache Block Invalidation instruction accesses the cache block containing A	Transaction of P1 fails.
II-5.Mem.29 Processor P1 performs Load from memory location A in transactional mode and dcbst instruction in suspended mode.			
P1 tbegin. 0 beq Load from A tsr. 0 dcbst tsr. 1 tend. 0		Initial state of P1 is Non-transactional. dcbst instruction accesses the cache block containing A.	Transaction of P1 succeeds.
II-5.Mem.30 Processor P1 performs Load from memory location A in transactional mode. P2 performs a cache block invalidation instruction.			
P1 tbegin. 0 beq Load from A tend. 0	P2 dcbst	Initial state of P1 is Non-transactional. dcbst instruction accesses the cache block containing A.	Transaction of P1 succeeds.
II-5.Mem.31 Processor P1 performs Store to memory location A in transactional mode and dcbst instruction in suspended mode.			
P1 tbegin. 0 beq Store to A tsr. 0 dcbst tsr. 1 tend. 0		Initial state of P1 is Non-transactional. Initial values of the source register used by Store and of location A are different. dcbst instruction accesses the cache block containing A.	Transaction of P1 fails. Value of A is unchanged.

Instruction sequence		Observability preconditions	Expected results
II-5.Mem.32 Processor P1 performs Store to memory location A in transactional mode. P2 performs a cache block invalidation instruction.			
P1 tbegin. 0 beq Store to A tend. 0	P2 dcbst	Initial state of P1 is Non-transactional. Initial values of the source register used by Store and of location A are different. dcbst instruction accesses the cache block containing A.	Transaction of P1 fails. Value of A is unchanged.
II-5.Mem.33 Processor P1 performs Load and Reserve from memory location A in transactional mode and Store Conditional in non-transactional mode.			
P1 tbegin. 0 beq lwarx from A tend. 0 stwcx. to A		Initial state of P1 is Non-transactional. Initial values of the source register used by Store Conditional and of location A are different.	Transaction of P1 succeeds. Store Conditional fails. Value of A is unchanged.
II-5.Mem.34 Processor P1 performs Load and Reserve from memory location A in non-transactional mode and Store Conditional in transactional mode.			
P1 lwarx from A tbegin. 0 beq stwcx. to A tend. 0		Initial state of P1 is Non-transactional. Initial values of the source register used by Store Conditional and of location A are different.	Transaction of P1 succeeds. Store Conditional fails. Value of A is unchanged.
II-5.Mem.35 Processor P1 performs Load and Reserve from memory location A in transactional mode and Store Conditional in suspended mode.			
P1 tbegin. 0 beq lwarx from A tsr. 0 stwcx. to A tsr. 1 tend. 0		Initial state of P1 is Non-transactional. Initial values of the source register used by Store Conditional and of location A are different.	Transaction of P1 succeeds. Store Conditional fails. Value of A is unchanged.
II-5.Mem.36 Processors P1 and P2 perform Loads and Stores from/to memory location A. P1 performs the accesses in transactional mode.			
P1 tbegin. 0 beq Store to A x 10 Load from A	P2 Load from A	Initial state of P1 and P2 is Non-transactional. Initial values of the source registers used by Stores are all different. Initial values of the source registers used by Stores and of location A are different.	If the transaction of P1 succeeds then the Load of P1 loads from A the value stored by its last Store instruction, while P2 loads from A either its initial value or the value stored by the last Store of P1. If the transaction of P1 fails then P2 loads from A its initial value.

Instruction sequence	Observability preconditions	Expected results
tend. 0		
II-5.Mem.37 Processor P1 performs Loads and Stores from/to memory locations A, B and C in transactional mode. One of the nested transactions is aborted.		
P1 tbegin. 0 beq Store to A x 10 Load from A tbegin. 0 beq Store to B x 10 Load from B tbegin. 0 beq Store to C x 10 Load from C Transaction Abort tend. 0 tend. 0 tend. 0	Initial state of P1 is Non-transactional. Initial values of the source registers used by Stores are all different. Initial values of the source registers used by Stores and of locations A, B and C are different.	All the transactions fail. The values of A, B and C are unchanged.
II-5.Mem.38 Processor P1 performs Load and Stores from/to memory location A in transactional mode and one of the Disallowed Instructions in suspended state.		
P1 tbegin. 0 beq Store to A Load from A tsr. 0 Disallowed Instruc-tion tsr. 1 tend. 0	Initial state of P1 is Non-transactional. Initial values of the source register used by Store and of location A are different.	Transaction of P1 succeeds. Value of A is updated by its Store instruction.
II-5.Mem.39 Processor P1 performs Load from memory location A in Rollback-only transactional mode. Processor P2 performs tlbie that invalidates translation path of A.		
P1 P2 tbegin. 1 tlbie of A beq Load from A tend. 0	Initial state of P1 is Non-transactional.	Transaction of P1 succeeds.

Instruction sequence		Observability preconditions	Expected results
II-5.Mem.40 Processor P1 performs tend. instruction in suspended mode			
P1 tbegin. 0 beq tsr. 0 tend. 0 tsr. 1 tend. 0		Initial state of P1 is Non-transactional.	TM Bad Thing type Program Interrupt occurs.
II-5.Mem.41 Processor P1 performs tend. instruction in non-transactional mode			
P1 tend. 0	Initial state of P1 is Non-transactional.	The tend. instruction is treated as a no-op.	

13. Time Base (Chapter II.6)

The following scenarios verify correct implementation of the Time Base facility described in Power ISA Book II Chapter 6.

Instruction sequence		Observability preconditions	Expected results
II-6.TimeBase.1	Verify the instruction sequences: mfspr Rx,268 and the 32-bit equivalent produce a monotonically increasing 64-bit Time Base value the test loop:		

<pre>load Rx with a loop count
mtctr Rx
mfspr Rx,268
 	
loop:
 	
mfspr Ry,268
cmpd cr0,Rx,Ry
bgt cr0,error
mr Rx,Ry
bdnz loop</pre>
		Within a single thread, rapidly read the Time Base in a loop.	For any two sequential 64-bit Time Base values; N and N+1
			N <= N+1
			i.e. the next time base value should be the greater than or equal to the previous value.

II-6.TimeBase.2	Verify that the Time Base clocks at a consistent and constant rate across cores and sockets, And that privileged code (Hypervisor or Operation system) can initialize the Time base to an consistent initial value per PowerISA 2.07B Book III section 7.2.1.		
With multiple threads of execution running on multiple cores and or sockets, Allocate atomically the next sequential element of the shared array, then read the time base and store the value in that shared array element.		Initialize a shared spin-lock to the locked state and initialize the shared array to zeros and the shared index to the first element of the array. Start the multiple test threads where each threads waits on the shared spin-lock until the controlling thread starts the test and releases the spin-lock. The multiple threads terminate when the shared index exceeds the size of the shared array.	After the shared array is sequentially filled with time base values from multiple threads of execution, Verify that for each sequence pair of time base values N and N+1, N <= N+1 i.e. the next time base value should be the greater than or equal to the previous value.

Note

See Chapter 20, “Timer Facilities (Chapter III.7)” [308] for tests associated with the Time Base facilities.

14. Event-Based Branch Facility (Chapter II.7)

Below are the scenarios that verify correct implementation of the Event-Based Branch Facility as described in the Power ISA Book II Chapter 7.

Instruction sequence	Observability preconditions	Expected results
II-7.Branch.1	Processor P1 performs rfebb	
P1 rfebb S		BESCR[GE] register field gets the value of the S operand. MSR[TS] register field gets the value of the BESCR[TS] bit.
II-7.Branch.2	Use mtspr to set the EBB Return Address, execute the Return from EBB instruction and verify the return from EBB	
• Obtain access to the EBB and PMU facilities. • Intialize the EBBHR to the address of event_handler: • Program the PMU to count cycle event for 10000, and set MMCR0 bit 43: Performance Monitor Event-Based Branch • Enable (EBE) • Update (mtspr) the BESCRU to set Global Enable (GE) and Performance Monitor Event-Based Exception Enable (PME) to '1' • Loop while (ebbrr_sa == 0); • After the loop exits examine the ebbrr_sa and ebbcr_sa	• BESCR bit 30 (EE) = 1 • BESCR is 0..0, GE is disabled, PME is disabled, TS and PMEO are 0. • A static dword EBBRR save area (ebbrr_sa) is initialized to 0 (NULL) • A static dword EBBCR save area (ebbcr_sa) is initialized to 0 • An event_handler code sequence exist that will; – obtain addressability to the ebbrr_sa and ebbcr_sa, – retrieve the EBBRR via mfspr and store the value into the ebbrr_sa dword, – retrieve the EBBCR via mfspr and store the value into the ebbcr_sa dword, – return control via rfebb 0	On loop exit: • the value in ebbrr_sa is the address of a instruction within the range of the while loop. • The value in ebbcr_sa will have: – Bit 0 (GE) will be 0 – Bit 30 (EE) will be 0 – Bits 32:33 (TS) will be 00 – Bit 62 (EEO) will be 1 – Bit 63 (PMEO) will be 1

Note

In privileged mode, the ebbrr, ebbcr and pmu facilities are available.

In user mode, issue a system call to get access to the facilities. For more details see the IBM Power Architecture Facilities Library project on github https://github.com/paflib/paflib and the associated Event Based Branching Overview, ABI, and API.

See Section 19.1.1, “FSCR Facility Enable (FE)” [247] and Section 19.1.2, “HFSCR Facility Enable (FE)” [248] for additional tests associated with the Event-Based Branch Facility (EBB).

Part III. Scenarios for Compliance Testing - Operating Environment (Book III)

This section describes the scenarios required for compliance testing the Operating Environment (Book III). The methodology and guidelines specified in Chapter 2, “Power ISA - OpenPOWER Profile Test Harness and Test Suite” [2] apply to all of the scenarios in this Part.

15. Logical Partitioning (LPAR) and Thread Control (Chapter III.2)

Table of Contents

15.1. Processor Compatibility Register (PCR)

Architecture sections:

- – III.2.5 Processor Compatibility Register (PCR)

Scenario groups:

- Availability of Transactional Memory instructions
- Availability of Transactional Memory SPRs

15.1.1. Availability of Transactional Memory instructions

Guideline: t he scenarios in this group are instruction-driven scenarios. Each scenario should be tested for every instruction in the list of **Instructions tested**, unless otherwise stated in the scenario

Instructions tested	Compliance conditions	Expected result
III.2.6.Instr.7 Transactional Memory instructions are treated as illegal instructions when $PCR_{V2.07}$ and $PCR_{V2.06}$ are set to 1 in problem state		
Transactional Memory instructions	$PCR_{V2.07}$ = 1 $PCR_{V2.06}$ = 1 MSR_{PR} = 1	Treated as an illegal instruction
III.2.6.Instr.9 Transactional Memory instructions are available when $PCR_{V2.07}$ and $PCR_{V2.06}$ are set to 0 in problem state		
Representative of Transactional Memory instructions	$PCR_{V2.07}$ = 0 $PCR_{V2.06}$ = 0 MSR_{PR} = 1	Correct instruction behavior
III.2.6.Instr.12 Transactional Memory instructions are available when $PCR_{V2.07}$ and $PCR_{V2.06}$ are set to 1 not in problem state		
Representative of Transactional Memory instructions	$PCR_{V2.07}$ = 1 $PCR_{V2.06}$ = 1 MSR_{PR} = 0	Correct instruction behavior

15.1.2. Availability of Transactional Memory SPRs

Guideline: *Access to an SPR* means execution of any instruction that accesses an SPR.

Instruction sequence	Compliance conditions	Expected result
III.2.6.SPR.3 Transactional Memory facility SPRs are treated as if they were not defined for the implementation, when $PCR_{V2.07}$ and $PCR_{V2.06}$ are set to 1 in problem state		
Access to an SPR	The designated SPR is a Transactional Memory facility SPR $PCR_{V2.07}$ = 1 $PCR_{V2.06}$ = 1 MSR_{PR} = 1	Instruction behaves as if the designated SPR is not defined for the implementation
III.2.6.SPR.6 Transactional Memory facility SPRs are available when $PCR_{V2.07}$ and $PCR_{V2.06}$ are set to 0 in problem state		
Access to an SPR	The designated SPR is a Transactional Memory facility SPR $PCR_{V2.07}$ = 0 $PCR_{V2.06}$ = 0 MSR_{PR} = 1	Correct instruction behavior
III.2.6.SPR.9 Transactional Memory facility SPRs are available when $PCR_{V2.07}$ and $PCR_{V2.06}$ are set to 1 not in problem state		
Access to an SPR	The designated SPR is a Transactional Memory facility SPR $PCR_{V2.07}$ = 1 $PCR_{V2.06}$ = 1 MSR_{PR} = 0	Correct instruction behavior

16. Branch Facility (Chapter III.3)

Table of Contents

16.1. Machine State Register (MSR)

Architecture sections:

- III.3.2.1 Machine State Register

Scenario groups:

- Available categories

16.1.1. Available categories

Guideline: *Access to an SPR* means execution of any instruction that accesses an SPR.

Instruction sequence	Compliance conditions	Expected result
III.3.2.1.Category.1 When MSR_{TM} =0 the thread cannot execute any Transactional Memory instructions		
Representative of Transactional Memory instructions	MSR_{TM} = 0	Treated as an illegal instruction
III.3.2.1.Category.2 When MSR_{TM} =0 the thread cannot access any Transactional Memory registers		
Access to an SPR	The designated SPR is a Transactional Memory facility SPR MSR_{TM} = 0	Instruction behaves as if the designated SPR is not defined for the implementation
III.3.2.1.Category.3 When MSR_{TM} =1 the thread can execute Transactional Memory instructions		
Representative of Transactional Memory instructions	MSR_{TM} = 1 No other register has made the Transactional Memory facility unavailable	Correct instruction behavior
III.3.2.1.Category.4 When MSR_{TM} =1 the thread can access Transactional Memory registers		
Access to an SPR	The designated SPR is a Transactional Memory facility SPR MSR_{TM} = 1 No other register has made the Transactional Memory facility unavailable	Correct instruction behavior
III.3.2.1.Category.5 When MSR_{TM} =1 and $PCR_{V2.07}$ = $PCR_{V2.06}$ =1, the instructions and facilities in the Transactional Memory category are unavailable in problem state		
Representative of Transactional Memory instructions	MSR_{TM} = 1 $PCR_{V2.07}$ = 1 $PCR_{V2.06}$ = 1	Treated as an illegal instruction

Instruction sequence	Compliance conditions	Expected result
	MSR_{PR} = 1	
III.3.2.1.Category.6 When MSR_{TM} =0 and $PCR_{V2.07}$ = $PCR_{V2.06}$ =0, the thread cannot execute any Transactional Memory instructions		
Representative of Transactional Memory instructions	MSR_{TM} = 0 $PCR_{V2.07}$ = 0 $PCR_{V2.06}$ = 0	Treated as an illegal instruction
III.3.2.1.Category.7 When MSR_{VEC} =0 the thread cannot execute any vector instructions		
Representative of Vector instructions	MSR_{VEC} =0	Treated as an illegal instruction
III.3.2.1.Category.8 When MSR_{VEC} =1 the thread can execute vector instructions unless they have been made unavailable by some other register		
Representative of Vector instructions	MSR_{VEC} =1	Correct instruction behavior
III.3.2.1.Category.9 When MSR_{VSX} =0 the thread cannot execute any VSX instructions		
Representative of VSX instructions	MSR_{VSX} =0	Treated as an illegal instruction
III.3.2.1.Category.10 When MSR_{VSX} =1 the thread can execute VSX instructions		
Representative of VSX instructions	MSR_{VSX} =1	Correct instruction behavior
III.3.2.1.Category.11 When MSR_{FP} =0 the thread cannot execute any floating-point instructions		
Representative of Floating-Point instructions	MSR_{FP} =0	Treated as an illegal instruction
III.3.2.1.Category.12 When MSR_{FP} =1 the thread can execute floating-point instructions		
Representative of Floating-Point instructions	MSR_{FP} =1	Correct instruction behavior

16.2. Transaction state transitions

Architecture sections:

- III.3.2.2 State Transitions Associated with the Transactional Memory Facility

Scenario groups:

- Transaction state transitions requested by instructions

16.2.1. Transaction state transitions requested by instructions

Instructions that update MSR_{TS} and MSR_{TM} : rfebb rfid rfscv hrfid mtmsrd

Input MSR_{TS}, Input MSR_{TM}: the MSR_{TS} and MSR_{TM} values supplied by

- BESCR for rfebb (just the TS value)
- SRR1 for rfid
- HSRR1 for hrfid
- register RS for mtmsrd

Note: when the **Compliance conditions** specify *Input* MSR_{TM} = x this means that the value of MSR_{TM} can be either 0 or 1. In this case, the value of MSR_{TM} in the **Expected result** is also denoted x, and must be the same as the x value in the **Compliance conditions**.

Instruction sequence	Compliance conditions	Expected result
III.3.2.2.Trans.1 In Non-Transactional state with Transactional Memory disabled, enabling/disabling Transactional Memory		

Instruction sequence	Compliance conditions	Expected result
Representative of *Instructions that update MSR_{TS} and MSR_{TM}*	MSR_{TS} MSR_{TM} = 000 before instruction execution *Input MSR_{TS}* = 00 *Input MSR_{TM}* = x	MSR_{TS} = 00 MSR_{TM} = x
III.3.2.2.Trans.2 In Non-transactional state with Transactional Memory disabled, attempting to transition to a different state		
Representative of *Instructions that update MSR_{TS} and MSR_{TM}*	MSR_{TS} MSR_{TM} = 000 before instruction execution *Input MSR_{TS}* != 00	MSR_{TS} MSR_{TM} = 000 TM Bad Thing type Program interrupt is generated
III.3.2.2.Trans.3 In Suspended state with Transactional Memory disabled, attempting to transition to Non-transactional state using instruction other than rfebb		
Representative of: rfid rfscv hrfid mtmsrd	MSR_{TS} MSR_{TM} = 010 before instruction execution *Input MSR_{TS}* = 00 *Input MSR_{TM}* = 0	MSR_{TS} MSR_{TM} = 010
III.3.2.2.Trans.4 In Suspended state with Transactional Memory disabled, attempting to transition to Non-transactional state using rfebb		
rfebb	MSR_{TS} MSR_{TM} = 010 before instruction execution *Input MSR_{TS}* = 00 *Input MSR_{TM}* = 0	MSR_{TS} MSR_{TM} = 010 TM Bad Thing type Program interrupt is generated
III.3.2.2.Trans.5 In Suspended state with Transactional Memory disabled, enabling Transactional Memory while transitioning to Transactional state		
Representative of *Instructions that update MSR $_{TS}$ and MSR_{TM}*	MSR_{TS} MSR_{TM} = 010 before instruction execution *Input MSR_{TS}* = 10 *Input MSR_{TM}* = 1	MSR_{TS} MSR_{TM} = 101
III.3.2.2.Trans.6 In Suspended state with Transactional Memory disabled, enabling/disabling Transactional Memory		
Representative of *Instructions that update MSR MSR_{TS} and MSR_{TM}*	MSR_{TS} MSR_{TM} = 010 before instruction execution *Input MSR_{TS}* = 01 *Input MSR_{TM}* = x	MSR_{TS} = 01 MSR_{TM} = x
III.3.2.2.Trans.7 In Suspended state with Transactional Memory disabled, attempting an illegal transition		
Representative of *Instructions that update MSR_{TS} and MSR_{TM}*	MSR_{TS} MSR_{TM} = 010 before instruction execution *Input MSR_{TS} MSR_{TM}* not one of 000, 101, 01x	MSR_{TS} MSR_{TM} = 010 TM Bad Thing type Program interrupt is generated
III.3.2.2.Trans.8 In Non-transactional state with Transactional Memory enabled, enabling/disabling Transactional Memory		
Representative of *Instructions that update MSR_{TS} and MSR_{TM}*	MSR_{TS} MSR_{TM} = 001 before instruction execution *Input MSR_{TS}* = 00 *Input MSR_{TM}* = x	MSR_{TS} = 00 MSR_{TM} = x
III.3.2.2.Trans.9 In Non-transactional state with Transactional Memory enabled, attempting to transition to a different state		
Representative of *Instructions that update MSR_{TS} and MSR_{TM}*	MSR_{TS} MSR_{TM} = 001 before instruction execution *Input MSR_{TS}* != 00	MSR_{TS} = 00 MSR_{TM} = 0

Instruction sequence	Compliance conditions	Expected result
		TM Bad Thing type Program interrupt is generated
III.3.2.2.Trans.10 In Transactional state with Transactional Memory enabled, any attempt to enable/disable Transactional Memory or transition to a different state		
Representative of *Instructions that update MSR_{TS} and MSR_{TM}*	MSR_{TS} MSR_{TM} = 101 before instruction execution	MSR_{TS} = 00 MSR_{TM} = 1
III.3.2.2.Trans.11 In Suspended state with Transactional Memory enabled, transitioning to Transactional state		
Representative of *Instructions that update MSR_{TS} and MSR_{TM}*	MSR_{TS} MSR_{TM} = 011 before instruction execution *Input* MSR_{TS} = 10 *Input* MSR_{TM} = 1	MSR_{TS} = 10 MSR_{TM} = 1
III.3.2.2.Trans.12 In Suspended state with Transactional Memory enabled, enabling/disabling Transactional Memory		
Representative of *Instructions that update MSR_{TS} and MSR_{TM}*	MSR_{TS} MSR_{TM} = 011 before instruction execution *Input* MSR_{TS} = 01 *Input* MSR_{TM} = x	MSR_{TS} = 01 MSR_{TM} = x
III.3.2.2.Trans.13 In Suspended state with Transactional Memory enabled, attempting an illegal transition		
Representative of *Instructions that update MSR_{TS} and MSR_{TM}*	MSR_{TS} MSR_{TM} = 011 before instruction execution *Input* MSR_{TS} MSR_{TM} not one of 101, 01x	MSR_{TS} = 01 MSR_{TM} = 0 TM Bad Thing type Program interrupt is generated
III.3.2.2.Trans.14 Attempt to set MSR_{TS} to 0b11 (reserved value)		
Representative of *Instructions that update MSR_{TS} and MSR_{TM}*	*Input* MSR_{TS} = 11	TM Bad Thing type Program interrupt is generated

16.3. System linkage instructions

Architecture sections:

- III.3.3.1 System Linkage Instructions

Scenario groups:

- Return from interrupt

16.3.1. Return from interrupt

Instructions tested	Compliance conditions	Expected result
III.3.3.1.Return.1 Correct computation of the next instruction for return from interrupt, in 64-bit mode		
rfid	The new MSR value does not cause any exceptions SF=1 in the new MSR value	NIA = $SRR0_{0:61}$ || 0b00
III.3.3.1.Return.2 Correct computation of the next instruction for return from interrupt, in 32-bit mode		
rfid	The new MSR value does not cause any exceptions SF=0 in the new MSR value	NIA = $^{32}0$ || $SRR0_{32:61}$ || 0b00

Instructions tested	Compliance conditions	Expected result
III.3.3.1.Return.3 Correct computation of the next instruction for hypervisor return from interrupt, in 64-bit mode		
hrfid	The new MSR value does not cause any exceptions SF=1 in the new MSR value	NIA = $HSRR0_{0:61}$ || 0b00
III.3.3.1.Return.4 Correct computation of the next instruction for hypervisor return from interrupt, in 32-bit mode		
hrfid	The new MSR value does not cause any exceptions SF=0 in the new MSR value	NIA = $^{32}0$ || $HSRR0_{32:61}$ || 0b00
III.3.3.1.Return.5 Correct computation of the next instruction for return from system call vectored, in 64-bit mode		
rfscv	The new MSR value does not cause any exceptions SF=1 in the new MSR value	NIA = $LR_{0:61}$ || 0b00
III.3.3.1.Return.6 Correct computation of the next instruction for return from system call vectored, in 32-bit mode		
rfscv	The new MSR value does not cause any exceptions SF=0 in the new MSR value	NIA = $^{32}0$ || $LR_{32:61}$ || 0b00

16.4. Power-Saving Mode Instruction

Architecture sections:

- III.3.3.2.2 Entering and Exiting Power-Saving Mode

Scenario groups:

- Entering power-saving mode
- Exiting power-saving mode

Power-saving mode instruction: stop

Instruction sequence for entering power-saving mode: the instruction sequence following the guidelines specified in Section III.3.3.2.2 of the architecture specification, executed with the *power-saving mode instruction*.

16.4.1. Entering power-saving mode

Instruction sequence	Compliance conditions	Expected result
III.3.3.2.2.Enter.1 Entering power-saving mode		
Instruction sequence for entering power-saving mode		Correct instruction behavior

16.4.2. Exiting power-saving mode

Exceptions that cause exit from power-saving mode without Machine Check: System Reset, Decrementer, External, Hypervisor Maintenance

Instruction sequence	Compliance conditions	Expected result
III.3.3.2.2.Exit.1 Exiting power-saving mode with exception other than Machine Check		

Instruction sequence	Compliance conditions	Expected result
Any instruction that may cause the relevant exceptions	The thread is in power-saving mode before executing the instruction sequence One of the *exceptions that cause exit from power-saving mode without Machine Check* occurs	A System Reset interrupt is generated The contents of SRR1 indicate the type of exception that caused exit from power-saving mode
III.3.3.2.2.Exit.2 Exiting power-saving mode with a Machine Check exception		
Any instruction that may cause a Machine Check exception	The thread is in power-saving mode before executing the instruction sequence A Machine Check exception occurs	A Machine Check interrupt is generated

17. Fixed-Point Facility (Chapter III.4)

Table of Contents

17.1. Fixed-Point Facility Registers

Architecture sections:

- III.4.3.7 Program Priority Register
- III.4.3.8 Problem State Priority Boost Register

Scenario groups:

- Program Priority Register (PPR) and Problem State Priority Boost Register (PSPB)

17.1.1. Program Priority Register (PPR) and Problem State Priority Boost Register (PSPB)

Instruction sequence	Compliance conditions	Expected result
III.4.3.7.Prio.1 Problem state programs may set PPR_{PRI} to values in the range of 0b001 to 0b100		
Any instruction that attempts to set the value of PPR_{PRI}	The program is in problem state The designated value is in the range of 0b001 to 0b100	PPR_{PRI} = the designated value
III.4.3.7.Prio.2 For all priorities except 0b101, if a problem state program attempts to set a value that is not allowed for its privilege level, the PRI field remains unchanged		
Any instruction that attempts to set the value of PPR_{PRI}	The program is in problem state The designated value is greater than 0b101 The PSPB register does not allow the value 0b101 for problem state programs	PPR_{PRI} is unchanged
III.4.3.7.Prio.3 If a problem state program attempts to set its priority value to 0b101 when this priority value is not allowed for problem state programs, the priority is set to 0b100.		
Any instruction that attempts to set the value of PPR_{PRI}	The program is in problem state The designated value is 0b101 The PSPB register does not allow the value 0b101 for problem state programs	PPR_{PRI} = 0b100
III.4.3.7.Prio.4 Privileged programs may also set values in the range of 0b101 to 0b110		
Any instruction that attempts to set the value of PPR_{PRI}	The program is in privileged state The designated value is in the range of 0b101 to 0b110	PPR_{PRI} = the designated value

Instruction sequence	Compliance conditions	Expected result
III.4.3.7.Prio.5 If a privileged program attempts to set a value that is not allowed for its privilege level, the PRI field remains unchanged		
Any instruction that attempts to set the value of PPR_{PRI}	The program is in problem state The designated value is 0b111	PPR_{PRI} is unchanged
III.4.3.7.Prio.6 Hypervisor software may also set 0b111		
Any instruction that attempts to set the value of PPR_{PRI}	The program is in hypervisor state The designated value is 0b111	PPR_{PRI} = the designated value
III.4.3.7.Prio.7 A problem state program is able to set the program priority to medium high only when the PSPB of the thread contains a non-zero value		
Any instruction that attempts to set the value of PPR_{PRI}	The program is in problem state The designated value is 0b101 PSPB contains a non-zero value	PPR_{PRI} = the designated value

17.2. Fixed-Point Load and Store Caching Inhibited Instructions

Architecture sections:

- III.4.4.1 Fixed-Point Load and Store Caching Inhibited Instructions

Scenario groups:

- Conditions for instruction execution
- Correct instruction execution

17.2.1. Conditions for instruction execution

Fixed-Point Load and Store Caching Inhibited Instructions: lbzcix lhzcix lwzcix ldcix stbcix sthcix stwcix stdcix

Instruction sequence	Compliance conditions	Expected result
III.4.4.1 Cond.1 Fixed-Point Load and Store Caching Inhibited instructions must be executed only when MSR_{DR} = 0		
Representative of *Fixed-Point Load and Store Caching Inhibited Instructions*	MSR_{DR} != 0 The storage location specified by the instruction is not in storage specified by the Hypervisor Real Mode Storage Control facility to be treated as non-Guarded	A Data Storage interrupt is generated
III.4.4.1 Cond.2 For Fixed-Point Load and Store Caching Inhibited instructions, the storage location specified by the instructions must not be in storage specified by the Hypervisor Real Mode Storage Control facility to be treated as non-Guarded		
Representative of *Fixed-Point Load and Store Caching Inhibited Instructions*	MSR_{DR} = 0 The storage location specified by the instruction is in storage specified by the Hypervisor Real Mode Storage Control facility to be treated as non-Guarded	A Data Storage interrupt is generated
III.4.4.1 Cond.3 The storage access caused by these instructions is performed as though the specified storage location is Caching Inhibited and Guarded.		

Instruction sequence	Compliance conditions	Expected result
Representative of *Fixed-Point Load and Store Caching Inhibited Instructions*		No copy of the accessed locations is placed into the caches only the storage location specified by the instruction is accessed
III.4.4.1 Cond.4 These instructions are hypervisor privileged		
Representative of *Fixed-Point Load and Store Caching Inhibited Instructions*	MSR_{HV} = 0	A Privileged Instruction type Program interrupt is generated

17.2.2. Correct instruction execution

Fixed-Point Load and Store Caching Inhibited Instructions: lbzcix lhzcix lwzcix ldcix stbcix sthcix stwcix stdcix

Guidelines:

- Each scenario in this group should be tested for every instruction in the **Instructions tested** list
- Each scenario in this group should be tested in both Big-Endian and Little-Endian mode

Instructions tested	Compliance conditions	Expected result
III.4.4.1 Correct.1 Fixed-Point Load and Store Caching Inhibited instructions behave correctly		
Fixed-Point Load and Store Caching Inhibited Instructions		Correct instruction behavior

17.3. OR Instruction

Architecture sections:

- III.4.4.2 OR Instruction

Scenario groups:

- Setting priority levels

17.3.1. Setting priority levels

Instruction sequence	Compliance conditions	Expected result
III.4.4.2.OR.1 The or Rx,Rx,Rx instruction can be used to set PPR_{PRI} to medium high priority		
or R5, R5, R5	The program is in privileged state, or in problem state with PSPB allowing priority value 0b101	PPR_{PRI} = 101
III.4.4.2.OR.2 The or Rx,Rx,Rx instruction can be used to set PPR_{PRI} to high priority		
or R3, R3, R3	The program is in privileged state	PPR_{PRI} = 110
III.4.4.2.OR.3 The or Rx,Rx,Rx instruction can be used to set PPR_{PRI} to very high priority		
or R7, R7, R7	The program is in hypervisor state	PPR_{PRI} = 111

17.4. Transactional Memory Instructions

Architecture sections:

- III. 4.4.3 Transactional Memory Instructions

Scenario groups:

- Transactional Memory privileged instructions

17.4.1. Transactional Memory privileged instructions

Instruction sequence	Compliance conditions	Expected result
III. 4.4.3.Priv.1 If an attempt is made to execute treclaim. in Non-transactional state, a TM Bad Thing type Program interrupt will be generated		
treclaim.	MSR_{TS} = 00	a TM Bad Thing type Program interrupt is generated
III. 4.4.3.Priv.2 The treclaim. and trechkpt. instructions are privileged		
Representative of: treclaim. trechkpt.	The program is in problem state	A Privileged Instruction type Program interrupt is generated
III. 4.4.3.Priv.3 If an attempt is made to execute trechkpt. in Transactional or Suspended, a TM Bad Thing type Program interrupt will be generated		
trechkpt.	MSR_{TS} = 01, or MSR_{TS} = 10	a TM Bad Thing type Program interrupt is generated
III. 4.4.3.Priv.4 If an attempt is made to execute trechkpt. when $TEXASR_{FS}$ = 0, a TM Bad Thing type Program interrupt will be generated		
trechkpt.	$TEXASR_{FS}$ = 0	a TM Bad Thing type Program interrupt is generated

17.5. Move To/From System Register Instructions

Architecture sections:

- III.4.4.4 Move To/From System Register Instructions
- III.6.2.12 Hypervisor Facility Status and Control Register

Scenario groups:

- Correct behavior of mtspr and mfspr
- Special cases of mtspr
- Special cases of mfspr
- Move to/from Performance Monitor SPRs
- mtmsr, mtmsrd, mfmsr

The following definitions refer to **Figure 18. SPR encodings** in section **III. 4.4.4 Move To/From System Register Instructions**.

SPRs that are hypervisor-privileged for mtspr: SPRs marked hypv in the Privileged-mtspr column

SPRs that are not defined for mtspr: SPRs marked "-" in the Privileged-mtspr column

SPRs that are hypervisor-privileged for mfspr: SPRs marked hypv in the Privileged-mfspr column

SPRs that are not defined for mfspr: SPRs marked "-" in the Privileged-mfspr column

17.5.1. Correct behavior of mtspr and mfspr

Guidelines:

Each scenario in this group should be tested for each SPR listed in **Figure 18. SPR encodings**. SPR_{test} denotes the tested SPR

Each scenario in this group, for each SPR_{test} , should be tested in all privilege states: problem, privileged non-hypervisor, and hypervisor. $State_{test}$ denotes the privilege state

Instruction sequence	Compliance conditions	Expected result
III.4.4.4.Move.1 The designated SPR is accessed correctly by mtspr in the specified privilege state		
mtspr	The designated SPR is SPR_{test} The privilege state is $State_{test}$	Correct behavior for mtspr applied to SPR_{test} in state $State_{test}$, according to **Figure 18. SPR encodings**
III.4.4.4.Move.2 The designated SPR is accessed correctly by mfspr in the specified privilege state		
mfspr	The designated SPR is SPR_{test} The privilege state is $State_{test}$	Correct behavior for mfspr applied to SPR_{test} in state $State_{test}$, according to **Figure 18. SPR encodings**

17.5.2. Special cases of mtspr

Instruction sequence	Compliance conditions	Expected result
III.4.4.4.mtspr.1 For mtspr, SPRs TBL and TBU are treated as separate 32-bit registers; setting one leaves the other unaltered		
mtspr	The designated SPR is TBL or TBU	The remaining 32 bits of Time Base are unchanged
III.4.4.4.mtspr.2 Execution of mtspr specifying a privileged SPR causes a Privileged Instruction type Program interrupt when MSR_{PR} = 1		
mtspr	spr_0 = 1 MSR_{PR} = 1	A Privileged Instruction type Program interrupt is generated
III.4.4.4.mtspr.3 Execution of mtspr specifying a hypervisor-privileged SPR causes a Privileged Instruction type Program interrupt when $MSR_{HV\ PR}$ = 0b00 and $LPCR_{EVIRT}$ = 0.		
mtspr	The designated SPR is in *SPRs that are hypervisor-privileged for mtspr* $MSR_{HV\ PR}$ = 0b00 $LPCR_{EVIRT}$ = 0	A Privileged Instruction type Program interrupt is generated
III.4.4.4.mtspr.3b Execution of mtspr specifying a hypervisor-privileged SPR causes a Hypervisor Emulation Assistance interrupt when $MSR_{HV\ PR}$ = 0b00 and $LPCR_{EVIRT}$ = 1.		
mtspr	The designated SPR is in *SPRs that are hypervisor-privileged for mtspr* $MSR_{HV\ PR}$ = 0b00 $LPCR_{EVIRT}$ = 1	A Hypervisor Emulation Assistance interrupt is generated
III.4.4.4.mtspr.4 Execution of mtspr specifying an SPR number that is not defined for the implementation, if spr_0 = 0 and MSR_{PR} = 1		
mtspr	The designated SPR is in *SPRs that are not defined for mtspr* spr_0 = 0 MSR_{PR} = 1	A Hypervisor Emulation Assistance interrupt is generated
III.4.4.4.mtspr.5 Execution of mtspr specifying SPR 0,4,5, or 6 that is not defined for the implementation, if spr_0 = 0 and MSR_{PR} = 0		
mtspr	The designated SPR is SPR 0,4,5, or 6, and is in *SPRs that are not defined for mtspr* spr_0 = 0 MSR_{PR} = 0	A Hypervisor Emulation Assistance interrupt is generated

Instruction sequence	Compliance conditions	Expected result
III.4.4.4.mtspr.6 Execution of mtspr specifying an SPR number (not SPR 0,4,5, or 6) that is not defined for the implementation, if spr_0 = 0, MSR_{PR} = 0, and $LPCR_{EVIRT}$ = 0		
mtspr	The designated SPR is not SPR 0,4,5, or 6, and is in *SPRs that are not defined for mtspr* spr_0 = 0 MSR_{PR} = 0 $LPCR_{EVIRT}$ = 0	No operation
III.4.4.4.mtspr.6b Execution of mtspr specifying an SPR number (not SPR 0,4,5, or 6) that is not defined for the implementation, if spr_0 = 0, MSR_{PR} = 0, and $LPCR_{EVIRT}$ = 1		
mtspr	The designated SPR is not SPR 0,4,5, or 6, and is in *SPRs that are not defined for mtspr* spr_0 = 0 MSR_{PR} = 0 $LPCR_{EVIRT}$ = 1	A Hypervisor Emulation Assistance interrupt is generated
III.4.4.4.mtspr.7 Execution of mtspr specifying an SPR number that is not defined for the implementation, if spr_0 = 1 and MSR_{PR} = 1		
mtspr	The designated SPR is in *SPRs that are not defined for mtspr* spr_0 = 1 MSR_{PR} = 1	A Privileged Instruction type Program interrupt is generated
III.4.4.4.mtspr.8 Execution of mtspr specifying an SPR number that is not defined for the implementation, if spr_0 = 1, MSR_{PR} = 0, and $LPCR_{EVIRT}$ = 0		
mtspr	The designated SPR is in *SPRs that are not defined for mtspr* spr_0 = 1 MSR_{PR} = 0 $LPCR_{EVIRT}$ = 0	No operation
III.4.4.4.mtspr.8b Execution of mtspr specifying an SPR number that is not defined for the implementation, if spr_0 = 1, MSR_{PR} = 0, and $LPCR_{EVIRT}$ = 1		
mtspr	The designated SPR is in *SPRs that are not defined for mtspr* spr_0 = 1 MSR_{PR} = 0 $LPCR_{EVIRT}$ = 1	A Hypervisor Emulation Assistance interrupt is generated
III.4.4.4.mtspr.9 If an attempt is made to execute mtspr specifying a TM SPR in other than Non-transactional state, a TM Bad Thing type Program interrupt is generated		
mtspr	The designated SPR is a TM SPR other than TFHAR MSR_{TS} != 10	A TM Bad Thing type Program interrupt is generated
III.4.4.4.mtspr.10 If an attempt is made to execute mtspr specifying TFHAR in suspended state, a TM Bad Thing type Program interrupt is generated		
mtspr	The designated SPR is TFHAR MSR_{TS} = 01	A TM Bad Thing type Program interrupt is generated

17.5.3. Special cases of mfspr

Instruction sequence	Compliance conditions	Expected result
III.4.4.4.mfspr.1 Execution of mfspr specifying a privileged SPR causes a Privileged Instruction type Program interrupt when MSR_{PR} =1		
mfspr	spr_0 = 1 MSR_{PR} = 1	A Privileged Instruction type Program interrupt is generated
III.4.4.4.mfspr.2 Execution of mfspr specifying a hypervisor-privileged SPR causes a Privileged Instruction type Program interrupt when $MSR_{HV\ PR}$ = 0b00 and $LPCR_{EVIRT}$ = 0		
mfspr	The designated SPR is in *SPRs that are hypervisor-privileged for mfspr* $MSR_{HV\ PR}$ = 0b00 $LPCR_{EVIRT}$ = 0	A Privileged Instruction type Program interrupt is generated
III.4.4.4.mfspr.2b Execution of mfspr specifying a hypervisor-privileged SPR causes a Hypervisor Emulation Assistance interrupt when $MSR_{HV\ PR}$ = 0b00 and $LPCR_{EVIRT}$ = 1		
mfspr	The designated SPR is in *SPRs that are hypervisor-privileged for mfspr* $MSR_{HV\ PR}$ = 0b00 $LPCR_{EVIRT}$ = 1	A Hypervisor Emulation Assistance interrupt is generated
III.4.4.4.mfspr.3 Execution of mfspr specifying an SPR number that is not defined for the implementation, if spr_0 = 0 and MSR_{PR} = 1		
mfspr	The designated SPR is in *SPRs that are not defined for mfspr* spr_0 = 0 MSR_{PR} = 1	A Hypervisor Emulation Assistance interrupt is generated
III.4.4.4.mfspr.4 Execution of mfspr specifying SPR 0,4,5 or 6 that is not defined for the implementation, if spr_0 = 0 and MSR_{PR} = 0		
mfspr	The designated SPR is SPR 0, 4, 5, or 6 and is in *SPRs that are not defined for mfspr* spr_0 = 0 MSR_{PR} = 0	A Hypervisor Emulation Assistance interrupt is generated
III.4.4.4.mfspr.5 Execution of mfspr specifying an SPR number (not SPR 0,4,5 or 6) that is not defined for the implementation, if spr_0 = 0, MSR_{PR} = 0, and $LPCR_{EVIRT}$ = 0		
mfspr	The designated SPR is not SPR 0, 4, 5, or 6 and is in *SPRs that are not defined for mfspr* spr_0 = 0 MSR_{PR} = 0 $LPCR_{EVIRT}$ = 0	No operation
III.4.4.4.mfspr.5b Execution of mfspr specifying an SPR number (not SPR 0,4,5 or 6) that is not defined for the implementation, if spr_0 = 0, MSR_{PR} = 0, and $LPCR_{EVIRT}$ = 1		
mfspr	The designated SPR is not SPR 0, 4, 5, or 6 and is in *SPRs that are not defined for mfspr* spr_0 = 0 MSR_{PR} = 0 $LPCR_{EVIRT}$ = 1	A Hypervisor Emulation Assistance interrupt is generated

Instruction sequence	Compliance conditions	Expected result
III.4.4.4.mfspr.6 Execution of mfspr specifying an SPR number that is not defined for the implementation, if spr_0 = 1 and MSR_{PR} = 1		
mfspr	The designated SPR is in *SPRs that are not defined for mfspr* spr_0 = 1 MSR_{PR} = 1	A Privileged Instruction type Program interrupt is generated
III.4.4.4.mfspr.7 Execution of mfspr specifying an SPR number that is not defined for the implementation, if spr_0 = 1, MSR_{PR} = 0, and $LPCR_{EVIRT}$ = 0		
mfspr	The designated SPR is in *SPRs that are not defined for mfspr* spr_0 = 1 MSR_{PR} = 0 $LPCR_{EVIRT}$ = 0	No operation
III.4.4.4.mfspr.7b Execution of mfspr specifying an SPR number that is not defined for the implementation, if spr_0 = 1, MSR_{PR} = 0, and $LPCR_{EVIRT}$ = 1		
mfspr	The designated SPR is in *SPRs that are not defined for mfspr* spr_0 = 1 MSR_{PR} = 0 $LPCR_{EVIRT}$ = 1	A Hypervisor Emulation Assistance interrupt is generated

17.5.4. Move to/from Performance Monitor SPRs

Group A: non-privileged read/write Performance Monitor registers: PMC1-PMC6, MMCR0, MMCR2, MMCRA

Group B: Non-privileged read-only Performance Monitor SPRs: SIER, SIAR, SDAR, MMCR1

Non-Privileged

Non-privileged Performance Monitor registers: the union of *Group A* and *Group B*

Instruction sequence	Compliance conditions	Expected result
III.6.2.12.PM.1 When $HFSCR_{PM}$ = 0, all non-privileged Performance Monitor SPRs are **not** available for read (mfspr) in problem state or privileged non-hypervisor state		
mfspr	The designated SPR is in *Non-privileged Performance Monitor registers* $HFSCR_{PM}$ = 0 The program is in problem state or privileged non-hypervisor state	A Hypervisor Facility Unavailable interrupt is generated
III.6.2.12.PM.2 When $MMCR0_{PMCC}$ = 00 and $HFSCR_{PM}$ = 1, all non-privileged Performance Monitor SPR are available for read (mfspr) in problem state		
mfspr	The designated SPR is in *Non-privileged Performance Monitor registers* $MMCR0_{PMCC}$ = 00 $HFSCR_{PM}$ = 1 The program is in problem state	The designated SPR is read correctly

Instruction sequence	Compliance conditions	Expected result
III.6.2.12.PM.3 When $MMCR0_{PMCC}$ = 01 and $HFSCR_{PM}$ = 1, all non-privileged Performance Monitor SPRs are **not** available for read (mfspr) in problem state		
mfspr	The designated SPR is in *Non-privileged Performance Monitor registers* $MMCR0_{PMCC}$ = 01 $HFSCR_{PM}$ = 1 The program is in problem state	A Facility Unavailable interrupt is generated
III.6.2.12.PM.4 When $MMCR0_{PMCC}$ = 10 and $HFSCR_{PM}$ = 1, all non-privileged Performance Monitor SPRs, except for MMCR1, are available for read (mfspr) in problem state		
mfspr	The designated SPR is in *Non-privileged Performance Monitor registers*, other than MMCR1 $MMCR0_{PMCC}$ = 10 $HFSCR_{PM}$ = 1 The program is in problem state	The designated SPR is read correctly
III.6.2.12.PM.5 When $MMCR0_{PMCC}$ = 10 and $HFSCR_{PM}$ = 1, MMCR1 is **not** available for read (mfspr) in problem state		
mfspr	The designated SPR is MMCR1 $MMCR0_{PMCC}$ = 10 $HFSCR_{PM}$ = 1 The program is in problem state	A Facility Unavailable interrupt is generated
III.6.2.12.PM.6 When $MMCR0_{PMCC}$ = 11 and $HFSCR_{PM}$ = 1, all non-privileged Performance Monitor SPRs, except for PMCs 5-6 and MMCR1, are available for read (mfspr) in problem state		
mfspr	The designated SPR is in *Non-privileged Performance Monitor registers*, other than PMCs 5-6 and MMCR1 $MMCR0_{PMCC}$ = 11 $HFSCR_{PM}$ = 1 The program is in problem state	The designated SPR is read correctly
III.6.2.12.PM.7 When $MMCR0_{PMCC}$ = 11 and $HFSCR_{PM}$ = 1, PMCs 5-6 and MMCR1 are **not** available for read (mfspr) in problem state		
mfspr	The designated SPR is PMC5, PMC6, or MMCR1 $MMCR0_{PMCC}$ = 11 $HFSCR_{PM}$ = 1 The program is in problem state	A Facility Unavailable interrupt is generated
III.6.2.12.PM.8 Group B Performance Monitor SPRs are read-only, and cannot be written in any privilege state		
mtspr	The designated SPR is in *Group B: Non-privileged read-only Performance Monitor SPRs* $MMCR0_{PMCC}$ = 00 $HFSCR_{PM}$ = 1 The program is in hypervisor state	Instruction behaves as if the designated SPR is not defined for the implementation
III.6.2.12.PM.9 When $HFSCR_{PM}$ = 0, all Group A Performance Monitor SPRs are **not** available for write (mtspr) in problem state or privileged non-hypervisor state		
mtspr	The designated SPR is in *Group A: non-privileged read/write Performance Monitor registers*	A Hypervisor Facility Unavailable interrupt is generated

Instruction sequence	Compliance conditions	Expected result
	$HFSCR_{PM}$ = 0 The program is in problem state or privileged non-hypervisor state	
III.6.2.12.PM.10 When $MMCR0_{PMCC}$ = 00 and $HFSCR_{PM}$ = 1, Group A Performance Monitor SPRs are **not** available for write (mtspr) in problem state		
mtspr	The designated SPR is in *Group A: non-privileged read/write Performance Monitor registers* $MMCR0_{PMCC}$ = 00 $HFSCR_{PM}$ = 1 The program is in problem state	A Hypervisor Emulation Assistance interrupt is generated
III.6.2.12.PM.11 When $MMCR0_{PMCC}$ = 01 and $HFSCR_{PM}$ = 1, Group A Performance Monitor SPRs are **not** available for write (mtspr) in problem state		
mtspr	The designated SPR is in *Group A: non-privileged read/write Performance Monitor registers* $MMCR0_{PMCC}$ = 01 $HFSCR_{PM}$ = 1 The program is in problem state	A Facility Unavailable interrupt is generated
III.6.2.12.PM.12 When $MMCR0_{PMCC}$ = 10 and $HFSCR_{PM}$ = 1, Group A Performance Monitor SPRs are available for write (mtspr) in problem state		
mtspr	The designated SPR is in *Group A: non-privileged read/write Performance Monitor registers* $MMCR0_{PMCC}$ = 10 $HFSCR_{PM}$ = 1 The program is in problem state	The designated SPR is written correctly
III.6.2.12.PM.13 When $MMCR0_{PMCC}$ = 11 and $HFSCR_{PM}$ = 1, Group A Performance Monitor SPRs, except for PMCs 5-6, are available for write (mtspr) in problem state		
mtspr	The designated SPR is in *Group A: non-privileged read/write Performance Monitor registers*, other than PMC5 or PMC6 $MMCR0_{PMCC}$ = 11 $HFSCR_{PM}$ = 1 The program is in problem state	The designated SPR is written correctly
III.6.2.12.PM.14 When $MMCR0_{PMCC}$ = 11 and $HFSCR_{PM}$ = 1, PMCs 5-6 are **not** available for write (mtspr) in problem state		
mtspr	The designated SPR is PMC5 or PMC6 $MMCR0_{PMCC}$ = 11 $HFSCR_{PM}$ = 1 The program is in problem state	A Facility Unavailable interrupt is generated

17.5.5. mtmsr, mtmsrd, mfmsr

Illegal transaction state transition: illegal transitions listed in Table 3 in section **III.3.2.2 State Transitions Associated with the Transactional Memory Facility**

Guideline: each scenario in this group should be tested for every instruction in the **Instructions tested** list

Instructions tested	Compliance conditions	Expected result
III.4.4.4.MSR.1 Correct instruction execution		
mtmsr mtmsrd mfmsr		Correct instruction behavior
III.4.4.4.MSR.2 These instructions are privileged		
mtmsr mtmsrd mfmsr	MSR_{PR} = 1	A Privileged Instruction type Program interrupt is generated
III.4.4.4.MSR.3 If mtmsrd attempts to cause an illegal transaction state transition, a TM Bad Thing type Program interrupt is generated		
mtmsrd	The instruction attempts to cause an *Illegal transaction state transition*	A TM Bad Thing type Program interrupt is generated
III.4.4.4.MSR.4 When TM is disabled by the PCR, if mtmsrd attempts to cause a transition to Problem state with an active transaction, a TM Bad Thing type Program interrupt is generated		
mtmsrd	TM is disabled by the PCR The instruction attempts to cause a transition to Problem state with an active transaction	A TM Bad Thing type Program interrupt is generated

18. Storage Control (Chapter III.5)

Table of Contents

Below are the scenarios that verify correct implementation of the Storage Control mechanism as described in the Power ISA Book III Chapter 5.

All scenarios should be implemented in 32- and 64-bit modes.

The following instruction groups are used in scenarios:

Load : Instructions from Sections 3.3.2, 4.6.2 of Book I.

Store : Instructions from Sections 3.3.3, 4.6.3 of Book I, dcbz

18.1. Hypervisor Real and Virtual Real Addressing Modes

Architecture sections:

- III.5.7.3 Hypervisor Real And Virtual Real Addressing Modes
- III.5.7.6.1 Partition Table
- III.5.7.6.2 Process Table

Instruction sequence	Observability preconditions	Expected results
III.5.StorageCont.1 Processor P1 accesses memory locations A and B in Hypervisor Offset Real Mode		
P1 Load from A Store to A Load from B Store to B	MSR[DR] = 0 MSR[HV] = 1 Bit 0 of the Effective Address of memory location A is 0 Bit 0 of the Effective Address of memory location B is 1 Initial values of the source registers used by Stores and of locations A and B are different.	Memory location A and B are updated by the Stores.
III.5.StorageCont.2 Processor P1 performs instruction fetch in Hypervisor Offset Real Mode		
P1 Any arithmetic integer instruction	MSR[IR] = 0 MSR[HV] = 1	The instruction is successfully executed.

Instruction sequence	Observability preconditions	Expected results
	Bit 0 of the instruction Effective Address is 0	
III.5.StorageCont.3 Processor P1 performs instruction fetch in Hypervisor Offset Real Mode		
P1 Any arithmetic integer instruction	MSR[IR] = 0 MSR[HV] = 1 Bit 0 of the instruction Effective Address is 1	The instruction is successfully executed.
III.5.StorageCont.4 Processor P1 accesses memory location A in Virtual Real Mode Addressing		
P1 Load from A Store to A	MSR[DR] = 0 MSR[HV] = 0 MSR[PR] = 0 PATE[HR] = 0 Initial value of the source register used by Store and of location A are different.	Memory location A is updated by the Store.
III.5.StorageCont.5 Processor P1 performs instruction fetch in Virtual Real Mode Addressing		
P1 Any arithmetic integer instruction	MSR[IR] = 0 MSR[HV] = 0 MSR[PR] = 0 PATE[HR] = 0	The instruction is successfully executed.
III.5.StorageCont.6 Processor P1 accesses memory location A in Host Real Address, Radix on Radix Mode (Guest OS access with translation off)		
P1 Load from A Store to A	MSR[DR] = 0 MSR[HV] = 0 MSR[PR] = 0 PATE[HR] = 1 Memory access is performed with no interrupt. Initial value of the source register used by Store and of location A are different.	Memory location A is updated by the Store.
III.5.StorageCont.7 Processor P1 performs instruction fetch in Host Real Address, Radix on Radix Mode (Guest OS access with translation off)		
P1 Any arithmetic integer instruction	MSR[IR] = 0 MSR[HV] = 0 MSR[PR] = 0 PATE[HR] = 1 Instruction fetch is performed with no interrupt.	The instruction is successfully executed.

18.2. Radix Tree Translation

Architecture sections:

- III.5.7.5.1 Effective Address Space Structure for Radix-using Partitions

- III.5.7.6.1 Partition Table
- III.5.7.6.2 Process Table
- III.5.7.10 Radix Tree Translation
- III.5.7.11. Translation Process

Instruction sequence	Observability preconditions	Expected results
III.5.7.5.Radix.1 Processor P1 accesses memory location A in quadrant 0 ($EA_{0:1}$=0b00) for the guest application or guest operating system (LPIDR != 0)		
P1 Load from A Store to A	LPIDR != 0 $EA_{0:1}$=0b00 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 0 Initial value of the source register used by Store and of location A are different. effPID=PIDR effLPID=LPIDR	Memory location A is updated by the Store.
III.5.7.5.Radix.2 Processor P1 performs instruction fetch in quadrant 0 ($EA_{0:1}$=0b00) for the guest application or guest operating system (LPIDR != 0)		
P1 Any arithmetic integer instructions	LPIDR != 0 $EA_{0:1}$=0b00 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 0 effPID=PIDR effLPID=LPIDR	The instruction is successfully executed.
III.5.7.5.Radix.3 Processor P1 accesses memory location A in quadrant 1 ($EA_{0:1}$=0b01) for the guest application or guest operating system which is invalid and causes a Data Segment exception (LPIDR != 0)		
P1 Load from A Store to A	LPIDR != 0 $EA_{0:1}$=0b01 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 0 Initial value of the source register used by Store and of location A are different. effPID=xx effLPID=xx	Results in Data Segment exception
III.5.7.5.Radix.4 Processor P1 performs instruction fetch in quadrant 1 ($EA_{0:1}$=0b01) for the guest application or guest operating system which is invalid and causes a Instruction Segment exception (LPIDR != 0)		
P1 Any arithmetic integer instructions	LPIDR != 0 $EA_{0:1}$=0b01 PATE[HR]=1 MSR[IR] = 1	Results in Instruction Segment exception

Instruction sequence	Observability preconditions	Expected results
	MSR[HV] = 0 effPID=xx effLPID=xx	
III.5.7.5.Radix.5 Processor P1 accesses memory location A in quadrant 2 ($EA_{0:1}$=0b10) for the guest application or guest operating system which is invalid and causes a Data Segment exception (LPIDR != 0)		
P1 Load from A Store to A	LPIDR != 0 $EA_{0:1}$=0b10 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 0 Initial value of the source register used by Store and of location A are different. effPID=xx effLPID=xx	Results in Data Segment exception
III.5.7.5.Radix.6 Processor P1 performs instruction fetch in quadrant 2 ($EA_{0:1}$=0b10) for the guest application or guest operating system which is invalid and causes a Instruction Segment exception (LPIDR != 0)		
P1 Any arithmetic integer instructions	LPIDR != 0 $EA_{0:1}$=0b10 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 0 effPID=xx effLPID=xx	Results in Instruction Segment exception
III.5.7.5.Radix.7 Processor P1 accesses memory location A in quadrant 3 ($EA_{0:1}$=0b11) for the guest application or guest operating system (LPIDR != 0)		
P1 Load from A Store to A	LPIDR != 0 $EA_{0:1}$=0b11 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 0 Initial value of the source register used by Store and of location A are different. effPID=0 effLPID=LPIDR	Memory location A is updated by the Store.
III.5.7.5.Radix.8 Processor P1 performs instruction fetch in quadrant 3 ($EA_{0:1}$=0b11) for the guest application or guest operating system (LPIDR != 0)		
P1 Any arithmetic integer instructions	LPIDR != 0 $EA_{0:1}$=0b11 PATE[HR]=1 MSR[IR] = 1	The instruction is successfully executed.

Instruction sequence	Observability preconditions	Expected results
	MSR[HV] = 0 effPID=0 effLPID=LPIDR	
III.5.7.5.Radix.9 Processor P1 accesses memory location A in quadrant 0 ($EA_{0:1}$=0b00) for the host application (LPIDR != 0)		
P1 Load from A Store to A	LPIDR != 0 $EA_{0:1}$=0b00 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 1 MSR[PR] = 1 Initial value of the source register used by Store and of location A are different. effPID=PIDR effLPID=0	Memory location A is updated by the Store.
III.5.7.5.Radix.10 Processor P1 performs instruction fetch in quadrant 0 ($EA_{0:1}$=0b00) for the host application (LPIDR != 0)		
P1 Any arithmetic integer instructions	LPIDR != 0 $EA_{0:1}$=0b00 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 1 effPID=PIDR effLPID=0	The instruction is successfully executed.
III.5.7.5.Radix.11 Processor P1 accesses memory location A in quadrant 1 ($EA_{0:1}$=0b01) for the host application which is invalid and causes a Data Segment exception (LPIDR != 0)		
P1 Load from A Store to A	LPIDR != 0 $EA_{0:1}$=0b01 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 1 MSR[PR] = 1 Initial value of the source register used by Store and of location A are different. effPID=xx effLPID=xx	Results in Data Segment exception
III.5.7.5.Radix.12 Processor P1 performs instruction fetch in quadrant 1 ($EA_{0:1}$=0b01) for the host application which is invalid and causes an Instruction Segment exception (LPIDR != 0)		
P1 Any arithmetic integer instructions	LPIDR != 0 $EA_{0:1}$=0b01	Results in Instruction Segment exception

Instruction sequence	Observability preconditions	Expected results
	PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 1 effPID=xx effLPID=xx	
III.5.7.5.Radix.13 Processor P1 accesses memory location A in quadrant 2 ($EA_{0:1}$=0b10) for the host application which is invalid and causes a Data Segment exception (LPIDR != 0)		
P1 Load from A Store to A	LPIDR != 0 $EA_{0:1}$=0b10 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 1 MSR[PR] = 1 Initial value of the source register used by Store and of location A are different. effPID=xx effLPID=xx	Results in Data Segment exception
III.5.7.5.Radix.14 Processor P1 performs instruction fetch in quadrant 2 ($EA_{0:1}$=0b10) for the host application which is invalid and causes an Instruction Segment exception (LPIDR != 0)		
P1 Any arithmetic integer instructions	LPIDR != 0 $EA_{0:1}$=0b10 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 1 effPID=xx effLPID=xx	Results in Instruction Segment exception
III.5.7.5.Radix.15 Processor P1 accesses memory location A in quadrant 3 ($EA_{0:1}$=0b11) for the host application (LPIDR != 0)		
P1 Load from A Store to A	LPIDR != 0 $EA_{0:1}$=0b11 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 1 MSR[PR] = 1 Initial value of the source register used by Store and of location A are different. effPID=0 effLPID=0	Memory location A is updated by the Store.

Instruction sequence	Observability preconditions	Expected results
III.5.7.5.Radix.16 Processor P1 performs instruction fetch in quadrant 3 ($EA_{0:1}$=0b11) for the host application (LPIDR != 0)		
P1 Any arithmetic integer instructions	LPIDR != 0 $EA_{0:1}$=0b11 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 1 effPID=0 effLPID=0	The instruction is successfully executed.
III.5.7.5.Radix.17 Processor P1 accesses memory location A in quadrant 0 ($EA_{0:1}$=0b00) for the hypervisor of the host application (LPIDR != 0)		
P1 Load from A Store to A	LPIDR != 0 $EA_{0:1}$=0b00 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 1 MSR[PR] = 0 Initial value of the source register used by Store and of location A are different. effPID=PIDR effLPID=0	Memory location A is updated by the Store.
III.5.7.5.Radix.18 Processor P1 performs instruction fetch in quadrant 0 ($EA_{0:1}$=0b00) for the hypervisor of the host application (LPIDR != 0)		
P1 Any arithmetic integer instructions	LPIDR != 0 $EA_{0:1}$=0b00 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 0 effPID=PIDR effLPID=0	The instruction is successfully executed.
III.5.7.5.Radix.19 Processor P1 accesses memory location A in quadrant 1 ($EA_{0:1}$=0b01) for the hypervisor of the guest application or guest operating system (LPIDR != 0)		
P1 Load from A Store to A	LPIDR != 0 $EA_{0:1}$=0b01 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 1 MSR[PR] = 0	Memory location A is updated by the Store.

Instruction sequence	Observability preconditions	Expected results
	Initial value of the source register used by Store and of location A are different. effPID=PIDR effLPID=LPIDR	
III.5.7.5.>Radix.20 Processor P1 performs instruction fetch in quadrant 1 ($EA_{0:1}$=0b01) for the hypervisor of the guest application or guest operating system (LPIDR != 0)		
P1 Any arithmetic integer instructions	LPIDR != 0 $EA_{0:1}$=0b01 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 0 effPID=PIDR effLPID=LPIDR	The instruction is successfully executed.
III.5.7.5.Radix.21 Processor P1 accesses memory location A in quadrant 2 ($EA_{0:1}$=0b10) for the hypervisor of the guest application or guest operating system (LPIDR != 0)		
P1 Load from A Store to A	LPIDR != 0 $EA_{0:1}$=0b10 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 1 MSR[PR] = 0 Initial value of the source register used by Store and of location A are different. effPID=0 effLPID=LPIDR	Memory location A is updated by the Store.
III.5.7.5.Radix.22 Processor P1 performs instruction fetch in quadrant 2 ($EA_{0:1}$=0b10) for the hypervisor of the guest application or guest operating system (LPIDR != 0)		
P1 Any arithmetic integer instructions	LPIDR != 0 $EA_{0:1}$=0b10 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 0 effPID=0 effLPID=LPIDR	The instruction is successfully executed.
III.5.7.5.Radix.23 Processor P1 accesses memory location A in quadrant 3 ($EA_{0:1}$=0b11) for the hypervisor of the host application (LPIDR != 0)		
P1 Load from A	LPIDR != 0 $EA_{0:1}$=0b11	Memory location A is updated by the Store.

Instruction sequence	Observability preconditions	Expected results
Store to A	PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 1 MSR[PR] = 0 Initial value of the source register used by Store and of location A are different. effPID=0 effLPID=0	
III.5.7.5.Radix.24 Processor P1 performs instruction fetch in quadrant 3 ($EA_{0:1}$=0b11) for the hypervisor of the host application (LPIDR != 0)		
P1 Any arithmetic integer instructions	LPIDR != 0 $EA_{0:1}$=0b11 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 0 effPID=0 effLPID=0	The instruction is successfully executed.
III.5.7.5.Radix.25 Processor P1 accesses memory location A in quadrant 0 ($EA_{0:1}$=0b00) for the guest application or guest operating system which is invalid and causes a Hypervisor Data Storage exception (LPIDR = 0)		
P1 Load from A Store to A	LPIDR = 0 $EA_{0:1}$=0b00 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 0 Initial value of the source register used by Store and of location A are different. effPID=xx effLPID=xx	Results in Hypervisor Data Storage exception
III.5.7.5.Radix.26 Processor P1 performs instruction fetch in quadrant 0 ($EA_{0:1}$=0b00) for the guest application or guest operating system which is invalid and causes a Hypervisor Instruction Storage exception (LPIDR = 0)		
P1 Any arithmetic integer instructions	LPIDR = 0 $EA_{0:1}$=0b00 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 0 effPID=xx effLPID=xx	Results in Hypervisor Instruction Storage exception
III.5.7.5.Radix.27 Processor P1 accesses memory location A in quadrant 1 ($EA_{0:1}$=0b01) for the guest application or guest operating system which is invalid and causes a Hypervisor Data Storage exception (LPIDR = 0)		

Instruction sequence	Observability preconditions	Expected results
P1 Load from A Store to A	LPIDR = 0 $EA_{0:1}$=0b01 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 0 Initial value of the source register used by Store and of location A are different. effPID=xx effLPID=xx	Results in Hypervisor Data Storage exception
III.5.7.5.Radix.28 Processor P1 performs instruction fetch in quadrant 1 ($EA_{0:1}$=0b01) for the guest application or guest operating system which is invalid and causes a Hypervisor Instruction Storage exception (LPIDR = 0)		
P1 Any arithmetic integer instructions	LPIDR = 0 $EA_{0:1}$=0b01 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 0 effPID=xx effLPID=xx	Results in Hypervisor Instruction Storage exception
III.5.7.5.Radix.29 Processor P1 accesses memory location A in quadrant 2 ($EA_{0:1}$=0b10) for the guest application or guest operating system which is invalid and causes a Hypervisor Data Storage exception (LPIDR = 0)		
P1 Load from A Store to A	LPIDR = 0 $EA_{0:1}$=0b10 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 0 Initial value of the source register used by Store and of location A are different. effPID=xx effLPID=xx	Results in Hypervisor Data Storage exception
III.5.7.5.Radix.30 Processor P1 performs instruction fetch in quadrant 2 ($EA_{0:1}$=0b10) for the guest application or guest operating system which is invalid and causes a Hypervisor Instruction Storage exception (LPIDR = 0)		
P1 Any arithmetic integer instructions	LPIDR = 0 $EA_{0:1}$=0b10 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 0 effPID=xx effLPID=xx	Results in Hypervisor Instruction Storage exception
III.5.7.5.Radix.31 Processor P1 accesses memory location A in quadrant 3 ($EA_{0:1}$=0b11) for the guest application or guest operating system which is invalid and causes a Hypervisor Data Storage exception (LPIDR = 0)		

Instruction sequence	Observability preconditions	Expected results
P1 Load from A Store to A	LPIDR = 0 $EA_{0:1}$=0b11 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 0 Initial value of the source register used by Store and of location A are different. effPID=xx effLPID=xx	Results in Hypervisor Data Storage exception
III.5.7.5.Radix.32 Processor P1 performs instruction fetch in quadrant 3 ($EA_{0:1}$=0b11) for the guest application or guest operating system which is invalid and causes a Hypervisor Instruction Storage exception (LPIDR = 0)		
P1 Any arithmetic integer instructions	LPIDR = 0 $EA_{0:1}$=0b11 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 0 effPID=xx effLPID=xx	Results in Hypervisor Instruction Storage exception
III.5.7.5.Radix.33 Processor P1 accesses memory location A in quadrant 0 ($EA_{0:1}$=0b00) for the host application (LPIDR = 0)		
P1 Load from A Store to A	LPIDR = 0 $EA_{0:1}$=0b00 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 1 MSR[PR] = 1 Initial value of the source register used by Store and of location A are different. effPID=PIDR effLPID=0	Memory location A is updated by the Store.
III.5.7.5.Radix.34 Processor P1 performs instruction fetch in quadrant 0 ($EA_{0:1}$=0b00) for the host application (LPIDR = 0)		
P1 Any arithmetic integer instructions	LPIDR = 0 $EA_{0:1}$=0b00 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 1 effPID=PIDR effLPID=0	The instruction is successfully executed.

Instruction sequence	Observability preconditions	Expected results
III.5.7.5.Radix.35 Processor P1 accesses memory location A in quadrant 1 ($EA_{0:1}$=0b01) for the host application which is invalid and causes a Data Segment exception (LPIDR = 0)		
P1 Load from A Store to A	LPIDR = 0 $EA_{0:1}$=0b01 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 1 MSR[PR] = 1 Initial value of the source register used by Store and of location A are different. effPID=xx effLPID=xx	Results in Data Segment exception
III.5.7.5.Radix.36 Processor P1 performs instruction fetch in quadrant 1 ($EA_{0:1}$=0b01) for the host application which is invalid and causes an Instruction Segment exception (LPIDR = 0)		
P1 Any arithmetic integer instructions	LPIDR = 0 $EA_{0:1}$=0b01 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 1 effPID=xx effLPID=xx	Results in Instruction Segment exception
III.5.7.5.Radix.37 Processor P1 accesses memory location A in quadrant 2 ($EA_{0:1}$=0b10) for the host application which is invalid and causes a Data Segment exception (LPIDR = 0)		
P1 Load from A Store to A	LPIDR = 0 $EA_{0:1}$=0b10 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 1 MSR[PR] = 1 Initial value of the source register used by Store and of location A are different. effPID=xx effLPID=xx	Results in Data Segment exception
III.5.7.5.Radix.38 Processor P1 performs instruction fetch in quadrant 2 ($EA_{0:1}$=0b10) for the host application which is invalid and causes an Instruction Segment exception (LPIDR = 0)		
P1 Any arithmetic integer instructions	LPIDR = 0 $EA_{0:1}$=0b10 PATE[HR]=1 MSR[IR] = 1	Results in Instruction Segment exception

Instruction sequence	Observability preconditions	Expected results
	MSR[HV] = 1 MSR[PR] = 1 effPID=xx effLPID=xx	
III.5.7.5.Radix.39 Processor P1 accesses memory location A in quadrant 3 ($EA_{0:1}$=0b11) for the host application (LPIDR = 0)		
P1 Load from A Store to A	LPIDR = 0 $EA_{0:1}$=0b11 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 1 MSR[PR] = 1 Initial value of the source register used by Store and of location A are different. effPID=0 effLPID=0	Memory location A is updated by the Store.
III.5.7.5.Radix.40 Processor P1 performs instruction fetch in quadrant 3 ($EA_{0:1}$=0b11) for the host application (LPIDR = 0)		
P1 Any arithmetic integer instructions	LPIDR = 0 $EA_{0:1}$=0b11 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 1 effPID=0 effLPID=0	The instruction is successfully executed.
III.5.7.5.Radix.41 Processor P1 accesses memory location A in quadrant 0 ($EA_{0:1}$=0b00) for the hypervisor of the host application (LPIDR = 0)		
P1 Load from A Store to A	LPIDR = 0 $EA_{0:1}$=0b00 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 1 MSR[PR] = 0 Initial value of the source register used by Store and of location A are different. effPID=PIDR effLPID=0	Memory location A is updated by the Store.
III.5.7.5.Radix.42 Processor P1 performs instruction fetch in quadrant 0 ($EA_{0:1}$=0b00) for the hypervisor of the host application (LPIDR = 0)		
P1	LPIDR = 0	The instruction is successfully executed.

Instruction sequence	Observability preconditions	Expected results
Any arithmetic integer instructions	$EA_{0:1}$=0b00 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 0 effPID=PIDR effLPID=0	
III.5.7.5.Radix.43 Processor P1 accesses memory location A in quadrant 1 ($EA_{0:1}$=0b01) for the hypervisor of the host application (LPIDR = 0)		
P1 Load from A Store to A	LPIDR = 0 $EA_{0:1}$=0b01 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 1 MSR[PR] = 0 Initial value of the source register used by Store and of location A are different. effPID=PIDR effLPID=LPIDR	Memory location A is updated by the Store.
III.5.7.5.Radix.44 Processor P1 performs instruction fetch in quadrant 1 ($EA_{0:1}$=0b01) for the hypervisor of the host application (LPIDR = 0)		
P1 Any arithmetic integer instructions	LPIDR = 0 $EA_{0:1}$=0b01 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 0 effPID=PIDR effLPID=LPIDR	The instruction is successfully executed.
III.5.7.5.Radix.45 Processor P1 accesses memory location A in quadrant 2 ($EA_{0:1}$=0b10) for the hypervisor of the host application (LPIDR = 0)		
P1 Load from A Store to A	LPIDR = 0 $EA_{0:1}$=0b10 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 1 MSR[PR] = 0 Initial value of the source register used by Store and of location A are different.	Memory location A is updated by the Store.

Instruction sequence	Observability preconditions	Expected results
	effPID=0 effLPID=LPIDR	
III.5.7.5.Radix.46 Processor P1 performs instruction fetch in quadrant 2 ($EA_{0:1}$=0b10) for the hypervisor of the host application (LPIDR = 0)		
P1 Any arithmetic integer instructions	LPIDR = 0 $EA_{0:1}$=0b10 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 0 effPID=0 effLPID=LPIDR	The instruction is successfully executed.
III.5.7.5.Radix.47 Processor P1 accesses memory location A in quadrant 3 ($EA_{0:1}$=0b11) for the hypervisor of the host application (LPIDR = 0)		
P1 Load from A Store to A	LPIDR = 0 $EA_{0:1}$=0b11 PATE[HR]=1 MSR[DR] = 1 MSR[HV] = 1 MSR[PR] = 0 Initial value of the source register used by Store and of location A are different. effPID=0 effLPID=0	Memory location A is updated by the Store.
III.5.7.5.Radix.48 Processor P1 performs instruction fetch in quadrant 3 ($EA_{0:1}$=0b11) for the hypervisor of the host application (LPIDR = 0)		
P1 Any arithmetic integer instructions	LPIDR = 0 $EA_{0:1}$=0b11 PATE[HR]=1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 0 effPID=0 effLPID=0	The instruction is successfully executed.

18.3. Virtual Address Generation

Architecture sections:

- III.5.7.8 Segment Translation Generation

Instruction sequence	Observability preconditions	Expected results
III.5.StorageCont.1 Processor P1 performs instruction fetch in Translation On mode		
P1 Any arithmetic integer instruction	MSR[IR] = 1 Instruction fetch is performed with no interrupt. Segment Size Selector is set to 0b00 (256 MB) Segment is Executable (N = 0)	The instruction is successfully executed.
III.5.StorageCont.2 Processor P1 performs instruction fetch in Translation On mode		
P1 Any arithmetic integer instruction	MSR[IR] = 1 Instruction fetch is performed with no interrupt. Segment Size Selector is set to 0b01 (1 TB) Segment is Executable (N = 0)	The instruction is successfully executed.
III.5.StorageCont.3 Processor P1 fails to perform instruction fetch in Translation On mode		
P1 Any arithmetic integer instruction	MSR[IR] = 1 Segment Size Selector is set to 0b00 (256 MB) Segment is No-execute (N = 1)	Instruction fetch fails. Instruction Segment Exception occurs. The instruction is not executed.
III.5.StorageCont.4 Processor P1 fails to perform instruction fetch in Translation On mode		
P1 Any arithmetic integer instruction	MSR[IR] = 1 Segment Size Selector is set to 0b01 (1 TB) Segment is No-execute (N = 1)	Instruction fetch fails. Instruction Segment Exception occurs. The instruction is not executed.
III.5.StorageCont.5 Processor P1 fails to access memory location A in Translation On mode		
P1 Load from A Store to A	MSR[DR] = 1 There is no valid SLB entry to translate Effective Address of A. Initial value of the source register used by Store and of location A are different.	Memory access to A fails. Data Segment Exception occurs. Memory location A is not updated by the Store.
III.5.StorageCont.6 The base page size is 4 KB according to the $SLB_{L\|\|LP}$ encoding		
P1 Load/Store instruction that is translated using the SLB	$SLBE_{L\|\|LP}$ = 0b000 The base page size is 4KB MSR[DR] = 1	The instruction is successfully executed.
III.5.StorageCont.7 The base page size is 64 KB according to the $SLB_{L\|\|LP}$ encoding		
P1 Load/store instruction that is translated using the SLB	$SLBE_{L\|\|LP}$ = 0b101 The base page size is 64KB MSR[DR] = 1	The instruction is successfully executed.

18.4. Virtual to Real Translation

Architecture sections:

- III.5.7.9 Hashed Page Table Translation

Instruction sequence	Observability preconditions	Expected results
III.5.StorageCont.1 Processor P1 accesses memory location A in Translation On mode		
P1 Load from A Store to A	MSR[DR] = 1 Memory access is performed with no interrupt. Segment Size Selector is set to 0b00 (256 MB) VA to RA translation is found in Primary PTEG. Page Size is 4K. Initial value of the source register used by Store and of location A are different.	Memory location A is updated by the Store.
III.5.StorageCont.2 Processor P1 accesses memory location A in Translation On mode		
P1 Load from A Store to A	MSR[DR] = 1 Memory access is performed with no interrupt. Segment Size Selector is set to 0b01 (1 TB) Initial value of the source register used by Store and of location A are different.	Memory location A is updated by the Store.
III.5.StorageCont.3 Processor P1 accesses memory location A in Translation On mode		
P1 Load from A Store to A	MSR[DR] = 1 Memory access is performed with no interrupt. Segment Size Selector is set to 0b00 (256 MB) VA to RA translation is found in Secondary PTEG. Initial value of the source register used by Store and of location A are different.	Memory location A is updated by the Store.
III.5.StorageCont.4 Processor P1 fails to perform instruction fetch in Translation On mode		
P1 Any arithmetic integer instruction	MSR[IR] = 1 Page is No-execute (N = 1)	Instruction fetch fails. Instruction Storage Exception occurs. The instruction is not executed.
III.5.StorageCont.5 Processor P1 fails to access memory location A in Translation On mode		
P1 Load from A Store to A	MSR[DR] = 1 There is a valid SLB entry to translate Effective Address of A. There is no valid PTE to translate Virtual Address of A. Initial value of the source register used by Store and of location A are different.	Memory access to A fails. Data Storage Exception occurs. Memory location A is not updated by the Store.
III.5.StorageCont.6 Processor P1 accesses memory location A in Translation On mode		
P1 Load from A	MSR[DR] = 1 Memory access is performed with no interrupt.	Memory location A is updated by the Store.

Instruction sequence	Observability preconditions	Expected results
Store to A	Segment Size Selector is set to 0b00 (256 MB) Page Size is 64K. Initial value of the source register used by Store and of location A are different.	
III.5.StorageCont.7 The base page size and actual page size are both 4KB		
P1 Load/store instruction that is translated using the page table	PTE_L = 0 (The actual page size is 4KB) $SLBE_{L\|\|LP}$ = 0b000 (The base page size is 4KB) MSR[DR] = 1	The instruction is successfully executed. Page size is 4KB
III.5.StorageCont.8 The base page size is 4KB, and the actual page size is 64KB		
P1 Load/store instruction that is translated using the page table	PTE_L = 1 $SLBE_{L\|\|LP}$ = 0b000 ((The base page size is 4KB) PTE_{LP} indicates that the actual page size is 64KB MSR[DR] = 1	The instruction is successfully executed. Page size is 64KB
III.5.StorageCont.9 The base page size and actual page size are both 64KB		
P1 Load/store instruction that is translated using the page table	PTE_L = 1 $SLBE_{L\|\|LP}$ = 0b101 ((The base page size is 64KB) PTE_{LP} indicates that the actual page size is 64KB MSR[DR] = 1	The instruction is successfully executed. Page size is 64KB

18.5. Reference and Change Recording

Architecture sections:

- III.5.7.12 Reference and Change Recording

Instruction sequence	Observability preconditions	Expected results
III.5.StorageCont.1 Processor P1 accesses memory location A in Translation On mode		
P1 Load from A Store to A	MSR[DR] = 1 Memory access is performed with no interrupt. Memory access to A crosses the virtual page boundary. Reference and Change Bits in all PTEs that map the Virtual Addresses of A are initialized to 0. Initial value of the source register used by Store and of location A are different.	Memory location A is updated by the Store. Reference and Change Bits are updated in all PTEs that map the Virtual Addresses of A.

Instruction sequence	Observability preconditions	Expected results
III.5.StorageCont.2 Processor P1 performs Store String Indexed instruction accessing memory location A in Translation On mode		
P1 Store String Indexed to A	MSR[DR] = 1 Memory access is performed with no interrupt. Reference and Change Bits in the PTE that maps the Virtual Address of A are initialized to 0. Length field of the XER register is set to 0. Initial value of the source register used by Store and of location A are different.	Memory location A is not updated by the Store. Reference and Change Bits in the PTE that maps the Virtual Addresses of A are not updated.
III.5.StorageCont.3 Processor P1 performs Store String Indexed instruction accessing memory location A in Translation On mode		
P1 Store String Indexed to A	MSR[DR] = 1 Memory access is performed with no interrupt. Reference and Change Bits in the PTE that maps the Virtual Address of A are initialized to 0. Length field of the XER register is set to a value other then 0. Initial value of the source register used by Store and of location A are different.	Memory location A is updated by the Store. Reference and Change Bits in the PTE that maps the Virtual Addresses of A are successfully updated.
III.5.StorageCont.4 Processor P1 performs Load String instruction accessing memory location A in Translation On mode		
P1 Load String from A	MSR[DR] = 1 Memory access is performed with no interrupt. Reference and Change Bits in the PTE that maps the Virtual Address of A are initialized to 0. Operand length is set to 0.	Load target registers are not updated. Reference Bit in the PTE that maps the Virtual Addresses of A is not updated.
III.5.StorageCont.5 Processor P1 performs Store Conditional instruction accessing memory location A in Translation On mode		
P1 Store Conditional to A	MSR[DR] = 1 Memory access is performed with no interrupt. Reference and Change Bits in the PTE that maps the Virtual Address of A are initialized to 0. Reservation bit of the reservation granule containing A is not set. Initial value of the source register used by Store and of location A are different.	Memory location A is not updated by the Store. Reference and Change Bits in the PTE that maps the Virtual Addresses of A are successfully updated.
III.5.StorageCont.6 Processor P1 performs Store instruction accessing memory location A in Translation On mode		
P1 Store to A	MSR[DR] = 1 Memory access is performed with Alignment Interrupt. Reference and Change Bits in the PTE that maps the Virtual Address of A are initialized to 0.	Memory location A is not updated by the Store. Reference and Change Bits in the PTE that maps the Virtual Addresses of A are successfully updated.

Instruction sequence	Observability preconditions	Expected results
	Initial value of the source register used by Store and of location A are different.	
III.5.StorageCont.7 Processor P1 performs Store Atomic instruction accessing memory location A in Translation On mode		
P1 Store Atomic	MSR[DR] = 1 Memory access is performed with no interrupt. Reference and Change Bits in the PTE that maps the Virtual Address of A are initialized to 0. Invalid function code Initial value of the source register used by Store and of location A are different.	Memory location A is not updated by the Store. Reference and Change Bits in the PTE that maps the Virtual Addresses of A are not updated.
III.5.StorageCont.8 Processor P1 performs Store Atomic instruction accessing memory location A in Translation On mode		
P1 Store Atomic	MSR[DR] = 1 Memory access is performed with no interrupt. Reference and Change Bits in the PTE that maps the Virtual Address of A are initialized to 0. Valid function code Initial value of the source register used by Store and of location A are different.	Memory location A is updated by the Store. Reference and Change Bits in the PTE that maps the Virtual Addresses of A are successfully updated.

18.6. Storage Protection

Architecture sections:

- III.5.7.13 Storage Protection
- III.5.8 Storage Control Attributes

Instruction sequence	Observability preconditions	Expected results
III.5.StorageCont.1 Processor P1 fails to read memory location A in Translation On mode		
P1 Load from A	MSR[DR] = 1 The field of the AMR register corresponding to the Key value of the PTE that maps the Virtual Address of A is initialized to 0bX1 (the first bit of the field is initialized to any value, the second bit of the field is initialized to 1).	Load target register is not updated. Data Storage Exception occurs.
III.5.StorageCont.2 Processor P1 fails to write memory location A in Translation On mode		
P1 Store to A	MSR[DR] = 1 The field of the AMR register corresponding to the Key value of the PTE that maps the Virtual Address of A is initialized to 0b1X (the first bit of the field is initialized to 1, the second bit of the field is initialized to any value). Initial value of the source register used by Store and of location A are different.	Memory location A not updated. Data Storage Exception occurs.

Instruction sequence	Observability preconditions	Expected results
III.5.StorageCont.3 Processor P1 fails to perform instruction fetch in Translation On mode		
P1 Any arithmetic integer instruction	MSR[IR] = 1 The field of the IAMR register corresponding to the Key value of the PTE that maps the instruction address is initialized to 0b01.	Instruction fetch fails. Instruction Storage Exception occurs. The instruction is not executed.
III.5.StorageCont.4 Processor P1 fails to perform instruction fetch in Translation On mode		
P1 Any arithmetic integer instruction	MSR[IR] = 1 The G bit of the PTE that maps the instruction address is initialized to 1.	Instruction fetch fails. Instruction Storage Exception occurs. The instruction is not executed.
III.5.StorageCont.5 Processor P1 fails to read memory location A in Translation On mode in privileged state		
P1 Load from A	MSR[DR] = 1 MSR[PR] = 0 The Ks field of the SLB entry that maps the Effective Address of A is initialized to 1. The PP field of the PTE that maps the Virtual Address of A is initialized to 0b000 or 0b110.	Load target register is not updated. Data Storage Exception occurs.
III.5.StorageCont.6 Processor P1 fails to perform instruction fetch in Translation On mode in privileged state		
P1 Any arithmetic integer instruction	MSR[IR] = 1 MSR[PR] = 0 The Ks field of the SLB entry that maps the instruction Effective Address is initialized to 1. The PP field of the PTE that maps the instruction Virtual Address is initialized to 0b000 or 0b110.	Instruction fetch fails. Instruction Storage Exception occurs. The instruction is not executed.
III.5.StorageCont.7 Processor P1 fails to read memory location A in Translation On mode in problem state		
P1 Load from A	MSR[DR] = 1 MSR[PR] = 1 The Kp field of the SLB entry that maps the Effective Address of A is initialized to 1. The PP field of the PTE that maps the Virtual Address of A is initialized to 0b000 or 0b110.	Load target register is not updated. Data Storage Exception occurs.
III.5.StorageCont.8 Processor P1 fails to perform instruction fetch in Translation On mode in problem state		
P1 Any arithmetic integer instruction	MSR[IR] = 1 MSR[PR] = 1 The Kp field of the SLB entry that maps the instruction Effective Address is initialized to 1. The PP field of the PTE that maps the instruction Virtual Address is initialized to 0b000 or 0b110.	Instruction fetch fails. Instruction Storage Exception occurs. The instruction is not executed.
III.5.StorageCont.9 Processor P1 fails to write memory location A in Translation On mode in privileged state		

Instruction sequence	Observability preconditions	Expected results
P1 Store to A	MSR[DR] = 1 MSR[PR] = 0 The Ks field of the SLB entry that maps the Effective Address of A is initialized to 0. The PP field of the PTE that maps the Virtual Address of A is initialized to 0b011 or 0b110. Initial value of the source register used by Store and of location A are different.	Memory location A is not updated. Data Storage Exception occurs.
III.5.StorageCont.10 Processor P1 fails to write memory location A in Translation On mode in privileged state		
P1 Store to A	MSR[DR] = 1 MSR[PR] = 0 The Ks field of the SLB entry that maps the Effective Address of A is initialized to 1. The PP field of the PTE that maps the Virtual Address of A is initialized to one of the following values: 0b000, 0b001, 0b011, 0b110. Initial value of the source register used by Store and of location A are different.	Memory location A is not updated. Data Storage Exception occurs.
III.5.StorageCont.11 Processor P1 fails to write memory location A in Translation On mode in problem state		
P1 Store to A	MSR[DR] = 1 MSR[PR] = 1 The Kp field of the SLB entry that maps the Effective Address of A is initialized to 0. The PP field of the PTE that maps the Virtual Address of A is initialized to 0b011 or 0b110. Initial value of the source register used by Store and of location A are different.	Memory location A is not updated. Data Storage Exception occurs.
III.5.StorageCont.12 Processor P1 fails to write memory location A in Translation On mode in problem state		
P1 Store to A	MSR[DR] = 1 MSR[PR] = 1 The Kp field of the SLB entry that maps the Effective Address of A is initialized to 1. The PP field of the PTE that maps the Virtual Address of A is initialized to one of the following values: 0b000, 0b001, 0b011, 0b110. Initial value of the source register used by Store and of location A are different.	Memory location A is not updated. Data Storage Exception occurs.
III.5.StorageCont.13 Processor P1 fails to write memory location A in Host Real Address, Radix on Radix Mode (Guest OS access with translation off)		

Instruction sequence	Observability preconditions	Expected results
P1 Store to A	MSR[DR] = 0 MSR[HV] = 0 MSR[PR] = 0 PATE[HR] = 1 Radix Tree Page Table Entry [EAA2]=Read/Write = 0 Initial value of the source register used by Store and of location A are different.	Memory location A is not updated by the Store. Hypervisor Data Storage Exception occurs.
III.5.StorageCont.14 Processor P1 fails to perform instruction fetch in Host Real Address, Radix on Radix Mode (Guest OS access with translation off)		
P1 Any arithmetic integer instruction	MSR[IR] = 0 MSR[HV] = 0 MSR[PR] = 0 PATE[HR] = 1 The G bit is set. Radix Tree Page Table Entry [58:59] Att Attributes = 0b10 (equivalent WIMG value = 0111 non-idempotent I/O)	Instruction fetch fails. Hypervisor Instruction Storage Exception occurs. The instruction is not executed.
III.5.StorageCont.15 Processor P1 fails to perform instruction fetch in Host Real Address, Radix on Radix Mode (Guest OS access with translation off)		
P1 Any arithmetic integer instruction	MSR[IR] = 0 MSR[HV] = 0 MSR[PR] = 0 PATE[HR] = 1 Radix Tree Page Table Entry [EAA3] = Execute = 0	Instruction fetch fails. Hypervisor Instruction Storage Exception occurs. The instruction is not executed.
III.5.StorageCont.16 Processor P1 successfully reads memory location A in Host Real Address, Radix on Radix Mode (Guest OS access with translation off) (the AMR register value is ignored)		
P1 Load from A	MSR[IR] = 0 MSR[HV] = 0 MSR[PR] = 0 PATE[HR] = 1 The field of the AMR register corresponding to the Key value of the PTE that maps the Virtual Address of A is initialized to 0bX1 (the first bit of the field is initialized to any value, the second bit of the field is initialized to 1).	Target register of Load is updated.
III.5.StorageCont.17 Processor P1 fails to access memory location A in Virtual Real Mode Addressing		
P1 Load from A	MSR[DR] = 0 MSR[HV] = 0 MSR[PR] = 0 PATE[HR] = 0	Target register of Load is not updated. Hypervisor Data Storage Exception occurs.

Instruction sequence	Observability preconditions	Expected results
	Software should specify PTEB = 0b01 for all Page Table Entries that map the VRMA The PP field of the PTE that maps the instruction Virtual Address is initialized to 0b011 or 0b110.	
III.5.StorageCont.18 Processor P1 fails to perform instruction fetch in Virtual Real Mode Addressing		
P1 Any arithmetic integer instruction	MSR[IR] = 0 MSR[HV] = 0 MSR[PR] = 0 PATE[HR] = 0 Software should specify PTEB = 0b01 for all Page Table Entries that map the VRMA The PP field of the PTE that maps the instruction Virtual Address is initialized to 0b011 or 0b110.	Instruction fetch fails. Hypervisor Instruction Storage Exception occurs. The instruction is not executed.

18.7. Storage Control Instructions

Architecture sections:

- III.5.9 Storage Control Instructions

Instruction sequence	Observability preconditions	Expected results
III.5.StorageCont.1 Processor P1 successfully performs slbie instruction		
P1 slbie	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 The V bit of the SLB entry specified by the slbie operand is initialized to 1. There is only one SLB entry that matches the contents of the slbie operand.	The SLB entry specified by the slbie operand is invalidated (V = 0)
III.5.StorageCont.1a Processor P1 successfully performs slbieg instruction in priveleged non-hypervisor state		
P1 slbieg	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 $LPCR_{GTSE}$ = 1 The V bit of the matching SLB entries specified by the slbieg are initialized to 1. There is only one SLB entry per thread that matches and is not software created.	For each thread with LPIDR=target_LPID and PIDR=target_PID, the matching SLB entry is invalidated (V = 0)
III.5.StorageCont.1b Processor P1 successfully performs slbieg instruction in priveleged hypervisor state		

Instruction sequence	Observability preconditions	Expected results
P1 slbieg	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 0 $LPCR_{GTSE}$ = 0 The V bit of the matching SLB entries specified by the slbieg are initialized to 1. There is only one SLB entry per thread that matches and is not software created.	For each thread with LPIDR=target_LPID and PIDR=target_PID, the matching SLB entry is invalidated (V = 0)
III.5.StorageCont.2 Processor P1 successfully performs slbia instruction for IH = 0b000, 0b001, 0b010, 0b110		
P1 slbia (test each case for IH = 0b000, 0b001, 0b010, 0b110)	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 The V bit of all SLB entries is initialized to 1.	All SLB entries except SLB entry 0 are invalidated (V = 0).
III.5.StorageCont.2a Processor P1 successfully performs slbia instruction for IH = 0b011		
P1 slbia (for IH = 0b011)	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 The V bit of all SLB entries is initialized to 1.	All SLB entries with $SLBE_{Class}$ = 1 are invalidated (V = 0).
III.5.StorageCont.2b Processor P1 successfully performs slbia instruction for IH = 0b100		
P1 slbia (for IH = 0b100)	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 The V bit of all SLB entries is initialized to 1.	All SLB entries are invalidated (V = 0).
III.5.StorageCont.2c Processor P1 successfully performs slbiag instruction in priveleged non-hypervisor state		
P1 slbiag	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 $LPCR_{GTSE}$ = 1 The V bit of the SLB entries specified by the slbiag are initialized to 1.	All matching SLB entries are invalidated (V = 0).
III.5.StorageCont.2d Processor P1 successfully performs slbiag instruction in priveleged hypervisor state		
P1 slbiag	MSR[DR] = 1 MSR[IR] = 1	All matching SLB entries are invalidated (V = 0).

Instruction sequence	Observability preconditions	Expected results
	MSR[HV] = 1 MSR[PR] = 0 $LPCR_{GTSE}$ = 0 The V bit of the SLB entries specified by the slbiag are initialized to 1.	
III.5.StorageCont.3 Processor P1 successfully performs slbmte instruction for $LPCR_{UPRT}$ = 0		
P1 slbmte	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 $LPCR_{UPRT}$ = 0 The target SLB entry and the source RS and RB registers are initialized to different values.	The target SLB entry is updated.
III.5.StorageCont.3a Processor P1 successfully performs slbmte instruction for $LPCR_{UPRT}$ = 1		
P1 slbmte	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 $LPCR_{UPRT}$ = 1 Value of index is 0, 1, 2, or 3 The target SLB entry and the source RS and RB registers are initialized to different values.	The target SLB entry is updated.
III.5.StorageCont.4 Processor P1 successfully performs slbmfev instruction for $LPCR_{UPRT}$ = 0		
P1 slbmfev	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 $LPCR_{UPRT}$ = 0 The source SLB entry, the source RB register and the target RT register are initialized to different values. The source SLB entry is initialized as valid (V=1).	The target RT register is updated.
III.5.StorageCont.4a Processor P1 successfully performs slbmfev instruction for $LPCR_{UPRT}$ = 1		
P1 slbmfev	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 $LPCR_{UPRT}$ = 1	The target RT register is updated.

Instruction sequence	Observability preconditions	Expected results
	Value of index is 0, 1, 2, or 3 The source SLB entry, the source RB register and the target RT register are initialized to different values. The source SLB entry is initialized as valid (V=1).	
III.5.StorageCont.5 Processor P1 successfully performs slbmfev instruction for $LPCR_{UPRT}$ = 0 with invalid entry (V=0)		
P1 slbmfev	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 $LPCR_{UPRT}$ = 0 The source SLB entry, the source RB register and the target RT register are initialized to different values. The source SLB entry is initialized as invalid (V=0).	The contents of the target RT register is set to zero.
III.5.StorageCont.5a Processor P1 successfully performs slbmfev instruction for $LPCR_{UPRT}$ = 1 with invalid entry (V=0)		
P1 slbmfev	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 $LPCR_{UPRT}$ = 1 Value of index is 0, 1, 2, or 3 The source SLB entry, the source RB register and the target RT register are initialized to different values. The source SLB entry is initialized as invalid (V=0).	The contents of the target RT register is set to zero.
III.5.StorageCont.6 Processor P1 successfully performs slbmfee instruction for $LPCR_{UPRT}$ = 0		
P1 slbmfee	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 $LPCR_{UPRT}$ = 0 The source SLB entry, the source RB register and the target RT register are initialized to different values. The source SLB entry is initialized as valid (V=1).	The contents of the target RT register is updated.
III.5.StorageCont.6a Processor P1 successfully performs slbmfee instruction for $LPCR_{UPRT}$ = 1		
P1 slbmfee	MSR[DR] = 1 MSR[IR] = 1	The contents of the target RT register is updated.

Instruction sequence	Observability preconditions	Expected results
	MSR[HV] = 0 MSR[PR] = 0 $LPCR_{UPRT}$ = 1 Value of index is 0, 1, 2, or 3 The source SLB entry, the source RB register and the target RT register are initialized to different values. The source SLB entry is initialized as valid (V=1).	
III.5.StorageCont.7 Processor P1 successfully performs slbmfee instruction for $LPCR_{UPRT}$ = 0 with invalid entry (V=0)		
P1 slbmfee	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 $LPCR_{UPRT}$ = 0 The source SLB entry, the source RB register and the target RT register are initialized to different values. The source SLB entry is initialized as invalid (V=0).	The contents of the target RT register is set to zero.
III.5.StorageCont.7a Processor P1 successfully performs slbmfee instruction for $LPCR_{UPRT}$ = 1 with invalid entry (V=0)		
P1 slbmfee	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 $LPCR_{UPRT}$ = 1 Value of index is 0, 1, 2, or 3 The source SLB entry, the source RB register and the target RT register are initialized to different values. The source SLB entry is initialized as invalid (V=0).	The contents of the target RT register is set to zero.
III.5.StorageCont.8 Processor P1 successfully performs slbfee instruction for $LPCR_{UPRT}$ = 0		
P1 slbfee	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 $LPCR_{UPRT}$ = 0 One valid SLB entry is initialized to translate the Effective Address specified by the RB register.	The contents of the target RT register is updated.
III.5.StorageCont.9 Processor P1 successfully performs slbfee instruction for $LPCR_{UPRT}$ = 0		

Instruction sequence	Observability preconditions	Expected results
P1 slbfee	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 $LPCR_{UPRT}$ = 0 There is no valid SLB entry that translates the Effective Address specified by the RB register.	The contents of the target RT register is set to zero.
III.5.StorageCont.10 Processor P1 successfully performs tlbie instruction IS=0 (Invalidate VA), RIC=0, PRS=0 (partition-scoped), R=0 (HPT translations)		
P1 tlbie IS=0, RIC=0, PRS=0, R=0	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 The V bit of the partition-scoped TLB entries matching the LPID and the Virtual Address specified by the RS and RB registers are initialized to 1.	All valid partition-scoped TLB entries in all threads of P1 matching the LPID and the Virtual Address specified by the RS and RB registers are invalidated.
III.5.StorageCont.10a Processor P1 successfully performs tlbie instruction IS=0 (Invalidate VA), RIC=0, PRS=0 (partition-scoped), R=1 (Radix Tree translations)		
P1 tlbie IS=0, RIC=0, PRS=0, R=1	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 The V bit of the partition-scoped TLB entries matching the LPID and the Virtual Address specified by the RS and RB registers are initialized to 1.	All valid partition-scoped TLB entries in all threads of P1 matching the LPID and the Virtual Address specified by the RS and RB registers are invalidated.
III.5.StorageCont.10b Processor P1 successfully performs tlbie instruction IS=0 (Invalidate VA), RIC=0, PRS=1 (process-scoped), R=1 (Radix Tree translations)		
P1 tlbie IS=0, RIC=0, PRS=1, R=1	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 The V bit of the process-scoped TLB entries matching the LPID, the PID and the Virtual Address specified by the RS and RB registers are initialized to 1.	All valid process-scoped TLB entries in all threads of P1 matching the LPID, the PID and the Virtual Address specified by the RS and RB registers are invalidated.
III.5.StorageCont.10c Processor P1 successfully performs tlbie instruction IS=1 (Invalidate matching PID), RIC=0 or 2, PRS=1 (process-scoped)		
P1 tlbie IS=1, RIC=0 or 2, PRS=1	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0	All valid process-scoped TLB entries in all threads of P1 matching the LPID and the PID specified by the RS register are invalidated.

Instruction sequence	Observability preconditions	Expected results
	The V bit of the process-scoped TLB entries matching the LPID and the PID specified by the RS register are initialized to 1.	
III.5.StorageCont.10d Processor P1 successfully performs tlbie instruction IS=2 (Invalidate matching LPID), RIC=0 or 2, PRS=0 (partition-scoped), R=0 (HPT translations)		
P1 tlbie IS=2, RIC=0 or 2, PRS=0, R=0	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 The V bit of the partition-scoped TLB entries matching the LPID are initialized to 1.	All valid partition-scoped TLB entries in all threads of P1 matching the LPID are invalidated.
III.5.StorageCont.10e Processor P1 successfully performs tlbie instruction IS=2 (Invalidate matching LPID), RIC=0 or 2, PRS=1 (process-scoped)		
P1 tlbie IS=2, RIC=0 or 2, PRS=1	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 The V bit of the process-scoped TLB entries matching the LPID are initialized to 1.	All valid process-scoped TLB entries in all threads of P1 matching the LPID are invalidated.
III.5.StorageCont.10f Processor P1 successfully performs tlbie instruction IS=3 in hypervisor (Invalidate all), RIC=0 or 2, PRS=0 (partition-scoped)		
P1 tlbie IS=3, RIC=0 or 2, PRS=0	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 0 The V bit of the partition-scoped TLB entries are initialized to 1.	All valid partition-scoped TLB entries in all threads of P1 are invalidated.
III.5.StorageCont.10g Processor P1 successfully performs tlbie instruction IS=3 in hypervisor (Invalidate all), RIC=0 or 2, PRS=1 (process-scoped)		
P1 tlbie IS=3, RIC=0 or 2, PRS=1	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 0 The V bit of the process-scoped TLB entries are initialized to 1.	All valid process-scoped TLB entries in all threads of P1 are invalidated.
III.5.StorageCont.10h Processor P1 successfully performs tlbie instruction IS=3 in non-hypervisor (Invalidate matching LPID), RIC=0 or 2, PRS=1 (process-scoped), R=0 (HPT translations)		
P1 tlbie IS=3, RIC=0 or 2, PRS=1, R=0	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0	All valid process-scoped TLB entries in all threads of P1 matching the LPID are invalidated.

Instruction sequence	Observability preconditions	Expected results
	The V bit of the partition-scoped TLB entries matching the LPID are initialized to 1.	
III.5.StorageCont.10i Processor P1 successfully performs tlbie instruction IS=3 in non-hypervisor (Invalidate matching LPID), RIC=0 or 2, PRS=0 (partition-scoped), R=0 (HPT translations)		
P1 tlbie IS=3, RIC=0 or 2, PRS=0, R=0	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 The V bit of the partition-scoped TLB entries matching the LPID are initialized to 1.	All valid partition-scoped TLB entries in all threads of P1 matching the LPID are invalidated.
III.5.StorageCont.11 Processor P1 successfully performs tlbiel instruction IS=0 (Invalidate VA), RIC=0, PRS=0 (partition-scoped), R=0 (HPT translations)		
P1 tlbiel IS=0, RIC=0, PRS=0, R=0	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 The V bit of the partition-scoped TLB entries matching the LPID and the Virtual Address specified by the RS and RB registers are initialized to 1.	All valid partition-scoped TLB entries in the thread of P1 that executed the tlbiel instruction matching the LPID and the Virtual Address specified by the RS and RB registers are invalidated.
III.5.StorageCont.11a Processor P1 successfully performs tlbiel instruction IS=0 (Invalidate VA), RIC=0, PRS=0 (partition-scoped), R=1 (Radix Tree translations)		
P1 tlbiel IS=0, RIC=0, PRS=0, R=1	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 The V bit of the partition-scoped TLB entries matching the LPID and the Virtual Address specified by the RS and RB registers are initialized to 1.	All valid partition-scoped TLB entries in the thread of P1 that executed the tlbiel instruction matching the LPID and the Virtual Address specified by the RS and RB registers are invalidated.
III.5.StorageCont.11b Processor P1 successfully performs tlbiel instruction IS=0 (Invalidate VA), RIC=0, PRS=1 (process-scoped), R=1 (Radix Tree translations)		
P1 tlbiel IS=0, RIC=0, PRS=1, R=1	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 The V bit of the process-scoped TLB entries matching the LPID, the PID and the Virtual Address specified by the RS and RB registers are initialized to 1.	All valid process-scoped TLB entries in the thread of P1 that executed the tlbiel instruction matching the LPID, the PID and the Virtual Address specified by the RS and RB registers are invalidated.
III.5.StorageCont.11c Processor P1 successfully performs tlbiel instruction IS=1 (Invalidate matching PID), RIC=0 or 2, PRS=1 (process-scoped)		
P1 tlbiel IS=1, RIC=0 or 2, PRS=1	MSR[DR] = 1 MSR[IR] = 1	All valid process-scoped TLB entries in the thread of P1 that executed the tlbiel

Instruction sequence	Observability preconditions	Expected results
	MSR[HV] = 0 MSR[PR] = 0 The V bit of the process-scoped TLB entries matching the LPID and the PID specified by the RS register are initialized to 1.	instruction matching the LPID and the PID specified by the RS register are invalidated.
III.5.StorageCont.11d Processor P1 successfully performs tlbiel instruction IS=2 (Invalidate matching LPID), RIC=0 or 2, PRS=0 (partition-scoped), R=0 (HPT translations)		
P1 tlbiel IS=2, RIC=0 or 2, PRS=0, R=0	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 The V bit of the partition-scoped TLB entries matching the LPID are initialized to 1.	All valid partition-scoped TLB entries in the thread of P1 that executed the tlbiel instruction matching the LPID are invalidated.
III.5.StorageCont.11e Processor P1 successfully performs tlbiel instruction IS=2 (Invalidate matching LPID), RIC=0 or 2, PRS=1 (process-scoped)		
P1 tlbiel IS=2, RIC=0 or 2, PRS=1	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 The V bit of the process-scoped TLB entries matching the LPID are initialized to 1.	All valid process-scoped TLB entries in the thread of P1 that executed the tlbiel instruction matching the LPID are invalidated.
III.5.StorageCont.11f Processor P1 successfully performs tlbiel instruction IS=3 in hypervisor (Invalidate all), RIC=0 or 2, PRS=0 (partition-scoped)		
P1 tlbiel IS=3, RIC=0 or 2, PRS=0	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 0 The V bit of the partition-scoped TLB entries are initialized to 1.	All valid partition-scoped TLB entries in the thread of P1 that executed the tlbiel instruction are invalidated.
III.5.StorageCont.11g Processor P1 successfully performs tlbiel instruction IS=3 in hypervisor (Invalidate all), RIC=0 or 2, PRS=1 (process-scoped)		
P1 tlbiel IS=3, RIC=0 or 2, PRS=1	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 1 MSR[PR] = 0 The V bit of the process-scoped TLB entries are initialized to 1.	All valid process-scoped TLB entries in the thread of P1 that executed the tlbiel instruction are invalidated.
III.5.StorageCont.11h Processor P1 successfully performs tlbiel instruction IS=3 in non-hypervisor (Invalidate matching LPID), RIC=0 or 2, PRS=1 (process-scoped), R=0 (HPT translations)		
P1 tlbiel IS=3, RIC=0 or 2, PRS=1, R=0	MSR[DR] = 1 MSR[IR] = 1	All valid process-scoped TLB entries in the thread of P1 that executed the tlbiel instruction matching the LPID are invalidated.

Instruction sequence	Observability preconditions	Expected results
	MSR[HV] = 0 MSR[PR] = 0 The V bit of the process-scoped TLB entries matching the LPID are initialized to 1.	
III.5.StorageCont.11i Processor P1 successfully performs tlbiel instruction IS=3 in non-hypervisor (Invalidate matching LPID), RIC=0 or 2, PRS=0 (partition-scoped), R=0 (HPT translations)		
P1 tlbiel IS=3, RIC=0 or 2, PRS=0, R=0	MSR[DR] = 1 MSR[IR] = 1 MSR[HV] = 0 MSR[PR] = 0 The V bit of the partition-scoped TLB entries matching the LPID are initialized to 1.	All valid partition-scoped TLB entries in the thread of P1 that executed the tlbiel instruction matching the LPID are invalidated.

18.8. Page Table Update Synchronization Requirements

Architecture sections:

- III.5.10 Translation Table Update Synchronization Requirements

Note: For the following scenarios, if the Observability condition *The base page size is equal to the actual page size for the virtual page* does not hold, the test can still be done but with multiple **tlbie** instructions within the Instruction sequence, one for each PTE corresponding to the virtual page.

Note: In the III.5.StorageCont.4 scenario, A *lbarx/stbcx*., *lharx/sthcx*., or *lwarx/stwcx*. pair (specifying the low-order byte, halfword, or word respectively of doubleword 0 of the PTE) can be used instead of the *ldarx /stdcx*. pair.

Instruction sequence	Observability preconditions	Expected results
III.5.StorageCont.1 Processor P1 successfully performs Adding a Page Table Entry		
P1 $PTE_{pp\ key\ B\ ARPN\ LP\ key\ R\ C\ WIMG\ N\ pp}$ ← new values eieio $PTE_{AVA,SW,L,H,V}$ ← new values (V=1) ptesync	V = 0 The base page size is equal to the actual page size for the virtual page MSR[DR] = 1 MSR[IR] = 1	V = 1 The virtual address translated by the updated entry will use the correct real address and associated attributes
III.5.StorageCont.2 Processor P1 successfully performs Modifying a Page Table Entry in General Case (more than one change to be done in the PTE)		
P1 r6 ← $PTE_{V,L,SW,RPN,R,C,Att,EAA}$ r4 ← addr(PTE) loop: lqarx r2,0,r4	The base page size is equal to the actual page size for the virtual page MSR[DR] = 1 MSR[IR] = 1 V = 1	V = 1 The translation instantiated by the old entry is no longer available The virtual address translated by the new entry will use the correct real address and associated attributes

Instruction sequence	Observability preconditions	Expected results
if V=0 abort, else stqcx r6,0,r4 bne- loop ptesync tlbie eieio tlbsync ptesync		
III.5.StorageCont.2a Processor P1 successfully performs Modifying a Page Table Entry in General Case (more than one change to be done in the PTE) for non-atomic hardware updates		
P1 $PTE_V \leftarrow 0$ ptesync tlbie eieio tlbsync ptesync $PTE_{ARPN,LP,AC,R,C,WIMG,N,PP} \leftarrow$ new values eieio $PTE_{B,AVA,SW,L,H,V} \leftarrow$ new values (V=1) ptesync	The base page size is equal to the actual page size for the virtual page MSR[DR] = 1 MSR[IR] = 1 V = 1	V = 1 The translation instantiated by the old entry is no longer available The virtual address translated by the new entry will use the correct real address and associated attributes
III.5.StorageCont.2b Processor P1 successfully performs Modifying a Segment Table Entry		
P1 $STE_V \leftarrow 0$ ptesync $slbieg_{old\_B,old\_ESID,old\_TA,old\_PID,old\_LPID}$ eieio slbsync ptesync $STE_{VSID,\ Ks,\ Kp,\ N,\ L,\ C,\ LP,\ SW} \leftarrow$ new values eieio $STE_{ESID,V} \leftarrow$ new values (V=1) ptesync	The base page size is equal to the actual page size for the virtual page MSR[DR] = 1 MSR[IR] = 1 V = 1	V = 1 The translation instantiated by the old entry is no longer available The effective address translated by the new entry will use the correct virtual address and associated attributes
III.5.StorageCont.3 Processor P1 successfully performs Modifying a Page Table Entry when the only change is to set the reference bit to 0		
P1 $oldR \leftarrow PTE_R$ if oldR = 1 then	The base page size is equal to the actual page size for the virtual page MSR[DR] = 1 MSR[IR] = 1	The new $PTE_R = 0$

Instruction sequence	Observability preconditions	Expected results
PTE_R ← 0 tlbie eieio tlbsync ptesync	V = 1	
III.5.StorageCont.4 Processor P1 successfully performs Modifying a Page Table Entry when the only change is to modify the SW field		
P1 loop: ldarx r1 ← PTE_dwd_0 if V=0 abort, else $r1_{57:60}$ ← new SW value stdcx. PTE_dwd_0 ← r1 bne- loop	The base page size is equal to the actual page size for the virtual page MSR[DR] = 1 MSR[IR] = 1 V = 1	The new SW field is set correctly
III.5.StorageCont.5 Processor P1 successfully performs Modifying the Effective Address (STE)		
P1 $STE_{ESID,V}$ ← new values (V=1) ptesync $slbieg_{old\_B,old\_ESID,old\_TA,old\_PID,old\_LPID}$ eieio slbsync ptesync	The base page size is equal to the actual page size for the virtual page MSR[DR] = 1 MSR[IR] = 1 V = 1	V = 1 The translation instantiated by the old entry is no longer available The effective address translated by the new entry will use the correct virtual address and associated attributes
III.5.StorageCont.6 Processor P1 successfully performs Deleting a Page Table Entry		
P1 PTE_V ← 0 ptesync tlbie eieio tlbsync ptesync	The base page size is equal to the actual page size for the virtual page MSR[DR] = 1 MSR[IR] = 1 V = 1	V = 0 The translation instantiated by an existing entry is no longer available

19. Interrupts (Chapter III.6)

Table of Contents

19.1. Interrupt registers

Architecture sections:

- III.6.2 Interrupt Registers

Scenario groups:

- FSCR Facility Enable (FE) field
- HFSCR Facility Enable (FE) field

19.1.1. FSCR Facility Enable (FE)

Facilities controlled by FSCR $_{FE}$: scv instruction, Target Address Register (TAR), Event-Based Branch Facility (EBB), Data Stream Control Register at SPR 3 (DSCR)

Guideline: each scenario in this group should be tested for each of the *facilities controlled by FSCR $_{FE}$* . F_{test} denotes the facility that is tested

Instruction sequence	Compliance conditions	Expected result
III.6.2.FSCR.1 When the FSCR makes a facility unavailable, attempted execution of an instruction in problem state causes a Facility Unavailable interrupt		
Any instruction of the F_{test} facility	The program is in problem state The bit in $FSCR_{FE}$ that controls F_{test} equals 0	A Facility Unavailable interrupt occurs
III.6.2.FSCR.2 When the FSCR makes a facility unavailable, attempted access of an SPR using mfspr/mtspr in problem state causes a Facility Unavailable interrupt		
Representative of: mtspr mfspr	The designated SPR is in the F_{test} facility The program is in problem state The bit in $FSCR_{FE}$ that controls F_{test} equals 0	A Facility Unavailable interrupt occurs
III.6.2.FSCR.3 When the FSCR makes a facility unavailable, rfebb, rfid, rfscv, hrfid and mtmsr[d] instructions have the same effect on bits in system registers as they would if the bits were available		
Representative of: rfebb rfid rfscv hrfid mtmsr[d]	The designated SPR is in the F_{test} facility The program is in problem state The bit in $FSCR_{FE}$ that controls F_{test} equals 0	Bits in the designated SPR are set correctly
III.6.2.FSCR.4 When the FSCR makes a facility available, instructions are available in problem state		

Instruction sequence	Compliance conditions	Expected result
Any instruction of the F_{test} facility	The program is in problem state The bit in $FSCR_{FE}$ that controls F_{test} equals 1 F_{test} is not made unavailable by another register	Correct instruction behavior
III.6.2.FSCR.5 When the FSCR makes a facility available, access to SPRs using mfspr/mtspr is available in problem state		
Representative of: mtspr mfspr	The designated SPR is in the F_{test} facility The program is in problem state The bit in $FSCR_{FE}$ that controls F_{test} equals 1 F_{test} is not made unavailable by another register	Correct instruction behavior
III.6.2.FSCR.6 When the FSCR makes a facility unavailable, instructions are available when not in problem state		
Any instruction of the F_{test} facility	The program is not in problem state The bit in $FSCR_{FE}$ that controls F_{test} equals 0 F_{test} is not made unavailable by another register	Correct instruction behavior
III.6.2.FSCR.7 When the FSCR makes a facility available, access to SPRs using mfspr/mtspr is available when not in problem state		
Representative of: mtspr mfspr	The designated SPR is in the F_{test} facility The program is not in problem state The bit in $FSCR_{FE}$ that controls F_{test} equals 0 F_{test} is not made unavailable by another register	Correct instruction behavior

19.1.2. HFSCR Facility Enable (FE)

Facilities controlled by $HFSCR_{FE}$: msgsndp and msgclrp instructions, Target Address Register (TAR), Event-Based Branch Facility (EBB), Transactional Memory (TM), BHRB Instructions (BHRB), Performance Monitor Facility SPRs (PM), Data Stream Control Register (DSCR), Vector and VSX Facilities (VECVSX), Floating Point Facility (FP),

Guideline: each scenario in this group should be tested for each of the *facilities controlled by* $HFSCR_{FE}$. F_{test} denotes the facility that is tested

Instruction sequence	Compliance conditions	Expected result
III.6.2.HFSCR.1 When the HFSCR makes a facility unavailable, attempted execution of an instruction in problem or privileged non-hypervisor states causes a Hypervisor Facility Unavailable interrupt		
Any instruction of the F_{test} facility	The program is in problem state or privileged non-hypervisor state The bit in $HFSCR_{FE}$ that controls F_{test} equals 0	A Hypervisor Facility Unavailable interrupt occurs
III.6.2.HFSCR.2 When the HFSCR makes a facility unavailable, attempted access of an SPR using mfspr/mtspr in problem or privileged non-hypervisor states causes a Hypervisor Facility Unavailable interrupt		
Representative of:	The designated SPR is in the F_{test} facility	A Hypervisor Facility Unavailable interrupt occurs

Instruction sequence	Compliance conditions	Expected result
mtspr mfspr	The program is in problem state or privileged non-hypervisor state The bit in $HFSCR_{FE}$ that controls F_{test} equals 0	
III.6.2.HFSCR.3 When the HFSCR makes a facility unavailable, rfebb, rfid, rfscv, hrfid and mtmsr[d] instructions have the same effect on bits in system registers as they would if the bits were available		
Representative of: rfebb rfid rfscv hrfid mtmsr[d]	The designated SPR is in the F_{test} facility The program is in problem state or privileged non-hypervisor state The bit in $HFSCR_{FE}$ that controls F_{test} equals 0	Bits in the designated SPR are set correctly
III.6.2.HFSCR.4 When the HFSCR makes a facility available, instructions are available in problem or privileged non-hypervisor states		
Any instruction of the F_{test} facility	The program is in problem state or privileged non-hypervisor state The bit in $HFSCR_{FE}$ that controls F_{test} equals 1 F_{test} is not made unavailable by another register	Correct instruction behavior
III.6.2.HFSCR.5 When the HFSCR makes a facility available, access to SPRs using mfspr/mtspr is available in problem or privileged non-hypervisor states		
Representative of: mtspr mfspr	The designated SPR is in the F_{test} facility The program is in problem state or privileged non-hypervisor state The bit in $HFSCR_{FE}$ that controls F_{test} equals 1 F_{test} is not made unavailable by another register	Correct instruction behavior
III.6.2.HFSCR.6 When the HFSCR makes a facility unavailable, instructions are available in hypervisor state		
Any instruction of the F_{test} facility	The program is in hypervisor state The bit in $HFSCR_{FE}$ that controls F_{test} equals 0 F_{test} is not made unavailable by another register	Correct instruction behavior
III.6.2.HFSCR.7 When the HFSCR makes a facility available, access to SPRs using mfspr/mtspr is available in hypervisor state		
Representative of: mtspr mfspr	The designated SPR is in the F_{test} facility The program is in hypervisor state The bit in $HFSCR_{FE}$ that controls F_{test} equals 0 F_{test} is not made unavailable by another register	Correct instruction behavior

19.2. Interrupt definitions

Architecture sections:

- III.6.5 Interrupt Definitions
- III.2.2 Logical Partitioning Control Register (LPCR)

All interrupt types: System Reset, Machine Check, Data Storage, Data Segment, Instruction Storage, Instruction Segment, External, Alignment, Program, FP Unavailable, Decrementer, Directed Privileged Doorbell, Hypervisor Decrementer, System Call, Trace, Hypervisor Data Storage, Hypervisor Instruction Storage, Hypervisor Emulation Assistance, Hypervisor Maintenance, Directed Hypervisor Doorbell, Hypervisor Virtualization, Performance Monitor, Vector Unavailable, VSX Unavailable, Facility Unavailable, Hypervisor Facility Unavailable, System Call Vectored

19.2.1. Setting MSR bits

Architecture sections:

- III.6.5 Interrupt Definitions, Figure 65 MSR setting due to interrupt

Scenario groups:

- Setting MSR bits that are uniform for all interrupts
- Setting ME
- Setting HV and RI
- Setting IR and DR
- Setting LE
- Setting TS

19.2.1.1. Setting MSR bits that are uniform for all interrupts

MSR bits set to 0 by all interrupts: FE0, FE1, EE, FP, PR, TM, VEC, VSX, PMM, bit 5, reserved bits

Guideline: the scenario in this group should be tested for a representative interrupt I_{test} selected from *all interrupt types*.

Instruction sequence	Compliance conditions	Expected result
III.6.5.Uniform.1 Setting MSR bits when an interrupt of type I_{test} occurs		
Any instruction that may cause I_{test} to occur	An interrupt of type I_{test} occurs	All of the *MSR bits set to 0 by all interrupts* equal 0 SF = 1

19.2.1.2. Setting ME

Instruction sequence	Compliance conditions	Expected result
III.6.5.Uniform.2 Interrupts other than Machine Check and System Reset do not change ME		
Any instruction that may cause an interrupt	A representative interrupt that is not Machine Check or System Reset occurs	ME unchanged
III.6.5.Uniform.3 Machine Check interrupts set ME to 0		
(No instruction sequence specified)	A Machine Check interrupt occurs	ME = 0
III.6.5.Uniform.4 Setting ME when a System Reset interrupt occurs while the thread is in power-saving mode		
(No instruction sequence specified)	A System Reset interrupt occurs The thread is in power-saving mode	ME = 1
III.6.5.Uniform.5 Setting ME when a System Reset interrupt occurs while the thread is **not** in power-saving mode		
(No instruction sequence specified)	A System Reset interrupt occurs The thread is **not** in power-saving mode	ME unchanged

19.2.1.3. Setting HV and RI

Interrupts that set HV to 1 and RI to 0: System Reset, Machine Check

Interrupts that set HV to 1 and dont change RI: Hypervisor Decrementer, Hypervisor Data Storage, Hypervisor Instruction Storage, Hypervisor Emulation Assistance, Hypervisor Maintenance, Directed Hypervisor Doorbell, Hypervisor Virtualization, Hypervisor Facility Unavailable

Instruction sequence	Compliance conditions	Expected result
III.6.5.HV.1 Setting RI to 0 and leaving HV unchanged		
Any instruction that may cause an interrupt	A representative interrupt that is **not** in *Interrupts that set HV to 1 and RI to 0* and **not** in *Interrupts that set HV to 1 and dont change RI* occurs	HV unchanged RI = 0
III.6.5.HV.2 Setting HV to 1 and RI to 0		
Any instruction that may cause an interrupt	A representative interrupt that is in *Interrupts that set HV to 1 and RI to 0* occurs	HV = 1 RI = 0
III.6.5.HV.3 Setting HV to 1 and leaving RI unchanged		
Any instruction that may cause an interrupt	A representative interrupt that is in *Interrupts that set HV to 1 and dont change RI* occurs	HV = 1 RI unchanged
III.6.5.HV.4 Setting HV and RI when an External interrupt occurs and $LPCR_{LPES}$ = 0		
Any instruction that may cause an interrupt	An External interrupt occurs $LPCR_{LPES}$ = 0	HV = 1 RI unchanged
III.6.5.HV.5 Setting HV and RI when an External interrupt occurs and $LPCR_{LPES}$ = 1		
Any instruction that may cause an interrupt	An External interrupt occurs $LPCR_{LPES}$ = 1	HV unchanged RI = 0
III.6.5.HV.6 Setting HV and RI when a System Call interrupt occurs with LEV = 1		
sc	LEV = 1 A System Call interrupt occurs	HV = 1 RI = 0
III.6.5.HV.7 Setting HV and RI when a System Call interrupt occurs with LEV = 0		
sc	LEV = 0 A System Call interrupt occurs	HV unchanged RI = 0

19.2.1.4. Setting IR and DR

Interrupts that always set IR and DR to zero: System Reset, Machine Check, Hypervisor Maintenance

Conditions for setting IR and DR to 1:

$LPCR_{AIL}$ = 2 or $LPCR_{AIL}$ = 3

$MSR_{IR\ DR}$ = 0b11

Instruction sequence	Compliance conditions	Expected result
III.6.5.IR.1 Setting IR and DR when an interrupt in *Interrupts that always set IR and DR to zero* occurs		
Any instruction that may cause an interrupt	A representative interrupt that is in *Interrupts that always set IR and DR to zero* occurs	IR = 0 DR = 0

Instruction sequence	Compliance conditions	Expected result
III.6.5.IR.2 Setting IR and DR when an interrupt that is **not** in *Interrupts that always set IR and DR to zero* occurs, when conditions for setting IR and DR to 1 are satisfied		
Any instruction that may cause an interrupt	A representative interrupt that is **not** in *Interrupts that always set IR and DR to zero* occurs All of the *conditions for setting IR and DR to 1* are satisfied	IR = 1 DR = 1
III.6.5.IR.3 Setting IR and DR when an interrupt that is **not** in *Interrupts that always set IR and DR to zero* occurs, when conditions for setting IR and DR to 1 are **not** satisfied		
Any instruction that may cause an interrupt	A representative interrupt that is **not** in *Interrupts that always set IR and DR to zero* occurs At least one of the *conditions for setting IR and DR to 1* is **not** satisfied	IR = 0 DR = 0

19.2.1.5. Setting LE

Instruction sequence	Compliance conditions	Expected result
III.6.5.LE.1 If an interrupt results in HV being equal to 1, the LE bit is copied from the HILE bit		
Any instruction that may cause an interrupt	An interrupt that results in HV = 1 occurs	LE = HILE
III.6.5.LE.2 If an interrupt results in HV being equal to 0, the LE bit is copied from the $LPCR_{ILE}$ bit.		
Any instruction that may cause an interrupt	An interrupt that results in HV = 0 occurs	LE = $LPCR_{ILE}$

19.2.1.6. Setting TS

Instruction sequence	Compliance conditions	Expected result
III.6.5.TS.1 If the TS field contained 0b10 (Transactional) when the interrupt occurred, the TS field is set to 0b01 (Suspended)		
Any instruction that may cause an interrupt	An interrupt occurs TS = 0b10 before the interrupt occurs	TS = 0b01
III.6.5.TS.2 If the TS field did **not** contain 0b10 (Transactional) when the interrupt occurred, the TS field is not altered.		
Any instruction that may cause an interrupt	An interrupt occurs TS != 0b10 before the interrupt occurs	TS unchanged

19.2.2. Effective address of interrupt vector

Architecture sections:

- III.6.5 Interrupt Definitions, Figure 66 Effective address of interrupt vector by interrupt type
- III.2.2 Logical Partitioning Control Register (LPCR)

Interrupts that are not controlled by $LPCR_{AIL}$: Machine Check, System Reset, Hypervisor Maintenance

Instruction sequence	Compliance conditions	Expected result
III.6.5.Vector.1 When $LPCR_{AIL}$ = 0, execution resumes at the effective address of the interrupt vector corresponding to the interrupt type, with no offset		
Any instruction that may cause an interrupt	A representative interrupt of type I_{test} that is **not** in *Interrupts that are not controlled by* $LPCR_{AIL}$ occurs	Execution resumes at the effective address specified for I_{test} in Figure 66

Instruction sequence	Compliance conditions	Expected result
	$LPCR_{AIL}$ = 0	No offset is applied
III.6.5.Vector.2 When $LPCR_{AIL}$ = 2, execution resumes at the effective address of the interrupt vector corresponding to the interrupt type, with an offset of 0x0000_0000_0001_8000		
Any instruction that may cause an interrupt	A representative interrupt of type I_{test} that is **not** in *Interrupts that are not controlled by* $LPCR_{AIL}$ occurs $LPCR_{AIL}$ = 2	Execution resumes at the effective address specified for I_{test} in Figure 66 An offset of 0x0000_0000_0001_8000 is applied
III.6.5.Vector.3 When $LPCR_{AIL}$ = 3, execution resumes at the effective address of the interrupt vector corresponding to the interrupt type (not System Call Vectored), with an offset of 0xC000_0000_0000_4000		
Any instruction that may cause an interrupt	A representative interrupt of type I_{test} that is **not** in *Interrupts that are not controlled by* $LPCR_{AIL}$ occurs $LPCR_{AIL}$ = 3	Execution resumes at the effective address specified for I_{test} in Figure 66 An offset of 0xC000_0000_0000_4000 is applied
III.6.5.Vector.3b When $LPCR_{AIL}$ = 3, execution resumes at the effective address of the interrupt vector corresponding to the interrupt type (System Call Vectored), with an offset of 0x0000_0000_3 || LEV || 0_4000		
scv	A representative interrupt of type I_{test} that is **not** in *Interrupts that are not controlled by* $LPCR_{AIL}$ occurs $LPCR_{AIL}$ = 3	Execution resumes at the effective address specified for I_{test} in Figure 66 An offset of 0xC000_0000_3 || LEV || 0_4000 is applied
III.6.5.Vector.4 Interrupts that cause a transition from MSR_{HV} = 0 to MSR_{HV} = 1 are always taken as if $LPCR_{AIL}$ = 0		
Any instruction that may cause an interrupt	A representative interrupt of type I_{test} that is **not** in *Interrupts that are not controlled by* $LPCR_{AIL}$ occurs MSR_{HV} = 0 before the interrupt occurs MSR_{HV} = 1 is set as a result of the interrupt $LPCR_{AIL}$!= 0	Execution resumes at the effective address specified for I_{test} in Figure 66 No offset is applied
III.6.5.Vector.5 Interrupts that occur when MSR_{IR} = 0 or MSR_{DR} = 0 are always taken as if $LPCR_{AIL}$ = 0		
Any instruction that may cause an interrupt	A representative interrupt of type I_{test} that is **not** in *Interrupts that are not controlled by* $LPCR_{AIL}$ occurs MSR_{IR} = 0 or MSR_{DR} = 0 before the interrupt occurs $LPCR_{AIL}$!= 0	Execution resumes at the effective address specified for I_{test} in Figure 66 No offset is applied
III.6.5.Vector.6 For interrupts not controlled by $LPCR_{AIL}$, the effective address offset is not applied		
Any instruction that may cause an interrupt	A representative interrupt of type I_{test} , that is in *Interrupts that are not controlled by* $LPCR_{AIL}$, occurs $LPCR_{AIL}$!= 0	Execution resumes at the effective address specified for I_{test} in Figure 66 No offset is applied

19.2.3. System Reset Interrupt

Architecture sections:

- III.6.5.1 System Reset Interrupt

Scenario groups:

- Conditions for occurrence of a System Reset interrupt
- Actions taken when a System Reset interrupt occurs

Exceptions potentially causing a System Reset interrupt: System Reset, External, Decrementer, Directed Privileged Doorbell, Directed Hypervisor Doorbell, Hypervisor Maintenance, Hypervisor Virtualization exception

Note: this is a system-caused interrupt, not directly caused by the execution of an instruction

19.2.3.1. Conditions for occurrence of a System Reset interrupt

Instruction sequence	Compliance conditions	Expected result
III.6.5.1.Cond.1 Occurrence of a System Reset interrupt when a System Reset exception exists		
(No instruction sequence specified)	A System Reset exception exists	A System Reset interrupt occurs
III.6.5.1.Cond.2 Occurrence of a System Reset interrupt when the thread is in power-saving mode and any of the specified exceptions exist		
(No instruction sequence specified)	The thread is in power-saving mode (At least) one of the exceptions in *Exceptions potentially causing a System Reset interrupt* exists The exception is enabled by its corresponding bit in $LPCR_{PECE}$ to cause exit from power saving mode	A System Reset interrupt occurs
III.6.5.1.Cond.3 System Reset interrupt does not occur when the thread is in power-saving mode and any of the specified exceptions exist, but are disabled by the LPCR		
(No instruction sequence specified)	The thread is in power-saving mode (At least) one of the exceptions in *Exceptions potentially causing a System Reset interrupt* exists The exception is **disabled** by its corresponding bit in $LPCR_{PECE}$ to cause exit from power saving mode	No System Reset interrupt occurs

19.2.3.2. Actions taken when a System Reset interrupt occurs

Conditions for non-recoverable System Reset interrupt: if **any** of the following conditions hold, the interrupt is non-recoverable

- The interrupt is not context synchronizing
- The interrupt causes the loss of a Machine Check exception or a Direct External exception
- The state of the thread has been corrupted

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250]. Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.1.Act.1 SRR0 is set correctly when a System Reset interrupt occurs when the thread is **not** in power-saving mode		
(No instruction sequence specified)	A System Reset interrupt occurs The thread is **not** in power-saving mode when the interrupt occurs	SRR0 = the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present
III.6.5.1.Act.2 Bits 42:45 of SRR1 are set correctly when a System Reset interrupt occurs when the thread is in power-saving mode		
(No instruction sequence specified)	A System Reset interrupt occurs One of the exceptions in *Exceptions potentially causing a System Reset interrupt* exists	$SRR1_{42:45}$ = correct value indicating the exception that caused exit from power-saving mode, as detailed in Section III.6.5.1

Instruction sequence	Compliance conditions	Expected result
	The thread is in power-saving mode when the interrupt occurs	
III.6.5.1.Act.3 Bits 42:45 of SRR1 are set correctly when a System Reset interrupt occurs when the thread is in power-saving mode and multiple exceptions that cause exit from power-saving mode exist		
(No instruction sequence specified)	A System Reset interrupt occurs At least two of the exceptions in *Exceptions potentially causing a System Reset interrupt* exist The thread is in power-saving mode when the interrupt occurs	$SRR1_{42:45}$ = correct value indicating the exception that caused exit from power-saving mode, as detailed in Section III.6.5.1 The exception reported is the exception corresponding to the interrupt that would have occurred if the same exceptions existed and the thread was not in power-saving mode
III.6.5.1.Act.4 Bits 46:47 of SRR1 are set correctly when a System Reset interrupt occurs when the thread is **not** in power-saving mode		
(No instruction sequence specified)	A System Reset interrupt occurs The thread is **not** in power-saving mode when the interrupt occurs	$SRR1_{46:47}$ = 00
III.6.5.1.Act.5 Bits 46:47 of SRR1 are set correctly when a System Reset interrupt occurs when the thread is in power-saving mode and the state of all resources is maintained		
(No instruction sequence specified)	A System Reset interrupt occurs The thread is in power-saving mode when the interrupt occurs The state of all resources is maintained as if the thread is not in power-saving mode	$SRR1_{46:47}$ = 01
III.6.5.1.Act.6 Bits 46:47 of SRR1 are set correctly when a System Reset interrupt occurs when the thread is in power-saving mode and the state of all hypervisor resources is maintained		
(No instruction sequence specified)	A System Reset interrupt occurs The thread is in power-saving mode when the interrupt occurs The state of some resources was not maintained, but the state of all hypervisor resources was maintained as if the thread was not in power-saving mode and the state of all other resources is such that the hypervisor can resume execution	$SRR1_{46:47}$ = 10
III.6.5.1.Act.7 Bits 46:47 of SRR1 are set correctly when a System Reset interrupt occurs when the thread is in power-saving mode and the state of some resources is such that the hypervisor cannot resume execution		
(No instruction sequence specified)	A System Reset interrupt occurs The thread is in power-saving mode when the interrupt occurs The state of some resources was not maintained, and the state of some hypervisor resources was not maintained or the state of some resources is such that the hypervisor cannot resume execution	$SRR1_{46:47}$ = 11
III.6.5.1.Act.8 Bit 62 of SRR1 is set correctly when a System Reset interrupt occurs when the thread is **not** in power-saving mode and the thread is in a recoverable state		
(No instruction sequence specified)	A System Reset interrupt occurs The thread is **not** in power-saving mode when the interrupt occurs	$SRR1_{62}$ = MSR_{62}

Instruction sequence	Compliance conditions	Expected result
	None of the *Conditions for non-recoverable System Reset interrupt* hold	
III.6.5.1.Act.9 Bit 62 of SRR1 is set correctly when a System Reset interrupt occurs when the thread is **not** in power-saving mode and **not** in a recoverable state		
(No instruction sequence specified)	A System Reset interrupt occurs The thread is **not** in power-saving mode when the interrupt occurs At least one of the *Conditions for non-recoverable System Reset interrupt* holds	$SRR1_{62}$ = 0
III.6.5.1.Act.10 Bit 62 of SRR1 is set correctly when a System Reset interrupt occurs when the thread is in power-saving mode and is in a recoverable state		
(No instruction sequence specified)	A System Reset interrupt occurs The thread is in power-saving mode when the interrupt occurs None of the *Conditions for non-recoverable System Reset interrupt* hold	$SRR1_{62}$ = 1
III.6.5.1.Act.11 Bit 62 of SRR1 is set correctly when a System Reset interrupt occurs when the thread is in power-saving mode and is **not** in a recoverable state		
(No instruction sequence specified)	A System Reset interrupt occurs The thread is in power-saving mode when the interrupt occurs At least one of the *Conditions for non-recoverable System Reset interrupt* holds	$SRR1_{62}$ = 0
III.6.5.1.Act.12 Remaining bits of SRR1 are set correctly when a System Reset interrupt occurs		
(No instruction sequence specified)	A System Reset interrupt occurs	$SRR1_{34:36}$ = 0 SRR1 bits 0:33, 37:41, 48:61, 63 = corresponding bits in the MSR
III.6.5.1.Act.13 If a System Reset interrupt occurs when the thread is in power-saving mode and is caused by an exception other than a System Reset exception, all other registers, except HSRR0 and HSRR1, that would be set by the corresponding interrupt if the exception occurred when the thread was not in power-saving mode are set by the System Reset interrupt		
(No instruction sequence specified)	A System Reset interrupt occurs The thread is in power-saving mode when the interrupt occurs The interrupt is caused by one of the *Exceptions potentially causing a System Reset interrupt*, other than a System Reset exception	All registers except SRR0, SRR1, HSRR0 and HSRR1, that would be set by the corresponding interrupt if the exception occurred when the thread was not in power-saving mode, are set to the values to which they would be set if the exception occurred when the thread was not in power-saving mode

19.2.4. Machine Check Interrupt

Architecture sections:

- III.6.5.2 Machine Check Interrupt

Scenario groups:

- Conditions for occurrence of a Machine Check interrupt
- Actions taken when a Machine Check interrupt occurs

Note: this is a system-caused interrupt, not directly caused by the execution of an instruction

19.2.4.1. Conditions for occurrence of a Machine Check interrupt

Instruction sequence	Compliance conditions	Expected result
III.6.5.2.Cond.1 Occurrence of a Machine Check interrupt when the thread is **not** in power-saving mode and Machine Check interrupts are enabled by the MSR		
(No instruction sequence specified)	A Machine Check exception exists The thread is **not** in power-saving mode when the interrupt occurs $MSR_{ME} = 1$	A Machine Check interrupt occurs
III.6.5.2.Cond.2 Entering Checkstop state when the thread is **not** in power-saving mode and Machine Check interrupts are disabled by the MSR		
(No instruction sequence specified)	A Machine Check exception exists The thread is **not** in power-saving mode when the interrupt occurs $MSR_{ME} = 0$	The thread enters the Checkstop state
III.6.5.2.Cond.3 Occurrence of a Machine Check interrupt when the thread is in power-saving mode and Machine Check interrupts are enabled by the LPCR to exit power-saving mode		
(No instruction sequence specified)	A Machine Check exception exists The thread is in power-saving mode $LPCR_{51} = 1$	A Machine Check interrupt occurs
III.6.5.2.Cond.4 Machine Check interrupt does not occur when the thread is in power-saving mode and Machine Check interrupts are **disabled** by the LPCR to exit power-saving mode		
(No instruction sequence specified)	A Machine Check exception exists The thread is in power-saving mode $LPCR_{51} = 0$	No Machine Check interrupt occurs

19.2.4.2. Actions taken when a Machine Check interrupt occurs

Conditions for non-recoverable Machine Check interrupt: if **any** of the following conditions hold, the interrupt is non-recoverable

The interrupt is not context synchronizing

The interrupt causes the loss of a Direct External exception

The state of the thread has been corrupted

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.2.Act.1 When a thread is in Checkstop state, instruction processing is suspended		
(No instruction sequence specified)	The thread enters Checkstop state	Instruction processing is suspended
III.6.5.2.Act.2 SRR0 is set correctly when a Machine Check interrupt occurs when the thread is **not** in power-saving mode		
(No instruction sequence specified)	A Machine Check interrupt occurs The thread is **not** in power-saving mode when the interrupt occurs	SRR0 = the effective address of some instruction that was executing or was about to be executed when the Machine Check exception occurred
III.6.5.2.Act.3 Bits 46:47 of SRR1 are set correctly when a Machine Check interrupt occurs when the thread is **not** in power-saving mode		

Instruction sequence	Compliance conditions	Expected result
(No instruction sequence specified)	A Machine Check interrupt occurs The thread is **not** in power-saving mode when the interrupt occurs	$SRR1_{46:47}$ = 00
III.6.5.2.Act.4 Bits 46:47 of SRR1 are set correctly when a Machine Check interrupt occurs when the thread is in power-saving mode and the state of all resources is maintained		
(No instruction sequence specified)	A Machine Check interrupt occurs The thread is in power-saving mode when the interrupt occurs The state of all resources is maintained as if the thread is not in power-saving mode	$SRR1_{46:47}$ = 01
III.6.5.2.Act.5 Bits 46:47 of SRR1 are set correctly when a Machine Check interrupt occurs when the thread is in power-saving mode and the state of all hypervisor resources is maintained		
(No instruction sequence specified)	A Machine Check interrupt occurs The thread is in power-saving mode when the interrupt occurs The state of some resources was not maintained, but the state of all hypervisor resources was maintained as if the thread was not in power-saving mode and the state of all other resources is such that the hypervisor can resume execution	$SRR1_{46:47}$ = 10
III.6.5.2.Act.6 Bits 46:47 of SRR1 are set correctly when a Machine Check interrupt occurs when the thread is in power-saving mode and the state of some resources is such that the hypervisor cannot resume execution		
(No instruction sequence specified)	A Machine Check interrupt occurs The thread is in power-saving mode when the interrupt occurs The state of some resources was not maintained, and the state of some hypervisor resources was not maintained or the state of some resources is such that the hypervisor cannot resume execution	$SRR1_{46:47}$ = 11
III.6.5.2.Act.7 Bit 62 of SRR1 is set correctly when a Machine Check interrupt occurs when the thread is **not** in power-saving mode and the thread is in a recoverable state		
(No instruction sequence specified)	A Machine Check interrupt occurs The thread is **not** in power-saving mode when the interrupt occurs None of the *Conditions for non-recoverable Machine Check interrupt* hold	$SRR1_{62}$ = MSR_{62}
III.6.5.2.Act.8 Bit 62 of SRR1 is set correctly when a Machine Check interrupt occurs when the thread is **not** in power-saving mode and **not** in a recoverable state		
(No instruction sequence specified)	A Machine Check interrupt occurs The thread is **not** in power-saving mode when the interrupt occurs At least one of the *Conditions for non-recoverable Machine Check interrupt* holds	$SRR1_{62}$ = 0
III.6.5.2.Act.9 Bit 62 of SRR1 is set correctly when a Machine Check interrupt occurs when the thread is in power-saving mode and is in a recoverable state		
(No instruction sequence specified)	A Machine Check interrupt occurs	$SRR1_{62}$ = 1

Instruction sequence	Compliance conditions	Expected result
	The thread is in power-saving mode when the interrupt occurs None of the *Conditions for non-recoverable Machine Check interrupt* hold	
III.6.5.2.Act.10 Bit 62 of SRR1 is set correctly when a Machine Check interrupt occurs when the thread is in power-saving mode and is **not** in a recoverable state		
(No instruction sequence specified)	A Machine Check interrupt occurs The thread is in power-saving mode when the interrupt occurs At least one of the *Conditions for non-recoverable Machine Check interrupt* holds	$SRR1_{62} = 0$

19.2.5. Data Storage Interrupt

Architecture sections:

- III.6.5.3 Data Storage Interrupt

Scenario groups:

- Conditions for occurrence of a Data Storage interrupt
- Actions taken when a Data Storage interrupt occurs

19.2.5.1. Conditions for occurrence of a Data Storage interrupt

Data Storage interrupt condition:

(a) a copy-paste transfer other than from main storage to a properly initiated accelerator is attempted, or

(b) $(MSR_{HV\ PR} = 0b10)$ & $(MSR_{DR} = 0)$), or

(c) HPT translation is being performed, the value of the expression $((MSR_{HV\ PR} = 0b10)$ | $((\neg VPM$ | $\neg PRTE_V)$ & $MSR_{DR}))$ is 1, and a data access cannot be performed, except for the case of $MSR_{HV\ PR}$ not equal to 0b10, VPM = 0, $LPCR_{KBV} = 1$, and a Virtual Storage Page Class Key Protection exception exists, or

(d) Radix Tree translation is being performed, and either a Data Address Watchpoint match occurs, an attempt is made to execute an AMO with an invalid function code, or process-scoped translation either does not complete or prevents the data access from being performed

Instructions potentially causing a Data Storage interrupt when the effective or virtual address cannot be translated to a real address: Load instructions, Store instructions, icbi dcbz dcbst dcbf[l]

Instructions potentially causing a Data Storage interrupt when referring to Write Through Required or Caching Inhibited storage: lq stq lwat ldat lbarx lharx lwarx ldarx lqarx stwat stdat stbcx. sthcx. stwcx. stdcx. stqcx.

Instructions potentially causing a Data Storage interrupt when referring to Caching Inhibited storage: copy paste.

Instructions potentially causing a Data Storage interrupt when referring to Guarded storage: lwat ldat stwat stdat

Instruction sequence	Compliance conditions	Expected result
III.6.5.3.Cond.1 Occurrence of a Data Storage interrupt when data address translation is enabled and the virtual address of any byte of the storage location specified by a Load, Store, icbi, dcbz, dcbst, or dcbf[l] instruction cannot be translated to a real address because no valid PTE was found for HPT translation with VPM off		
Representative of *Instructions potentially causing a Data Storage interrupt when the effective or virtual address cannot be translated to a real address*	The *Data Storage interrupt condition* holds MSR_{DR} = 1 $LPCR_{VPM}$ = 0 $LPCR_{HR}$ = 0 The virtual address of any byte of the storage location specified by the instruction cannot be translated to a real address No higher priority exception exists	A Data Storage interrupt occurs
III.6.5.3.Cond.1b Occurrence of a Data Storage interrupt when data address translation is enabled and the effective address of any byte of the storage location specified by a Load, Store, icbi, dcbz, dcbst, or dcbf[l] instruction cannot be translated to a real address because no valid PTE was found for the process-scoped Radix Tree translation		
Representative of *Instructions potentially causing a Data Storage interrupt when the effective or virtual address cannot be translated to a real address*	The *Data Storage interrupt condition* holds MSR_{DR} = 1 $LPCR_{VPM}$ = 0 $LPCR_{HR}$ = 1 The effective address of any byte of the storage location specified by the instruction cannot be translated to a real address No higher priority exception exists	A Data Storage interrupt occurs
III.6.5.3.Cond.2 Occurrence of a Data Storage interrupt when the effective address specified by a lq, stq, lwat, ldat, lbarx, lharx, lwarx, ldarx, lqarx, stwat, stdat, stbcx., sthcx., stwcx., stdcx., or stqcx. instruction refers to storage that is Write Through Required or Caching Inhibited		
Representative of *Instructions potentially causing a Data Storage interrupt when referring to Write Through Required or Caching Inhibited storage*	The *Data Storage interrupt condition* holds The effective address specified by the instruction refers to storage that is Write Through Required or Caching Inhibited No higher priority exception exists	A Data Storage interrupt occurs
III.6.5.3.Cond.3 Occurrence of a Data Storage interrupt when the access violates Basic Storage Protection		
Any Load or Store instruction	The *Data Storage interrupt condition* holds The access violates Basic Storage Protection No higher priority exception exists	A Data Storage interrupt occurs
III.6.5.3.Cond.4 Occurrence of a Data Storage interrupt when the access violates Virtual Page Class Key Storage Protection and $LPCR_{KBV}$ = 0		
Any Load or Store instruction	The *Data Storage interrupt condition* holds The access violates Virtual Page Class Key Storage Protection	A Data Storage interrupt occurs

Instruction sequence	Compliance conditions	Expected result
	$LPCR_{KBV}$ = 0 No higher priority exception exists	
III.6.5.3.Cond.5 Occurrence of a Data Storage interrupt when a Data Address Watchpoint match occurs		
Any Load or Store instruction	The *Data Storage interrupt condition* holds A Data Address Watchpoint match occurs No higher priority exception exists	A Data Storage interrupt occurs
III.6.5.3.Cond.6 (removed)		
(removed)	(removed)	removed)
III.6.5.3.Cond.7 Occurrence of a Data Storage interrupt when an attempt is made to execute a Fixed-Point Load or Store Caching Inhibited instruction with MSR_{DR} = 1		
Any Fixed-Point Load or Store Caching Inhibited instruction	The *Data Storage interrupt condition* holds MSR_{DR} = 1 No higher priority exception exists	A Data Storage interrupt occurs
III.6.5.3.Cond.8 Occurrence of a Data Storage interrupt when an attempt is made to execute a Fixed-Point Load or Store Caching Inhibited instruction specifying a storage location that is specified by the Hypervisor Real Mode Storage Control facility to be treated as non-Guarded		
Any Fixed-Point Load or Store Caching Inhibited instruction	The *Data Storage interrupt condition* holds The instruction specifies a storage location that is specified by the Hypervisor Real Mode Storage Control facility to be treated as non-Guarded No higher priority exception exists	A Data Storage interrupt occurs
III.6.5.3.Cond.9 If the XER specifies a length of zero for an indexed Move Assist instruction, a Data Storage interrupt does not occur.		
Representative of *Indexed Move Assist instructions*	$XER_{57:63}$ = 0 No higher priority exception exists	No Data Storage interrupt occurs
III.6.5.3.Cond.10 Occurrence of a Data Storage interrupt when the address of the appropriate process table entry or segment table entry group cannot be translated when HR=0 and VPM=0		
Any Load or Store instruction	The *Data Storage interrupt condition* holds $LPCR_{VPM}$ = 0 $LPCR_{HR}$ = 0 The address of the appropriate process table entry or segment table entry group cannot be translated No higher priority exception exists	A Data Storage interrupt occurs
III.6.5.3.Cond.11 Occurrence of a Data Storage interrupt when the effective address specified by a copy or paste. instruction refers to storage that is Caching Inhibited		
P1 cpabort copy paste.	The *Data Storage interrupt condition* holds The effective address specified by a copy or paste. instruction refers to storage that is Caching Inhibited No higher priority exception exists	A Data Storage interrupt occurs

Instruction sequence	Compliance conditions	Expected result
III.6.5.3.Cond.12 Occurrence of a Data Storage interrupt when the effective address specified by a lwat, ldat, stwat, or stdat instruction refers to storage that is Guarded		
Representative of *Instructions potentially causing a Data Storage interrupt when referring to Guarded storage*	The *Data Storage interrupt condition* holds The effective address specified by a lwat, ldat, stwat, or stdat instruction refers to storage that is Guarded No higher priority exception exists	A Data Storage interrupt occurs
III.6.5.3.Cond.13 Occurrence of a Data Storage interrupt when an accelerator is specified as the source of a copy instruction, normal memory is specified as the target of a paste. instruction, or an attempt is made to access an accelerator that is not properly configured for the software's use		
P1 cpabort copy paste.	The *Data Storage interrupt condition* holds An accelerator is specified as the source of a copy instruction, normal memory is specified as the target of a paste. instruction, or an attempt is made to access an accelerator that is not properly configured for the software's use No higher priority exception exists	A Data Storage interrupt occurs
III.6.5.3.Cond.14 Occurrence of a Data Storage interrupt when the translation for an attempted access has conflicting process- and partition-scoped page attributes		
Any Load or Store instruction	The *Data Storage interrupt condition* holds The translation for an attempted access has conflicting process- and partition-scoped page attributes No higher priority exception exists	A Data Storage interrupt occurs
III.6.5.3.Cond.15 Occurrence of a Data Storage interrupt when the translation for an attempted access has unsupported radix tree configuration in process-scoped tables		
Any Load or Store instruction	The *Data Storage interrupt condition* holds The translation for an attempted access has unsupported radix tree configuration in process-scoped tables No higher priority exception exists	A Data Storage interrupt occurs
III.6.5.3.Cond.16 Occurrence of a Data Storage interrupt when the Load Atomic or Store Atomic has an invalid function code		
Any Load Atomic or Store Atomic instruction	The *Data Storage interrupt condition* holds The Load Atomic or Store Atomic has an invalid function code No higher priority exception exists	A Data Storage interrupt occurs

19.2.5.2. Actions taken when a Data Storage interrupt occurs

Instructions potentially causing a Data Storage interrupt: Load instructions, Store instructions, icbi dcbz dcbst dcbf[l] lq stq lwat ldat lbarx lharx lwarx ldarx lqarx stwat stdat stbcx. sthcx. stwcx. stdcx. stqcx.

Value of DSISR bit 38: 1 for a Store, dcbz, Load Atomic, or Store Atomic instruction; otherwise 0

Notes:

Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

For DSISR bits 33, 34, 36, 37, 38, 41, 42, 44, 45, 46, 60, 61, 62: any one or more of these bits may be set to 1 in the DSISR, if multiple Data Storage exceptions occur for a given effective address

Instruction sequence	Compliance conditions	Expected result
III.6.5.3.Act.1 SRR0 and SRR1 are set correctly when a Data Storage interrupt occurs		
Representative of *Instructions potentially causing a Data Storage interrupt*	A Data Storage interrupt occurs	SRR0 = the effective address of the instruction that caused the interrupt $SRR1_{33:36}$ = 0 $SRR1_{42:47}$ = 0 All other bits of SRR1 = corresponding bits in the MSR
III.6.5.3.Act.2 DSISR is set correctly when a Data Storage interrupt occurs because MSR_{DR} = 1 and the translation for an attempted access is not found in the Page Table		
Representative of *Instructions potentially causing a Data Storage interrupt*	A Data Storage interrupt occurs MSR_{DR} = 1 The translation is not found in the Page Table	$DSISR_{33}$ = 1 $DSISR_{38}$ = *Value of DSISR bit 38* DSISR bits 32, 35, 39:40, 47:59, 63 = 0
III.6.5.3.Act.3 DSISR is set correctly when a Data Storage interrupt occurs because the effective address specified by the instruction refers to storage that is Write Through Required or Caching Inhibited		
Representative of *Instructions potentially causing a Data Storage interrupt when referring to Write Through Required or Caching Inhibited storage*	The effective address specified by the instruction refers to storage that is Write Through Required or Caching Inhibited A Data Storage interrupt occurs	$DSISR_{37}$ = 1 $DSISR_{38}$ = *Value of DSISR bit 38* DSISR bits 32, 35, 39:40, 47:59, 63 = 0
III.6.5.3.Act.4 DSISR is set correctly when a Data Storage interrupt occurs because the access violates Basic Storage Protection		
Any Load or Store instruction	The access violates Basic Storage Protection A Data Storage interrupt occurs	$DSISR_{36}$ = 1 $DSISR_{38}$ = *Value of DSISR bit 38* DSISR bits 32, 35, 39:40, 47:59, 63 = 0
III.6.5.3.Act.5 DSISR is set correctly when a Data Storage interrupt occurs because the access violates Virtual Page Class Key Storage Protection and $LPCR_{KBV}$ = 0		
Any Load or Store instruction	The access violates Virtual Page Class Key Storage Protection $LPCR_{KBV}$ = 0 A Data Storage interrupt occurs	$DSISR_{42}$ = 1 $DSISR_{38}$ = *Value of DSISR bit 38* DSISR bits 32, 35, 39:40, 47:59, 63 = 0
III.6.5.3.Act.6 DSISR is set correctly when a Data Storage interrupt occurs because a Data Address Watchpoint match occurs		
Any Load or Store instruction	A Data Address Watchpoint match occurs A Data Storage interrupt occurs	$DSISR_{41}$ = 1 $DSISR_{38}$ = *Value of DSISR bit 38* DSISR bits 32, 35, 39:40, 47:59, 63 = 0
III.6.5.3.Act.7 (removed)		
(removed)	(removed)	(removed)
III.6.5.3.Act.8 DSISR is set correctly when a Data Storage interrupt occurs because an attempt is made to execute a Fixed-Point Load or Store Caching Inhibited instruction with MSR_{DR} = 1		

Instruction sequence	Compliance conditions	Expected result
Any Fixed-Point Load or Store Caching Inhibited instruction	MSR_{DR} = 1 A Data Storage interrupt occurs	$DSISR_{62}$ = 1 $DSISR_{38}$ = *Value of DSISR bit 38* DSISR bits 32, 35, 39:40, 47:59, 63 = 0
III.6.5.3.Act.9 DSISR is set correctly when a Data Storage interrupt occurs because an attempt is made to execute a Fixed-Point Load or Store Caching Inhibited instruction specifying a storage location that is specified by the Hypervisor Real Mode Storage Control facility to be treated as non-Guarded		
Any Fixed-Point Load or Store Caching Inhibited instruction	The instruction specifies a storage location that is specified by the Hypervisor Real Mode Storage Control facility to be treated as non-Guarded A Data Storage interrupt occurs	$DSISR_{62}$ = 1 $DSISR_{38}$ = *Value of DSISR bit 38* DSISR bits 32, 35, 39:40, 47:59, 63 = 0
III.6.5.3.Act.10 DAR is set correctly when a Data Storage interrupt is caused by a cache management instruction (for reasons other than a Data Address Watchpoint match)		
Representative cache management instruction that is in *Instructions potentially causing a Data Storage interrupt*	A Data Storage interrupt occurs No Data Address Watchpoint match occurs	DAR is set to the effective address of a byte in the block that caused the exception
III.6.5.3.Act.11 DAR is set correctly when a Data Storage interrupt is caused by a quadword load or store instruction (for reasons other than a Data Address Watchpoint match)		
Representative quadword load or store instruction that is in *Instructions potentially causing a Data Storage interrupt*	A Data Storage interrupt occurs No Data Address Watchpoint match occurs	DAR is set to the effective address of a byte in the first aligned quadword for which access was attempted in the segment that caused the exception
III.6.5.3.Act.12 DAR is set correctly when a Data Storage interrupt is caused by a non-quadword load or store instruction (for reasons other than a Data Address Watchpoint match)		
Representative non-quadword load or store instruction that is in *Instructions potentially causing a Data Storage interrupt*	A Data Storage interrupt occurs No Data Address Watchpoint match occurs	DAR is set to the effective address of a byte in the first aligned doubleword for which access was attempted in the segment that caused the exception
III.6.5.3.Act.13 (removed)		
(removed)	(removed)	(removed)
III.6.5.3.Act.14 If a Data Storage interrupt occurs in 32-bit mode the high-order 32 bits of the DAR are set to 0		
Representative of *Instructions potentially causing a Data Storage interrupt*	A Data Storage interrupt occurs 32-bit computation mode DAR is set to a defined value	$DAR_{32:63}$ = 0
III.6.5.3.Act.15 DSISR is set correctly when a Data Storage interrupt occurs when the translation for an attempted access with conflicting process- and partition-scoped page attributes		
Representative of *Instructions potentially causing a Data Storage interrupt*	A Data Storage interrupt occurs The translation has conflicting process- and partition-scoped page attributes	$DSISR_{34}$ = 1 $DSISR_{38}$ = *Value of DSISR bit 38* DSISR bits 32, 35, 39:40, 47:59, 63 = 0
III.6.5.3.Act.16 DSISR is set correctly when a Data Storage interrupt occurs when the translation for an attempted access with unsupported radix tree configuration is found		
Representative of *Instructions potentially causing a Data Storage interrupt*	A Data Storage interrupt occurs The translation has unsupported radix tree configuration	$DSISR_{44}$ = 1 $DSISR_{38}$ = *Value of DSISR bit 38* DSISR bits 32, 35, 39:40, 47:59, 63 = 0
III.6.5.3.Act.17 DSISR is set correctly when a Data Storage interrupt occurs when the appropriate process table entry or segment table entry group cannot be translated when VPM = 0 and HR = 0, or the process table entry is invalid (independent of VPM) when HR = 0		
Representative of *Instructions potentially causing a Data Storage interrupt*	A Data Storage interrupt occurs	$DSISR_{46}$ = 1

Instruction sequence	Compliance conditions	Expected result
	The appropriate process table entry or segment table entry group cannot be translated when VPM = 0 and HR = 0, or the process table entry is invalid (independent of VPM) when HR = 0	$DSISR_{38}$ = *Value of DSISR bit 38* DSISR bits 32, 35, 39:40, 47:59, 63 = 0
III.6.5.3.Act.18 DSISR is set correctly when a Data Storage interrupt occurs when an accelerator is specified as the source of a copy instruction, normal memory is specified as the target of a paste. instruction, or an attempt is made to access an accelerator that is not properly configured for the software's use		
P1 cpabort copy paste.	A Data Storage interrupt occurs An accelerator is specified as the source of a copy instruction, normal memory is specified as the target of a paste. instruction, or an attempt is made to access an accelerator that is not properly configured for the software's use	$DSISR_{60}$ = 1 $DSISR_{38}$ = *Value of DSISR bit 38* DSISR bits 32, 35, 39:40, 47:59, 63 = 0
III.6.5.3.Act.19 DSISR is set correctly when a Data Storage interrupt occurs because MSR_{DR} = 1 and the Load Atomic or Store Atomic has an invalid function code		
Any Load Atomic or Store Atomic instruction	A Data Storage interrupt occurs The Load Atomic or Store Atomic has an invalid function code	$DSISR_{61}$ = 1 $DSISR_{38}$ = *Value of DSISR bit 38* DSISR bits 32, 35, 39:40, 47:59, 63 = 0

19.2.6. Data Segment Interrupt

Architecture sections:

- III.6.5.4 Data Segment Interrupt

Scenario groups:

- Conditions for occurrence of a Data Segment interrupt
- Actions taken when a Data Segment interrupt occurs

19.2.6.1. Conditions for occurrence of a Data Segment interrupt

Instructions potentially causing a Data Segment interrupt: Load instructions, Store instructions, icbi dcbz dcbst dcbf[l]

Indexed Move Assist instructions: lswx stswx

Instruction sequence	Compliance conditions	Expected result
III.6.5.4.Cond.1 A Data Segment interrupt occurs when a data access cannot be performed because data address translation is enabled and the effective address of any byte of the specified storage location cannot be translated to a virtual address (Paravirtualized HPT Translation with data address translation enabled)		
Representative of *Instructions potentially causing a Data Segment interrupt*	MSR_{DR} = 1 $LPCR_{VPM}$ = 1 $LPCR_{HR}$ = 0 The effective address of some byte of the storage location specified by the instruction cannot be translated to a virtual address No higher priority exception exists	A Data Segment interrupt occurs

Instruction sequence	Compliance conditions	Expected result
III.6.5.4.Cond.1b A Data Segment interrupt occurs when a data access cannot be performed because for the effective address specified by a Load, Store, icbi, dcbz, dcbst, or dcbf[l] instruction, $EA_{0:1}$ = 0b01 or $EA_{0:1}$ = 0b10 when $MSR_{HV\ PR}$ not equal to 0b10 and data address translation is enabled (Radix Tree Translation)		
Representative of *Instructions potentially causing a Data Segment interrupt*	MSR_{DR} = 1 $LPCR_{VPM}$ = 0 $LPCR_{HR}$ = 1 For the effective address specified by the instruction, $EA_{0:1}$ = 0b01 or $EA_{0:1}$ = 0b10 when $MSR_{HV\ PR}$ not equal to 0b10 No higher priority exception exists	A Data Segment interrupt occurs
III.6.5.4.Cond.2 If the XER specifies a length of zero for an indexed Move Assist instruction, a Data Segment interrupt does not occur.		
Representative of *Indexed Move Assist instructions*	$XER_{57:63}$ = 0 No higher priority exception exists	No Data Segment interrupt occurs

19.2.6.2. Actions taken when a Data Segment interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250]. Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252].

Instruction sequence	Compliance conditions	Expected result
III.6.5.4.Act.1 SRR0 and SRR1 are set correctly when a Data Segment interrupt occurs		
Representative of *Instructions potentially causing a Data Segment interrupt*	A Data Segment interrupt occurs	SRR0 = the effective address of the instruction that caused the interrupt $SRR1_{33:36}$ = 0 $SRR1_{42:47}$ = 0 All other bits of SRR1 = corresponding bits in the MSR
III.6.5.4.Act.2 DAR is set correctly when a Data Segment interrupt is caused by a cache management instruction		
Representative cache management instruction that is in *Instructions potentially causing a Data Segment interrupt*	A Data Segment interrupt occurs	DAR is set to the effective address of a byte in the block that caused the exception
III.6.5.4.Act.3 DAR is set correctly when a Data Segment interrupt is caused by a quadword load or store instruction		
Representative quadword load or store instruction that is in *Instructions potentially causing a Data Segment interrupt*	A Data Segment interrupt occurs	DAR is set to the effective address of a byte in the first aligned quadword for which access was attempted in the segment that caused the exception
III.6.5.4.Act.4 DAR is set correctly when a Data Segment interrupt is caused by a non-quadword load or store instruction		
Representative non-quadword load or store instruction that is in *Instructions potentially causing a Data Segment interrupt*	A Data Segment interrupt occurs	DAR is set to the effective address of a byte in the first aligned doubleword for which access was attempted in the segment that caused the exception
III.6.5.4.Act.5 If a Data Segment interrupt occurs in 32-bit mode the high-order 32 bits of the DAR are set to 0		
Representative of *Instructions potentially causing a Data Segment interrupt*	A Data Segment interrupt occurs 32-bit computation mode	$DAR_{32:63}$ = 0

19.2.7. Instruction Storage Interrupt

Architecture sections:

- III.6.5.5 Instruction Storage Interrupt

Scenario groups:

- Conditions for occurrence of an Instruction Storage interrupt
- Actions taken when an Instruction Storage interrupt occurs

19.2.7.1. Conditions for occurrence of an Instruction Storage interrupt

Instruction Storage interrupt condition:

(a) HPT Translation is being performed, the value of the expression (($MSR_{HV\ PR}$ = 0b10) | ((¬VPM | ¬$PRTE_V$) & MSR_{IR})) is 1, and the next instruction to be executed cannot be fetched, or

(b) Radix Tree translation is being performed and process-scoped translation prevents the next instruction to be executed from being fetched

Instruction sequence	Compliance conditions	Expected result
III.6.5.5.Cond.1 Occurrence of an Instruction Storage interrupt when the next instruction to be executed cannot be fetched because instruction address translation is enabled and the virtual address cannot be translated to a real address because no valid PTE was found for HPT translation with VPM off		
Any instruction	The *Instruction Storage interrupt condition* holds MSR_{IR} = 1 $LPCR_{VPM}$ = 0 $LPCR_{HR}$ = 0 The virtual address cannot be translated to a real address No higher priority exception exists	An Instruction Storage interrupt occurs
III.6.5.5.Cond.1b Occurrence of an Instruction Storage interrupt when the next instruction to be executed cannot be fetched because instruction address translation is enabled and the effective address cannot be translated to a real address because no valid PTE was found for the process-scoped Radix Tree translation		
Any instruction	The *Instruction Storage interrupt condition* holds MSR_{IR} = 1 $LPCR_{VPM}$ = 0 $LPCR_{HR}$ = 1 The virtual address cannot be translated to a real address No higher priority exception exists	An Instruction Storage interrupt occurs
III.6.5.5.Cond.2 Occurrence of an Instruction Storage interrupt when the next instruction to be executed cannot be fetched because the fetch access violates storage protection		
Any instruction	The *Instruction Storage interrupt condition* holds	An Instruction Storage interrupt occurs

Instruction sequence	Compliance conditions	Expected result
	The fetch access violates storage protection No higher priority exception exists	
III.6.5.5.Cond.3 Occurrence of an Instruction Storage interrupt when the next instruction to be executed cannot be fetched because the appropriate process table entry or segment table entry group cannot be translated when VPM = 0 and HR = 0, or the process table entry is invalid (independent of VPM) when HR = 0		
Any instruction	The *Instruction Storage interrupt condition* holds The appropriate process table entry or segment table entry group cannot be translated when VPM = 0 and HR = 0, or the process table entry is invalid (independent of VPM) when HR = 0 No higher priority exception exists	An Instruction Storage interrupt occurs
III.6.5.5.Cond.4 Occurrence of an Instruction Storage interrupt when the next instruction to be executed cannot be fetched because the translation has conflicting process- and partition-scoped page attributes		
Any instruction	The *Instruction Storage interrupt condition* holds The translation has conflicting process- and partition-scoped page attributes No higher priority exception exists	An Instruction Storage interrupt occurs
III.6.5.5.Cond.5 Occurrence of an Instruction Storage interrupt when the next instruction to be executed cannot be fetched because the translation has unsupported radix tree configuration		
Any instruction	The *Instruction Storage interrupt condition* holds The translation has unsupported radix tree configuration No higher priority exception exists	An Instruction Storage interrupt occurs

19.2.7.2. Actions taken when an Instruction Storage interrupt occurs

Notes:

Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

For SRR1 bits 33, 34, 35, 36, 42, 44, 45, 46: any one or more of these bits may be set to 1, if multiple Instruction Storage exceptions occur due to attempting to fetch a single instruction

Instruction sequence	Compliance conditions	Expected result
III.6.5.5.Act.1 SRR0 is set correctly when an Instruction Storage interrupt occurs		
Any instruction	An Instruction Storage interrupt occurs	SRR0 = the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present
III.6.5.5.Act.2 SRR1 is set correctly when an Instruction Storage interrupt occurs because MSR_{IR} = 1 and the translation for an attempted fetch access is not found in the Page Table		
Any instruction	MSR_{IR} = 1 The translation is not found in the Page Table	$SRR1_{33}$ = 1 SRR1 bits 43, 47 = 0

Instruction sequence	Compliance conditions	Expected result
	An Instruction Storage interrupt occurs	SRR1 bits 0:32, 37:41, 48:63 = corresponding bits in the MSR
III.6.5.5.Act.3 SRR1 is set correctly when an Instruction Storage interrupt occurs because the attempted fetch access is to No-execute or Guarded storage		
Any instruction	The fetch access is to No-execute or Guarded storage An Instruction Storage interrupt occurs	$SRR1_{35}$ = 1 SRR1 bits 43, 47 = 0 SRR1 bits 0:32, 37:41, 48:63 = corresponding bits in the MSR
III.6.5.5.Act.4 SRR1 is set correctly when an Instruction Storage interrupt occurs because the attempted fetch access violates Basic Storage Protection		
Any instruction	The fetch access violates Basic Storage Protection An Instruction Storage interrupt occurs	$SRR1_{36}$ = 1 SRR1 bits 43, 47 = 0 SRR1 bits 0:32, 37:41, 48:63 = corresponding bits in the MSR
III.6.5.5.Act.5 SRR1 is set correctly when an Instruction Storage interrupt occurs because the attempted fetch access is not permitted by virtual page class key protection		
Any instruction	The fetch access violates Virtual Page Class Key Storage Protection An Instruction Storage interrupt occurs	$SRR1_{42}$ = 1 SRR1 bits 43, 47 = 0 SRR1 bits 0:32, 37:41, 48:63 = corresponding bits in the MSR
III.6.5.5.Act.6 SRR1 is set correctly when an Instruction Storage interrupt occurs because the attempted fetch access has conflicting process- and partition-scoped page attributes		
Any instruction	The translation for an attempted fetch access has conflicting process- and partition-scoped page attributes An Instruction Storage interrupt occurs	$SRR1_{34}$ = 1 SRR1 bits 43, 47 = 0 SRR1 bits 0:32, 37:41, 48:63 = corresponding bits in the MSR
III.6.5.5.Act.7 SRR1 is set correctly when an Instruction Storage interrupt occurs because the translation for an attempted fetch access has unsupported radix tree configuration		
Any instruction	The translation for an attempted fetch access has unsupported radix tree configuration An Instruction Storage interrupt occurs	$SRR1_{44}$ = 1 SRR1 bits 43, 47 = 0 SRR1 bits 0:32, 37:41, 48:63 = corresponding bits in the MSR
III.6.5.5.Act.8 SRR1 is set correctly when an Instruction Storage interrupt occurs because the appropriate process table entry or segment table entry group cannot be translated when VPM = 0 and HR = 0, or the process table entry is invalid (independent of VPM) when HR = 0		
Any instruction	The appropriate process table entry or segment table entry group cannot be translated when VPM = 0 and HR = 0, or the process table entry is invalid (independent of VPM) when HR = 0 An Instruction Storage interrupt occurs	$SRR1_{46}$ = 1 SRR1 bits 43, 47 = 0 SRR1 bits 0:32, 37:41, 48:63 = corresponding bits in the MSR

19.2.8. Instruction Segment Interrupt

Architecture sections:

- III.6.5.6 Instruction Segment Interrupt

Scenario groups:

- Conditions for occurrence of an Instruction Segment interrupt
- Actions taken when an Instruction Segment interrupt occurs

19.2.8.1. Conditions for occurrence of an Instruction Segment interrupt

Instruction Segment interrupt condition:

For Paravirtualized HPT Translation, an Instruction Segment interrupt occurs when no higher priority exception exists and the next instruction to be executed cannot be fetched because instruction address translation is enabled and the effective address cannot be translated to a virtual address. For Radix Tree Translation (in other than hypervisor real mode), an Instruction Segment interrupt occurs when no higher priority exception exists and the next instruction to be executed cannot be fetched because $EA_{0:1}$ = 0b01 or $EA_{0:1}$ = 0b10 when $MSR_{HV\ PR}$ is not equal to 0b10 and instruction address translation is enabled, or $EA_{2:63}$ is outside the range translated by the appropriate Radix Tree

Instruction sequence	Compliance conditions	Expected result
III.6.5.6.Cond.1 Occurrence of an Instruction Segment interrupt when the next instruction to be executed cannot be fetched because the effective address cannot be translated to a virtual address (Paravirtualized HPT Translation with instruction address translation enabled)		
Any instruction	The *Instruction Segment interrupt condition* holds MSR_{IR} = 1 $LPCR_{VPM}$ = 1 $LPCR_{HR}$ = 0 The effective address cannot be translated to a virtual address No higher priority exception exists	An Instruction Segment interrupt occurs
III.6.5.6.Cond.1b Occurrence of an Instruction Segment interrupt when the next instruction to be executed cannot be fetched because $EA_{0:1}$ = 0b01 or $EA_{0:1}$ = 0b10 when $MSR_{HV\ PR}$ is not equal to 0b10 and instruction address translation is enabled (Radix Tree Translation)		
Any instruction	The *Instruction Segment interrupt condition* holds MSR_{IR} = 1 $LPCR_{VPM}$ = 0 $LPCR_{HR}$ = 1 The next instruction to be executed cannot be fetched because $EA_{0:1}$ = 0b01 or $EA_{0:1}$ = 0b10 when $MSR_{HV\ PR}$ is not equal to 0b10 No higher priority exception exists	An Instruction Segment interrupt occurs

19.2.8.2. Actions taken when an Instruction Segment interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.6.Act.1 SRR0 and SRR1 are set correctly when an Instruction Segment interrupt occurs		
Any instruction	An Instruction Segment interrupt occurs	SRR0 = the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present $SRR1_{33:36} = 0$ $SRR1_{42:47} = 0$ All other bits of SRR1 = corresponding bits in the MSR

19.2.9. Direct External Interrupt

Architecture sections:

- III.6.5.7.1 Direct External Interrupt

Scenario groups:

- Conditions for occurrence of a Direct External Interrupt
- Actions taken when a Direct External Interrupt occurs

Note: this is a system-caused interrupt, not directly caused by the execution of an instruction

19.2.9.1. Conditions for occurrence of a Direct External Interrupt

Direct external interrupt condition:

MSR_{EE} & ⌐(MSR_{HV} & ⌐MSR_{PR} & $LPCR_{HEIC}$) | (⌐(LPES) & (⌐(MSR_{HV}) | MSR_{PR})) = 1

Instruction sequence	Compliance conditions	Expected result
III.6.5.7.1.Cond.1 Occurrence of a Direct External Interrupt when a Direct External exception exists		
(No instruction sequence specified)	The *Direct external interrupt condition* holds A Direct External exception exists No higher priority exception exists	A Direct External Interrupt occurs

19.2.9.2. Actions taken when a Direct External Interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.7.1.Act.1 HSRR0 and HSRR1 are set correctly when a Direct External Interrupt occurs and LPES = 0		
(No instruction sequence specified)	A Direct External Interrupt occurs $LPCR_{LPES} = 0$	HSRR0 = the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present $HSRR1_{33:36} = 0$ $HSRR1_{42:47} = 0$

Instruction sequence	Compliance conditions	Expected result
		All other bits of HSRR1 = corresponding bits in the MSR
III.6.5.7.1.Act.2 SRR0 and SRR1 are set correctly when a Direct External Interrupt occurs and LPES = 1		
(No instruction sequence specified)	A Direct External Interrupt occurs $LPCR_{LPES} = 1$	SRR0 = the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present $SRR1_{33:36} = 0$ $SRR1_{42:47} = 0$ All other bits of SRR1 = corresponding bits in the MSR
III.6.5.7.1.Act.3 The occurrence of a Direct External interrupt does not cause the Direct External exception to cease to exist		
(No instruction sequence specified)	A Direct External Interrupt occurs	The Direct External exception that caused the interrupt still exists

19.2.10. Mediated External Interrupt

Architecture sections:

- III.6.5.7.2 Mediated External Interrupt

Scenario groups:

- Conditions for occurrence of a Mediated External Interrupt
- Actions taken when a Mediated External Interrupt occurs

Note: this is a system-caused interrupt, not directly caused by the execution of an instruction

19.2.10.1. Conditions for occurrence of a Mediated External Interrupt

Mediated external interrupt condition:

MSR_{EE} & (¬(MSR_{HV}) | MSR_{PR}) = 1

Instruction sequence	Compliance conditions	Expected result
III.6.5.7.2 Cond.1 Occurrence of a Mediated External Interrupt when a Mediated External exception exists		
(No instruction sequence specified)	The *Mediated external interrupt condition* holds A Mediated External exception exists No higher priority exception exists	A Mediated External Interrupt occurs

19.2.10.2. Actions taken when a Mediated External Interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.7.2 Act.1 HSRR0 and HSRR1 are set correctly when a Mediated External Interrupt occurs and LPES = 0		
(No instruction sequence specified)	A Mediated External Interrupt occurs $LPCR_{LPES} = 0$	HSRR0 = the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present

Instruction sequence	Compliance conditions	Expected result
		$HSRR1_{33:36}$ = 0 $HSRR1_{42}$ = 1 $HSRR1_{43:47}$ = 0 All other bits of HSRR1 = corresponding bits in the MSR
III.6.5.7.2 Act.2 SRR0 and SRR1 are set correctly when a Mediated External Interrupt occurs and LPES = 1		
(No instruction sequence specified)	A Mediated External Interrupt occurs $LPCR_{LPES}$ = 1	SRR0 = the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present $SRR1_{33:36}$ = 0 $SRR1_{42:47}$ = 0 All other bits of SRR1 = corresponding bits in the MSR
III.6.5.7.2 Act.3 The occurrence of a Mediated External interrupt does not cause the Mediated External exception to cease to exist		
(No instruction sequence specified)	A Mediated External Interrupt occurs	The Mediated External exception that caused the interrupt still exists

19.2.11. Alignment Interrupt

Architecture sections:

- III.6.5.8 Alignment Interrupt

Scenario groups:

- Conditions for occurrence of an Alignment interrupt
- Actions taken when an Alignment interrupt occurs

19.2.11.1. Conditions for occurrence of an Alignment interrupt

Instruction sequence	Compliance conditions	Expected result
III.6.5.8.Cond.1 Occurrence of an Alignment interrupt when a Load/Store Multiple instruction is executed in Little-Endian mode		
Any Load/Store Multiple instruction	MSR_{LE} = 1 No higher priority exception exists	An Alignment interrupt occurs
III.6.5.8.Cond.2 Occurrence of an Alignment interrupt when a Move Assist instruction is executed in Little-Endian mode, when the string length is not zero		
Any Move Assist instruction	MSR_{LE} = 1 The designated string length != 0 No higher priority exception exists	An Alignment interrupt occurs
III.6.5.8.Cond.3 Occurrence of an Alignment interrupt when an instruction has an unaligned storage operand		
Representative of: copy paste. lwat ldat lharx lwarx ldarx lqarx stwat stdat sthcx. stwcx. stdcx. stqcx.	The instruction has an unaligned storage operand Execution of the instruction does not yield boundedly undefined results (according to the instruction description) No higher priority exception exists	An Alignment interrupt occurs

Instruction sequence	Compliance conditions	Expected result
III.6.5.8.Cond.4 Occurrence of an Alignment interrupt when a Load Atomic or Store Atomic instruction operand crosses a 32-byte boundary		
Any Load Atomic or Store Atomic instruction	The instruction operand crosses a 32-byte boundary No higher priority exception exists	An Alignment interrupt occurs

19.2.11.2. Actions taken when an Alignment interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.8.Act.1 SRR0 and SRR1 are set correctly when an Alignment interrupt occurs		
Any instruction that may cause an Alignment interrupt	An Alignment interrupt occurs	SRR0 = the effective address of the instruction that caused the interrupt $SRR1_{33:36}$ = 0 $SRR1_{42:47}$ = 0 All other bits of SRR1 = corresponding bits in the MSR
III.6.5.8.Act.2 DAR is set correctly when an Alignment interrupt occurs in 64-bit mode		
Any instruction that may cause an Alignment interrupt	An Alignment interrupt occurs 64-bit computation mode	DAR = the effective address computed by the instruction
III.6.5.8.Act.3 If an Alignment interrupt occurs in 32-bit mode the high-order 32 bits of the DAR are set to 0		
Any instruction that may cause an Alignment interrupt	An Alignment interrupt occurs 32-bit computation mode	$DAR_{32:63}$ = 0

19.2.12. Program Interrupt

Architecture sections:

- III.6.5.9 Program Interrupt

Scenario groups:

- Conditions for occurrence of a Program interrupt
- Actions taken when a Program interrupt occurs

19.2.12.1. Conditions for occurrence of a Program interrupt

Instruction sequence	Compliance conditions	Expected result
III.6.5.9.Cond.1 Occurrence of a Floating-Point Enabled Exception type Program interrupt when a floating-point instruction causes an enabled exception		
Any floating-point instruction	(MSR_{FE0} | MSR_{FE1}) & $FPSCR_{FEX}$ = 1	A Floating-Point Enabled Exception type Program interrupt occurs
III.6.5.9.Cond.2 Occurrence of a TM Bad Thing type Program interrupt when an rfebb, rfid, rfscv, hrfid, or mtmsrd instruction attempts to cause an illegal state transition		
Representative of: rfebb rfid rfscv hrfid mtmsrd	The instruction attempts to cause an illegal state transition	A TM Bad Thing type Program interrupt occurs

Instruction sequence	Compliance conditions	Expected result
III.6.5.9.Cond.3 Occurrence of a TM Bad Thing type Program interrupt when an rfid, rfscv, hrfid, or mtmsrd instruction attempts to cause a transition to Problem state with a legal transaction state transition resulting in an active transaction (Transactional or Suspended state) when TM is disabled by the PCR ($PCR_{v2.06}$ = 1)		
Representative of: rfid rfscv hrfid mtmsrd	The instruction attempts to cause a transition to Problem state with a legal transaction state transition resulting in an active transaction (Transactional or Suspended state) $PCR_{v2.06}$ = 1	A TM Bad Thing type Program interrupt occurs
III.6.5.9.Cond.4 (removed)		
(removed)	(removed)	(removed)
III.6.5.9.Cond.5 Occurrence of a TM Bad Thing type Program interrupt when an attempt is made to execute trechkpt. in Transactional or Suspended state or when $TEXASR_{FS}$ = 0		
trechkpt	The thread is in Transactional or Suspended state, or $TEXASR_{FS}$ = 0	A TM Bad Thing type Program interrupt occurs
III.6.5.9.Cond.6 Occurrence of a TM Bad Thing type Program interrupt when an attempt is made to execute tend. in Suspended state		
tend.	The thread is in Suspended state	A TM Bad Thing type Program interrupt occurs
III.6.5.9.Cond.7 Occurrence of a TM Bad Thing type Program interrupt when an attempt is made to execute treclaim. in Non-transactional state		
treclaim.	The thread is in Non-transactional state	A TM Bad Thing type Program interrupt occurs
III.6.5.9.Cond.8 Occurrence of a TM Bad Thing type Program interrupt when an attempt is made to execute an mtspr instruction targeting a TM register in other than Non-transactional state		
mtspr	The designated SPR is a TM register The thread is **not** in Non-transactional state	A TM Bad Thing type Program interrupt occurs
III.6.5.9.Cond.9 A TM Bad Thing type Program interrupt does not occur when an attempt is made to execute an mtspr instruction targeting TFHAR in Suspended state		
mtspr	The designated SPR is TFHAR The thread is **not** in Non-transactional state	No TM Bad Thing type Program interrupt occurs
III.6.5.9.Cond.10 Occurrence of a TM Bad Thing type Program interrupt when an attempt is made to execute a stop instruction in Suspended state		
stop	The thread is in Suspended state	A TM Bad Thing type Program interrupt occurs
III.6.5.9.Cond.11 Occurrence of a Privileged Instruction type Program interrupt when MSR_{PR} = 1 and execution is attempted of a privileged instruction		
Any privileged instruction	MSR_{PR} = 1	A Privileged Instruction type Program interrupt occurs
III.6.5.9.Cond.12 Occurrence of a Privileged Instruction type Program interrupt when MSR_{PR} = 1 and execution is attempted of an mtspr or mfspr instruction with an SPR field that contains a value having spr_0 = 1		
mtspr or mfspr	MSR_{PR} = 1 The SPR field contains a value having spr_0 = 1	A Privileged Instruction type Program interrupt occurs
III.6.5.9.Cond.13 Occurrence of a Privileged Instruction type Program interrupt when $MSR_{HV\ PR}$ = 0b00 and $LPCR_{EVIRT}$ = 0, and execution is attempted of an mtspr or mfspr instruction with an SPR field that designates an SPR that is accessible by the instruction only when the thread is in hypervisor state		
mtspr or mfspr	$MSR_{HV\ PR}$ = 0b00 $LPCR_{EVIRT}$ = 0	A Privileged Instruction type Program interrupt occurs

Instruction sequence	Compliance conditions	Expected result
	The designated SPR is accessible by the instruction only when the thread is in hypervisor state	
III.6.5.9.Cond.14 Occurrence of a Privileged Instruction type Program interrupt when $MSR_{HV\ PR}$ = 0b00 and $LPCR_{EVIRT}$ = 0, and execution of a hypervisor-privileged instruction is attempted		
Any hypervisor-privileged instruction	$MSR_{HV\ PR}$ = 0b00 $LPCR_{EVIRT}$ = 0	A Privileged Instruction type Program interrupt occurs
III.6.5.9.Cond.15 Occurrence of a Trap type Program interrupt when any of the conditions specified in a Trap instruction is met		
Any Trap instruction	Any of the conditions specified in the instruction is met	A Trap type Program interrupt occurs

19.2.12.2. Actions taken when a Program interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.9.Act.1 SRR0 and $SRR1_{47}$ are set correctly when a Program interrupt occurs, for all Program interrupts except a Floating-Point Enabled Exception type Program interrupt		
Any instruction that may cause a Program interrupt	A Program interrupt occurs, of any type other than Floating-Point Enabled Exception	SRR0 = the effective address of the instruction that caused the corresponding exception $SRR1_{47}$ = 0
III.6.5.9.Act.2 SRR0 and $SRR1_{47}$ are set correctly when a Program interrupt occurs, for a Floating-Point Enabled Exception type Program interrupt when $MSR_{FE0\ FE1}$ = 0b00		
Any floating-point instruction	$MSR_{FE0\ FE1}$ = 0b00 before instruction execution $FPSCR_{FEX}$ = 1 The instruction changes $MSR_{FE0\ FE1}$ to a nonzero value A Floating-Point Enabled Exception type Program interrupt occurs	SRR0 = the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present $SRR1_{47}$ = 1
III.6.5.9.Act.3 SRR0 and $SRR1_{47}$ are set correctly when a Program interrupt occurs, for a Floating-Point Enabled Exception type Program interrupt when $MSR_{FE0\ FE1}$ = 0b11		
Any floating-point instruction	$MSR_{FE0\ FE1}$ = 0b11 A Floating-Point Enabled Exception type Program interrupt occurs	SRR0 = the effective address of the instruction that caused the Floating-Point Enabled Exception $SRR1_{47}$ = 0
III.6.5.9.Act.4 SRR0 and $SRR1_{47}$ are set correctly when a Program interrupt occurs, for a Floating-Point Enabled Exception type Program interrupt when $MSR_{FE0\ FE1}$ = 0b01 or 0b10		
Any floating-point instruction	$MSR_{FE0\ FE1}$ = 0b01 or 0b10 A Floating-Point Enabled Exception type Program interrupt occurs	SRR0 = the effective address of the first instruction that caused a Floating-Point Enabled Exception since the most recent time $FPSCR_{FEX}$ was changed from 1 to 0, or of some subsequent instruction $SRR1_{47}$ = 0 if there is only one instruction causing the exception, otherwise 1
III.6.5.9.Act.5 Bits 42, 43, 45, 46 of SRR1 are set correctly when a TM Bad Thing type Program interrupt occurs		
Any instruction that may cause a TM Bad Thing type Program interrupt	A TM Bad Thing type Program interrupt occurs	$SRR1_{42}$ = 1

Instruction sequence	Compliance conditions	Expected result
		$SRR1_{43, 45, 46} = 0$
III.6.5.9.Act.6 Bits 42, 43, 45, 46 of SRR1 are set correctly when a Floating-Point Enabled Exception type Program interrupt occurs		
Any instruction that may cause a Floating-Point Enabled Exception type Program interrupt	A Floating-Point Enabled Exception type Program interrupt occurs	$SRR1_{43} = 1$ $SRR1_{42, 45, 46} = 0$
III.6.5.9.Act.7 Bits 42, 43, 45, 46 of SRR1 are set correctly when a Privileged Instruction type Program interrupt occurs		
Any instruction that may cause a Privileged Instruction type Program interrupt	A Privileged Instruction type Program interrupt occurs	$SRR1_{45} = 1$ $SRR1_{42, 43, 46} = 0$
III.6.5.9.Act.8 Bits 42, 43, 45, 46 of SRR1 are set correctly when a Trap type Program interrupt occurs		
Any instruction that may cause a Trap type Program interrupt	A Trap type Program interrupt occurs	$SRR1_{46} = 1$ $SRR1_{42, 43, 45} = 0$
III.6.5.9.Act.9 Remaining bits of SRR1 are set correctly when a Program interrupt occurs		
Any instruction that may cause a Program interrupt	A Program interrupt occurs	$SRR1_{33:36} = 0$ $SRR1_{44} = 0$ SRR1 bits 0:32, 37:41, 48:63 = corresponding bits in the MSR

19.2.13. Floating-Point Unavailable Interrupt

Architecture sections:

- III.6.5.10 Floating-Point Unavailable Interrupt

Scenario groups:

- Conditions for occurrence of a Floating-Point Unavailable interrupt
- Actions taken when a Floating-Point Unavailable interrupt occurs

19.2.13.1. Conditions for occurrence of a Floating-Point Unavailable interrupt

Instruction sequence	Compliance conditions	Expected result
III.6.5.10.Cond.1 Occurrence of a Floating-Point Unavailable interrupt when an attempt is made to execute a floating-point instruction and $MSR_{FP} = 0$		
Any floating-point instruction	$MSR_{FP} = 0$ No higher priority exception exists	A Floating-Point Unavailable interrupt occurs

19.2.13.2. Actions taken when a Floating-Point Unavailable interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.10.Act.1 SRR0 and SRR1 are set correctly when a Floating-Point Unavailable interrupt occurs		
Any floating-point instruction	A Floating-Point Unavailable interrupt occurs	SRR0 = the effective address of the instruction that caused the interrupt

Instruction sequence	Compliance conditions	Expected result
		$SRR1_{33:36}$ = 0 $SRR1_{42:47}$ = 0 All other bits of SRR1 = corresponding bits in the MSR

19.2.14. Decrementer Interrupt

Architecture sections:

- III.6.5.11 Decrementer Interrupt

Scenario groups:

- Conditions for occurrence of a Decrementer interrupt
- Actions taken when a Decrementer interrupt occurs

Note: this is a system-caused interrupt, not directly caused by the execution of an instruction

19.2.14.1. Conditions for occurrence of a Decrementer interrupt

Instruction sequence	Compliance conditions	Expected result
III.6.5.11.Cond.1 Occurrence of a Decrementer interrupt when a Decrementer exception exists		
(No instruction sequence specified)	A Decrementer exception exists MSR_{EE} = 1 No higher priority exception exists	A Decrementer interrupt occurs

19.2.14.2. Actions taken when a Decrementer interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.11.Act.1 SRR0 and SRR1 are set correctly when a Decrementer interrupt occurs		
(No instruction sequence specified)	A Decrementer interrupt occurs	SRR0 = the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present $SRR1_{33:36}$ = 0 $SRR1_{42:47}$ = 0 All other bits of SRR1 = corresponding bits in the MSR

19.2.15. Hypervisor Decrementer Interrupt

Architecture sections:

- III.6.5.12 Hypervisor Decrementer Interrupt

Scenario groups:

- Conditions for occurrence of a Hypervisor Decrementer interrupt
- Actions taken when a Hypervisor Decrementer interrupt occurs

Note: this is a system-caused interrupt, not directly caused by the execution of an instruction

19.2.15.1. Conditions for occurrence of a Hypervisor Decrementer interrupt

Hypervisor Decrementer interrupt condition:

$(MSR_{EE} \mid (\neg MSR_{HV}) \mid MSR_{PR}) \& HDICE = 1$

Instruction sequence	Compliance conditions	Expected result
III.6.5.12.Cond.1 Occurrence of a Hypervisor Decrementer interrupt when a Hypervisor Decrementer exception exists		
(No instruction sequence specified)	The *Hypervisor Decrementer interrupt condition* holds A Hypervisor Decrementer exception exists No higher priority exception exists	A Hypervisor Decrementer interrupt occurs

19.2.15.2. Actions taken when a Hypervisor Decrementer interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.12.Act.1 HSRR0 and HSRR1 are set correctly when a Hypervisor Decrementer interrupt occurs		
(No instruction sequence specified)	A Hypervisor Decrementer interrupt occurs	HSRR0 = the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present $HSRR1_{33:36} = 0$ $HSRR1_{42:47} = 0$ All other bits of HSRR1 = corresponding bits in the MSR

19.2.16. Directed Privileged Doorbell Interrupt

Architecture sections:

- III.6.5.13 Directed Privileged Doorbell Interrupt

Scenario groups:

- Conditions for occurrence of a Directed Privileged Doorbell interrupt
- Actions taken when a Directed Privileged Doorbell interrupt occurs

Note: this is a system-caused interrupt, not directly caused by the execution of an instruction

19.2.16.1. Conditions for occurrence of a Directed Privileged Doorbell interrupt

Instruction sequence	Compliance conditions	Expected result
III.6.5.13.Cond.1 Occurrence of a Directed Privileged Doorbell interrupt when a Directed Privileged Doorbell exception exists		
(No instruction sequence specified)	A Directed Privileged Doorbell exception exists MSR_{EE} = 1 No higher priority exception exists	A Directed Privileged Doorbell interrupt occurs

19.2.16.2. Actions taken when a Directed Privileged Doorbell interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.13.Act.1 SRR0 and SRR1 are set correctly when a Directed Privileged Doorbell interrupt occurs		
(No instruction sequence specified)	A Directed Privileged Doorbell interrupt occurs	SRR0 = the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present $SRR1_{33:36}$ = 0 $SRR1_{42:47}$ = 0 All other bits of SRR1 = corresponding bits in the MSR

19.2.17. System Call Interrupt

Architecture sections:

- III.6.5.14 System Call Interrupt

Scenario groups:

- Conditions for occurrence of a System Call interrupt
- Actions taken when a System Call interrupt occurs

19.2.17.1. Conditions for occurrence of a System Call interrupt

Instruction sequence	Compliance conditions	Expected result
III.6.5.14.Cond.1 Occurrence of a System Call interrupt when a System Call instruction is executed		
sc		A System Call interrupt occurs

19.2.17.2. Actions taken when a System Call interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.14.Act.1 SRR0 and SRR1 are set correctly when a System Call interrupt occurs		
sc	A System Call interrupt occurs	SRR0 = the effective address of the instruction following the System Call instruction $SRR1_{33:36}$ = 0 $SRR1_{42:47}$ = 0 All other bits of SRR1 = corresponding bits in the MSR

19.2.18. Trace Interrupt

Architecture sections:

- III.6.5.15 Trace Interrupt

Scenario groups:

- Conditions for occurrence of a Trace interrupt
- Actions taken when a Trace interrupt occurs

19.2.18.1. Conditions for occurrence of a Trace interrupt

Conditions in which instructions are not traced:

- The instruction is one of: rfid hrfid rfscv sc scv, Power-Saving Mode instruction
- The instruction is a Trap instruction that traps
- The instruction causes an interrupt other than Trace interrupt
- In Transactional state: the instruction is disallowed in Transactional state, or causes types of accesses that are disallowed in Transactional state
- The instruction is tbegin. executed at maximum nesting depth

Instruction sequence	Compliance conditions	Expected result
III.6.5.15.Cond.1 Occurrence of a Trace interrupt when the instruction is mtmsr[d] and MSR_{TE} = 0b10		
Representative of: mtmsr mtmsrd	MSR_{TE} = 0b10 when the instruction was initiated The instruction is successfully completed None of the *Conditions in which instructions are not traced* occurs No higher priority exception exists	A Trace interrupt occurs
III.6.5.15.Cond.2 Occurrence of a Trace interrupt when the instruction is not mtmsr[d] and MSR_{TE} = 0b10 when the instruction was initiated		
Any instruction that is not mtmsr[d], and is not listed in the *Conditions in which instructions are not traced*	MSR_{TE} = 0b10 The instruction is successfully completed None of the *Conditions in which instructions are not traced* occurs No higher priority exception exists	A Trace interrupt occurs
III.6.5.15.Cond.3 Occurrence of a Trace interrupt when a CIABR match occurs		

Instruction sequence	Compliance conditions	Expected result
Any instruction that is not listed in the *Conditions in which instructions are not traced*	The instruction is successfully completed None of the *Conditions in which instructions are not traced* occurs A CIABR match occurs No higher priority exception exists	A Trace interrupt occurs
III.6.5.15.Cond.4 Occurrence of a Trace interrupt when the instruction is a Branch instruction and MSR_{TE} = 0b01		
Any Branch instruction, and is not listed in the *Conditions in which instructions are not traced*	MSR_{TE} = 0b01 The instruction is successfully completed None of the *Conditions in which instructions are not traced* occurs No higher priority exception exists	A Trace interrupt occurs

19.2.18.2. Actions taken when a Trace interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.15.Act.1 SRR0 is set correctly when a Trace interrupt occurs		
Any instruction that is not listed in the *Conditions in which instructions are not traced*	A Trace interrupt occurs	SRR0 = the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present
III.6.5.15.Act.2 SRR1 is set correctly when a Trace interrupt occurs **not** as the result of a CIABR match and the traced instruction is a Load instruction (other than a Load String instruction with a string length of 0) or is specified to be treated as a Load instruction		
A Load instruction (other than a Load String instruction with a string length of 0), or an instruction specified to be treated as a Load instruction	A Trace interrupt occurs No CIABR match occurs	$SRR1_{35}$ = 1 $SRR1_{33}$ = 1 SRR1 bits 34, 44:47 = 0 SRR1 bits 0:32, 37:42, 48:63 = corresponding bits in the MSR
III.6.5.15.Act.3 SRR1 is set correctly when a Trace interrupt occurs **not** as the result of a CIABR match and the traced instruction is a Store instruction (other than a Store String instruction with a string length of 0) or is specified to be treated as a Store instruction		
A Store instruction (other than a Store String instruction with a string length of 0), or an instruction specified to be treated as a Store instruction	A Trace interrupt occurs No CIABR match occurs	$SRR1_{36}$ = 1 $SRR1_{33}$ = 1 SRR1 bits 34, 44:47 = 0 SRR1 bits 0:32, 37:42, 48:63 = corresponding bits in the MSR
III.6.5.15.Act.4 SRR1 is set correctly when a Trace interrupt occurs as the result of a CIABR match		
Any instruction that is not listed in the *Conditions in which instructions are not traced*	A Trace interrupt occurs A CIABR match occurs	$SRR1_{43}$ = 1 $SRR1_{33}$ = 1 SRR1 bits 34, 44:47 = 0 SRR1 bits 0:32, 37:42, 48:63 = corresponding bits in the MSR
III.6.5.15.Act.5 SIAR is set correctly when a Trace interrupt occurs **not** as the result of a CIABR match		

Instruction sequence	Compliance conditions	Expected result
Any instruction that is not listed in the *Conditions in which instructions are not traced*	A Trace interrupt occurs No CIABR match occurs $MMCR0_{PMAE}$ = 0	SIAR = the effective address of the traced instruction
III.6.5.15.Act.6 SDAR is set correctly when a Trace interrupt occurs **not** as the result of a CIABR match		
Any instruction that is not listed in the *Conditions in which instructions are not traced*	The traced instruction has a storage operand A Trace interrupt occurs No CIABR match occurs $MMCR0_{PMAE}$ = 0	SIAR = the effective address of the storage operand of the traced instruction

19.2.19. Hypervisor Data Storage Interrupt

Architecture sections:

- III.6.5.16 Hypervisor Data Storage Interrupt

Scenario groups:

- Conditions for occurrence of an Hypervisor Data Storage interrupt
- Actions taken when a Hypervisor Data Storage interrupt occurs

19.2.19.1. Conditions for occurrence of a Hypervisor Data Storage interrupt

Hypervisor Data Storage interrupt condition:

Thread is not in hypervisor state or an unsupported MMU configuration has been found or the access has been prevented by a problem in partition-scoped Radix Tree translation, and either

(a) HPT translation is being performed, VPM = 0, $LPCR_{KBV}$ = 1, and a Virtual Storage Page Class Key Protection exception exists or

(b) HPT translation is being performed, the value of the expression ($\neg MSR_{DR}$) | (VPM & $PRTE_{V}$ & MSR_{DR}) is 1, and a data access cannot be performed, or

(c) Radix Tree translation is being performed and partition-scoped translation either does not complete or prevents an access from being performed

Instructions potentially causing a Hypervisor Data Storage interrupt when a virtual address cannot be translated to a real address or a guest real address cannot be translated to host real address: Load instructions, Store instructions, icbi dcbz dcbst dcbf[l]

Instructions potentially causing a Hypervisor Data Storage interrupt when referring to Write Through Required or Caching Inhibited storage: lq stq lbarx lharx lwarx ldarx lqarx stbcx. sthcx. stwcx. stdcx. stqcx.

Instructions potentially causing a Hypervisor Data Storage interrupt when referring to Caching Inhibited storage: copy paste.

Instructions potentially causing a Hypervisor Data Storage interrupt when referring to Guarded storage: lwat ldat stwat stdat

Instruction sequence	Compliance conditions	Expected result
III.6.5.16.Cond.1 Occurrence of a Hypervisor Data Storage interrupt when a HPT translation is being performed, VPM=0, $LPCR_{KBV}$=1, and a Virtual Storage Page Class Key Protection exception exists		
Any instruction that performs a data access	The thread is not in hypervisor state $LPCR_{VPM}$ = 0 $LPCR_{HR}$ = 0 $LPCR_{KBV}$ = 1 A Virtual Storage Page Class Key Protection exception exists No higher priority exception exists	A Hypervisor Data Storage interrupt occurs
III.6.5.16.Cond.2 Occurrence of a Hypervisor Data Storage interrupt when data address translation is enabled and the virtual address of any byte of the storage location specified by a Load, Store, icbi, dcbz, dcbst, or dcbf[l] instruction cannot be translated to a real address because no valid PTE was found for the VPM translation (HR=0)		
Representative of *Instructions potentially causing a Hypervisor Data Storage interrupt when a virtual address cannot be translated to a real address*	The thread is not in hypervisor state The *Hypervisor Data Storage interrupt condition* holds MSR_{DR} = 1 $LPCR_{VPM}$ = 1 $LPCR_{HR}$ = 0 The virtual address of any byte of the storage location specified by the instruction cannot be translated to a real address because no valid PTE was found for the VPM translation No higher priority exception exists	A Hypervisor Data Storage interrupt occurs
III.6.5.16.Cond.2b Occurrence of a Hypervisor Data Storage interrupt when the guest real address of any byte of the storage location specified by a Load, Store, icbi, dcbz, dcbst, or dcbf[l] instruction cannot be translated to a host real address because no valid PTE was found in the partition-scoped page table (Host Radix enabled)		
Representative of *Instructions potentially causing a Hypervisor Data Storage interrupt when a guest real address cannot be translated to a host real address*	The thread is not in hypervisor state The *Hypervisor Data Storage interrupt condition* holds $LPCR_{VPM}$ = 1 $LPCR_{HR}$ = 1 The guest real address of any byte of the storage location specified by the instruction cannot be translated to a host real address because no valid PTE was found in the partition-scoped page table No higher priority exception exists	A Hypervisor Data Storage interrupt occurs
III.6.5.16.Cond.3 Occurrence of a Hypervisor Data Storage interrupt when data address translation is disabled and the virtual address of any byte of the storage location specified by a Load, Store, icbi, dcbz, dcbst, or dcbf[l] instruction cannot be translated to a real address by means of the virtual real addressing mechanism (HR=0)		
Representative of *Instructions potentially causing a Hypervisor Data Storage interrupt when a virtual address cannot be translated to a real address*	The thread is not in hypervisor state The *Hypervisor Data Storage interrupt condition* holds	A Hypervisor Data Storage interrupt occurs

Instruction sequence	Compliance conditions	Expected result
	MSR_{DR} = 0 $LPCR_{VPM}$ = 1 $LPCR_{HR}$ = 0 The virtual address of any byte of the storage location specified by the instruction cannot be translated to a real address by means of the virtual real addressing mechanism. No higher priority exception exists	
III.6.5.16.Cond.4 Occurrence of a Hypervisor Data Storage interrupt when the effective address specified by a lq, stq, lwat, ldat, lbarx, lharx, lwarx, ldarx, lqarx, stwat, stdat, stbcx., sthcx., stwcx., stdcx., or stqcx. instruction refers to storage that is Write Through Required or Caching Inhibited		
Representative of *Instructions potentially causing a Hypervisor Data Storage interrupt when referring to Write Through Required or Caching Inhibited storage*	The thread is not in hypervisor state The *Hypervisor Data Storage interrupt condition* holds The effective address specified by the instruction refers to storage that is Write Through Required or Caching Inhibited No higher priority exception exists	A Hypervisor Data Storage interrupt occurs
III.6.5.16.Cond.5 Occurrence of a Hypervisor Data Storage interrupt when the access violates storage protection		
Any Load or Store instruction	The thread is not in hypervisor state The *Hypervisor Data Storage interrupt condition* holds The access violates storage protection No higher priority exception exists	A Hypervisor Data Storage interrupt occurs
III.6.5.16.Cond.6 Occurrence of a Hypervisor Data Storage interrupt when a Data Address Watchpoint match occurs (HR = 0)		
Any Load or Store instruction	The thread is not in hypervisor state The *Hypervisor Data Storage interrupt condition* holds $LPCR_{VPM}$ = 1 $LPCR_{HR}$ = 0 A Data Address Watchpoint match occurs No higher priority exception exists	A Hypervisor Data Storage interrupt occurs
III.6.5.16.Cond.7 (removed)		
(removed)	(removed)	(removed)
III.6.5.16.Cond.8 If the XER specifies a length of zero for an indexed Move Assist instruction, a Hypervisor Data Storage interrupt does not occur.		
Representative of *Indexed Move Assist instructions*	$XER_{57:63}$ = 0 No higher priority exception exists	No Hypervisor Data Storage interrupt occurs
III.6.5.16.Cond.9 Occurrence of a Hypervisor Data Storage interrupt when the guest real address of a page directory entry or process table entry could not be translated when HR=1		
Any instruction that performs a data access	The thread is not in hypervisor state The *Hypervisor Data Storage interrupt condition* holds $LPCR_{VPM}$ = 1	A Hypervisor Data Storage interrupt occurs

Instruction sequence	Compliance conditions	Expected result
	$LPCR_{HR}$ = 1 The guest real address of a page directory entry or process table entry cannot be translated No higher priority exception exists	
III.6.5.16.Cond.10 Occurrence of a Hypervisor Data Storage interrupt when the virtual address of a process table entry or segment table entry group could not be translated when VPM=1 and HR=0		
Any instruction that performs a data access	The thread is not in hypervisor state The *Hypervisor Data Storage interrupt condition* holds $LPCR_{VPM}$ = 1 $LPCR_{HR}$ = 0 The virtual address of a process table entry or segment table entry group cannot be translated No higher priority exception exists	A Hypervisor Data Storage interrupt occurs
III.6.5.16.Cond.11 Occurrence of a Hypervisor Data Storage interrupt when an unsupported MMU configuration is found		
Any instruction that performs a data access	The thread is not in hypervisor state The *Hypervisor Data Storage interrupt condition* holds An unsupported MMU configuration is found No higher priority exception exists	A Hypervisor Data Storage interrupt occurs
III.6.5.16.Cond.12 Occurrence of a Hypervisor Data Storage interrupt when the effective address specified by a copy or paste. instruction refers to storage that is Caching Inhibited		
P1 cpabort copy paste.	The thread is not in hypervisor state The *Hypervisor Data Storage interrupt condition* holds The effective address specified by a copy or paste. instruction refers to storage that is Caching Inhibited No higher priority exception exists	A Hypervisor Data Storage interrupt occurs
III.6.5.16.Cond.13 Occurrence of a Hypervisor Data Storage interrupt when the effective address specified by a lwat, ldat, stwat, or stdat instruction refers to storage that is Guarded		
lwat ldat stwat stdat	The thread is not in hypervisor state The *Hypervisor Data Storage interrupt condition* holds The effective address specified by a lwat, ldat, stwat, or stdat instruction refers to storage that is Guarded No higher priority exception exists	A Hypervisor Data Storage interrupt occurs
III.6.5.16.Cond.14 Occurrence of a Hypervisor Data Storage interrupt when an accelerator is specified as the source of a copy instruction, normal memory is specified at the target of a paste. instruction, or an attempt is made to access an accelerator that is not properly configured for the software's use (HR=0)		
P1 cpabort	The thread is not in hypervisor state The *Hypervisor Data Storage interrupt condition* holds	A Hypervisor Data Storage interrupt occurs

Instruction sequence	Compliance conditions	Expected result
copy paste.	$LPCR_{VPM}$ = 1 $LPCR_{HR}$ = 0 An accelerator is specified as the source of a copy instruction, normal memory is specified at the target of a paste. instruction, or an attempt is made to access an accelerator that is not properly configured for the software's use No higher priority exception exists	
III.6.5.16.Cond.15 Occurrence of a Hypervisor Data Storage interrupt when the Load Atomic or Store Atomic has an invalid function code (HR=0)		
Any Load Atomic or Store Atomic instruction	The thread is not in hypervisor state The *Hypervisor Data Storage interrupt condition* holds $LPCR_{VPM}$ = 1 $LPCR_{HR}$ = 0 The Load Atomic or Store Atomic has an invalid function code No higher priority exception exists	A Hypervisor Data Storage interrupt occurs

19.2.19.2. Actions taken when a Hypervisor Data Storage interrupt occurs

Instructions potentially causing a Hypervisor Data Storage interrupt: Load instructions, Store instructions, icbi dcbz dcbst dcbf[l] lq stq lwat ldat lbarx lharx lwarx ldarx lqarx stwat stdat stbcx. sthcx. stwcx. stdcx. stqcx.

Value of HDSISR bit 38: 1 for a Store or dcbz instruction; otherwise 0

Notes:

- Setting MSR bits is checked in Section Section 19.2.1, "Setting MSR bits" [250]. Resuming execution at the correct location is checked in Section Section 19.2.2, "Effective address of interrupt vector" [252].
- For HDSISR bits 33, 36, 37, 38, 41, 42, 43, 44, 45, 46, 60, 61: any one or more of these bits may be set to 1 in the HDSISR, if multiple Hypervisor Data Storage exceptions occur for a given effective address

Instruction sequence	Compliance conditions	Expected result
III.6.5.16.Act.1 HSRR0 and HSRR1 are set correctly when a Hypervisor Data Storage interrupt occurs		
Representative of *Instructions potentially causing a Hypervisor Data Storage interrupt*	A Hypervisor Data Storage interrupt occurs	HSRR0 = the effective address of the instruction that caused the interrupt $HSRR1_{33:36}$ = 0 $HSRR1_{42:47}$ = 0 All other bits of HSRR1 = corresponding bits in the MSR
III.6.5.16.Act.2 HDSISR is set correctly when a Hypervisor Data Storage interrupt occurs because the translation for an attempted access is not found in the Page Table		

Instruction sequence	Compliance conditions	Expected result
Representative of *Instructions potentially causing a Hypervisor Data Storage interrupt*	A Hypervisor Data Storage interrupt occurs $(MSR_{DR}) \mid (\neg MSR_{DR} \& VPM_0) = 1$ The translation is not found in the Page Table	$HDSISR_{33} = 1$ $HDSISR_{38}$ = *Value of HDSISR bit 38* HDSISR bits 32, 34:35, 39:40, 43, 47:59, 62:63 = 0
III.6.5.16.Act.3 HDSISR is set correctly when a Hypervisor Data Storage interrupt occurs because the effective address specified by the instruction refers to storage that is Write Through Required or Caching Inhibited		
Representative of *Instructions potentially causing a Hypervisor Data Storage interrupt when referring to Write Through Required or Caching Inhibited storage*	The effective address specified by the instruction refers to storage that is Write Through Required or Caching Inhibited A Hypervisor Data Storage interrupt occurs	$HDSISR_{37} = 1$ $HDSISR_{38}$ = *Value of HDSISR bit 38* HDSISR bits 32, 34:35, 39:40, 43, 47:59, 62:63 = 0
III.6.5.16.Act.4 HDSISR is set correctly when a Hypervisor Data Storage interrupt occurs because the access violates Basic Storage Protection		
Any Load or Store instruction	The access violates Basic Storage Protection A Hypervisor Data Storage interrupt occurs	$HDSISR_{36} = 1$ $HDSISR_{38}$ = *Value of HDSISR bit 38* HDSISR bits 32, 34:35, 39:40, 43, 47:59, 62:63 = 0
III.6.5.16.Act.5 HDSISR is set correctly when a Hypervisor Data Storage interrupt occurs because the access violates Virtual Page Class Key Storage Protection and $LPCR_{KBV} = 0$		
Any Load or Store instruction	The access violates Virtual Page Class Key Storage Protection A Hypervisor Data Storage interrupt occurs	$HDSISR_{42} = 1$ $HDSISR_{38}$ = *Value of HDSISR bit 38* HDSISR bits 32, 34:35, 39:40, 43, 47:59, 62:63 = 0
III.6.5.16.Act.6 HDSISR is set correctly when a Hypervisor Data Storage interrupt occurs because a Data Address Watchpoint match occurs		
Any Load or Store instruction	A Data Address Watchpoint match occurs A Hypervisor Data Storage interrupt occurs	$HDSISR_{41} = 1$ $HDSISR_{38}$ = *Value of HDSISR bit 38* HDSISR bits 32, 34:35, 39:40, 43, 47:59, 62:63 = 0
III.6.5.16.Act.7 (removed)		
(removed)	(removed)	(removed)
III.6.5.16.Act.8 HDAR is set correctly when a Hypervisor Data Storage interrupt is caused by a cache management instruction (for reasons other than a Data Address Watchpoint match)		
Representative cache management instruction that is in *Instructions potentially causing a Hypervisor Data Storage interrupt*	A Hypervisor Data Storage interrupt occurs No Data Address Watchpoint match occurs	HDAR is set to the effective address of a byte in the block that caused the exception
III.6.5.16.Act.9 HDAR is set correctly when a Hypervisor Data Storage interrupt is caused by a quadword load or store instruction (for reasons other than a Data Address Watchpoint match)		
Representative quadword load or store instruction that is in *Instructions potentially causing a Hypervisor Data Storage interrupt*	A Hypervisor Data Storage interrupt occurs No Data Address Watchpoint match occurs	HDAR is set to the effective address of a byte in the first aligned quadword for which access was attempted in the page that caused the exception
III.6.5.16.Act.10 HDAR is set correctly when a Hypervisor Data Storage interrupt is caused by a non-quadword load or store instruction (for reasons other than a Data Address Watchpoint match)		
Representative non-quadword load or store instruction that is in *Instructions*	A Hypervisor Data Storage interrupt occurs	HDAR is set to the effective address of a byte in the first aligned doubleword for

Instruction sequence	Compliance conditions	Expected result
potentially causing a Hypervisor Data Storage interrupt	No Data Address Watchpoint match occurs	which access was attempted in the page that caused the exception
III.6.5.16.Act.11 (removed)		
(removed)	(removed)	(removed)
III.6.5.16.Act.12 If a Hypervisor Data Storage interrupt occurs in 32-bit mode the high-order 32 bits of the HDAR are set to 0		
Representative of *Instructions potentially causing a Hypervisor Data Storage interrupt*	A Hypervisor Data Storage interrupt occurs 32-bit computation mode HDAR is set to a defined value	$HDAR_{32:63}$ = 0
III.6.5.16.Act.13 HDSISR is set correctly when a Hypervisor Data Storage interrupt occurs because an unsupported MMU configuration is found during the translation process		
Any Load or Store instruction	An unsupported MMU configuration is found during the translation process A Hypervisor Data Storage interrupt occurs	$HDSISR_{44}$ = 1 $HDSISR_{38}$ = *Value of HDSISR bit 38* HDSISR bits 32, 34:35, 39:40, 43, 47:59, 62:63 = 0
III.6.5.16.Act.14 HDSISR is set correctly because HR=1 and the virtual / guest real address of a page directory entry, page table entry, or process table entry could not be translated		
Any Load or Store instruction	$LPCR_{VPM}$ = 1 $LPCR_{HR}$ = 1 The virtual / guest real address of a page directory entry, page table entry, or process table entry cannot be translated A Hypervisor Data Storage interrupt occurs	$HDSISR_{46}$ = 1 $HDSISR_{38}$ = *Value of HDSISR bit 38* HDSISR bits 32, 34:35, 39:40, 43, 47:59, 62:63 = 0
III.6.5.16.Act.15 HDSISR is set correctly when a Hypervisor Data Storage interrupt occurs because HR=0, VPM=1, and the virtual address of a process table entry or segment table entry group could not be translated		
Any Load or Store instruction	$LPCR_{VPM}$ = 1 $LPCR_{HR}$ = 0 The virtual address of a process table entry or segment table entry group cannot be translated A Hypervisor Data Storage interrupt occurs	$HDSISR_{46}$ = 1 $HDSISR_{38}$ = *Value of HDSISR bit 38* HDSISR bits 32, 34:35, 39:40, 43, 47:59, 62:63 = 0
III.6.5.16.Act.16 HDSISR is set correctly when a Hypervisor Data Storage interrupt occurs because an accelerator is specified as the source of a copy instruction, normal memory is specified as the target of a paste. instruction, or an attempt is made to access an accelerator that is not properly configured for the software's use (HR=0)		
P1 cpabort copy paste.	An accelerator is specified as the source of a copy instruction, normal memory is specified as the target of a paste. instruction, or an attempt is made to access an accelerator that is not properly configured for the software's use A Hypervisor Data Storage interrupt occurs	$HDSISR_{60}$ = 1 $HDSISR_{38}$ = *Value of HDSISR bit 38* HDSISR bits 32, 34:35, 39:40, 43, 47:59, 62:63 = 0
III.6.5.16.Act.17 HDSISR is set correctly when a Hypervisor Data Storage interrupt occurs because the Load Atomic or Store Atomic has an invalid function code		
Any Load Atomic or Store Atomic instruction	The Load Atomic or Store Atomic has an invalid function code	$HDSISR_{61}$ = 1 $HDSISR_{38}$ = *Value of HDSISR bit 38*

Instruction sequence	Compliance conditions	Expected result
	A Hypervisor Data Storage interrupt occurs	HDSISR bits 32, 34:35, 39:40, 43, 47:59, 62:63 = 0
III.6.5.16.Act.18 HDAR is set correctly when a Hypervisor Data Storage interrupt occurs because a process table entry or segment table entry group virtual address cannot be translated in Paravirtualized HPT mode with VPM=1		
Any Load or Store instruction	$LPCR_{VPM}$ = 1 $LPCR_{HR}$ = 0 A process table entry or segment table entry group virtual address cannot be translated A Hypervisor Data Storage interrupt occurs	HDAR is loaded with the virtual address of the table entry or group
III.6.5.16.Act.19 ASDR is set correctly when a Hypervisor Data Storage interrupt occurs because the process table or process-scoped page directory or page table entry guest real address cannot be translated		
Any Load or Store instruction	The process table or process-scoped page directory or page table entry guest real address cannot be translated A Hypervisor Data Storage interrupt occurs	ASDR is loaded with the guest real address of the table entry
III.6.5.16.Act.19b ASDR is set correctly when a Hypervisor Data Storage interrupt occurs because the process or segment table entry virtual address cannot be translated		
Any Load or Store instruction	The process or segment table entry virtual address cannot be translated A Hypervisor Data Storage interrupt occurs	ASDR is loaded with the VSID of the table entry
III.6.5.16.Act.20 ASDR is set correctly when a Hypervisor Data Storage interrupt occurs because there is an unsupported radix tree configuration in the partition-scoped table		
Any Load or Store instruction	There is an unsupported radix tree configuration in the partition-scoped table A Hypervisor Data Storage interrupt occurs	ASDR is loaded with the guest real address of the storage element, process table entry, page directory entry, or page table entry

19.2.20. Hypervisor Instruction Storage Interrupt

Architecture sections:

- III.6.5.17 Hypervisor Instruction Storage Interrupt

Scenario groups:

- Conditions for occurrence of an Hypervisor Instruction Storage interrupt
- Actions taken when a Hypervisor Instruction Storage interrupt occurs

19.2.20.1. Conditions for occurrence of a Hypervisor Instruction Storage interrupt

Hypervisor Instruction Storage interrupt condition:

Thread is not in hypervisor state or an unsupported MMU configuration has been found or the access has been prevented by a problem in partition-scoped Radix Tree translation, and either

(a) HPT translation is being performed, the value of the expression (($\neg MSR_{IR}$) | (VPM & $PRTE_V$ & MSR_{IR})) is 1, and the next instruction to be executed cannot be fetched, or

(b) Radix Tree translation is being performed and partition-scoped translation prevents the next instruction to be executed from being fetched

Instruction sequence	Compliance conditions	Expected result
III.6.5.17.Cond.1 Occurrence of a Hypervisor Instruction Storage interrupt when the next instruction to be executed cannot be fetched because instruction address translation is enabled and the virtual address cannot be translated to a real address because no valid PTE was found for the HPT translation		
Any instruction	The thread is not in hypervisor state The *Hypervisor Instruction Storage interrupt condition* holds $MSR_{IR} = 1$ $LPCR_{VPM} = 1$ $LPCR_{HR} = 0$ The virtual address cannot be translated to a real address because no valid PTE was found for the HPT translation No higher priority exception exists	A Hypervisor Instruction Storage interrupt occurs
III.6.5.17.Cond.1b Occurrence of a Hypervisor Instruction Storage interrupt when the next instruction to be executed cannot be fetched because the guest real address of the instruction cannot be translated to a host real address because no valid PTE was found in the partition-scoped page table (HR=1)		
Any instruction	The thread is not in hypervisor state The *Hypervisor Instruction Storage interrupt condition* holds $LPCR_{VPM} = 1$ $LPCR_{HR} = 1$ The virtual address cannot be translated to a real address because no valid PTE was found in the partition-scoped page table No higher priority exception exists	A Hypervisor Instruction Storage interrupt occurs
III.6.5.17.Cond.2 Occurrence of a Hypervisor Instruction Storage interrupt when the next instruction to be executed cannot be fetched because instruction address translation is disabled and the virtual address cannot be translated to a real address by means of the virtual real addressing mechanism (HR=0)		
Any instruction	The thread is not in hypervisor state The *Hypervisor Instruction Storage interrupt condition* holds $MSR_{IR} = 0$ $LPCR_{VPM} = 1$ $LPCR_{HR} = 0$ The virtual address cannot be translated to a real address by means of the virtual real addressing mechanism No higher priority exception exists	A Hypervisor Instruction Storage interrupt occurs
III.6.5.17.Cond.3 Occurrence of a Hypervisor Instruction Storage interrupt when the next instruction to be executed cannot be fetched because the fetch access violates storage protection		
Any instruction	The thread is not in hypervisor state The *Hypervisor Instruction Storage interrupt condition* holds	A Hypervisor Instruction Storage interrupt occurs

Instruction sequence	Compliance conditions	Expected result
	The fetch access violates storage protection No higher priority exception exists	
III.6.5.17.Cond.4 Occurrence of a Hypervisor Instruction Storage interrupt when the next instruction to be executed cannot be fetched because the guest real address of a page directory entry or process table entry could not be translated when HR=1		
Any instruction	The thread is not in hypervisor state The *Hypervisor Instruction Storage interrupt condition* holds $LPCR_{VPM}$ = 1 $LPCR_{HR}$ = 1 The guest real address of a page directory entry or process table entry cannot be translated No higher priority exception exists	A Hypervisor Instruction Storage interrupt occurs
III.6.5.17.Cond.5 Occurrence of a Hypervisor Instruction Storage interrupt when the next instruction to be executed cannot be fetched because the virtual address of a process table entry or segment table entry group could not be translated when VPM=1 and HR=0		
Any instruction	The thread is not in hypervisor state The *Hypervisor Instruction Storage interrupt condition* holds $LPCR_{VPM}$ = 1 $LPCR_{HR}$ = 0 The virtual address of a process table entry or segment table entry group cannot be translated No higher priority exception exists	A Hypervisor Instruction Storage interrupt occurs
III.6.5.17.Cond.6 Occurrence of a Hypervisor Instruction Storage interrupt when the next instruction to be executed cannot be fetched because an unsupported MMU configuration is found		
Any instruction	The thread is not in hypervisor state The *Hypervisor Instruction Storage interrupt condition* holds An unsupported MMU configuration is found No higher priority exception exists	A Hypervisor Instruction Storage interrupt occurs

19.2.20.2. Actions taken when a Hypervisor Instruction Storage interrupt occurs

Notes:

- Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250]. Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252].
- For HSRR1 bits 33, 35, 36, 42, 44, 45, 46, 47: any one or more of these bits may be set to 1, if multiple Hypervisor Instruction Storage exceptions occur due to attempting to fetch a single instruction

Instruction sequence	Compliance conditions	Expected result
III.6.5.17.Act.1 HSRR0 is set correctly when a Hypervisor Instruction Storage interrupt occurs		
Any instruction	A Hypervisor Instruction Storage interrupt occurs	HSRR0 = the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present
III.6.5.17.Act.2 HSRR1 is set correctly when a Hypervisor Instruction Storage interrupt occurs because the translation for an attempted fetch access is not found in the Page Table		
Any instruction	The translation is not found in the Page Table A Hypervisor Instruction Storage interrupt occurs	$HSRR1_{33}$ = 1 HSRR1 bits 34, 43 = 0 HSRR1 bits 0:32, 37:41, 48:63 = corresponding bits in the MSR
III.6.5.17.Act.3 HSRR1 is set correctly when a Hypervisor Instruction Storage interrupt occurs because the attempted fetch access is to No-execute or Guarded storage		
Any instruction	The fetch access is to No-execute or Guarded storage A Hypervisor Instruction Storage interrupt occurs	$HSRR1_{35}$ = 1 HSRR1 bits 34, 43 = 0 HSRR1 bits 0:32, 37:41, 48:63 = corresponding bits in the MSR
III.6.5.17.Act.4 HSRR1 is set correctly when a Hypervisor Instruction Storage interrupt occurs because the attempted fetch access violates Basic Storage Protection		
Any instruction	The fetch access violates Basic Storage Protection A Hypervisor Instruction Storage interrupt occurs	$HSRR1_{36}$ = 1 HSRR1 bits 34, 43 = 0 HSRR1 bits 0:32, 37:41, 48:63 = corresponding bits in the MSR
III.6.5.17.Act.5 HSRR1 is set correctly when a Hypervisor Instruction Storage interrupt occurs because the attempted fetch access is not permitted by virtual page class key protection		
Any instruction	The fetch access violates Virtual Page Class Key Storage Protection A Hypervisor Instruction Storage interrupt occurs	$HSRR1_{42}$ = 1 HSRR1 bits 34, 43 = 0 HSRR1 bits 0:32, 37:41, 48:63 = corresponding bits in the MSR
III.6.5.17.Act.6 HSRR1 is set correctly when a Hypervisor Instruction Storage interrupt occurs when the next instruction to be executed cannot be fetched because an unsupported MMU configuration is found during the translation process		
Any instruction	An unsupported MMU configuration is found during the translation process A Hypervisor Instruction Storage interrupt occurs	$HSRR1_{44}$ = 1 HSRR1 bits 34, 43 = 0 HSRR1 bits 0:32, 37:41, 48:63 = corresponding bits in the MSR
III.6.5.17.Act.7 HSRR1 is set correctly when a Hypervisor Instruction Storage interrupt occurs when the next instruction to be executed cannot be fetched because HR=1 and the guest real address of a page directory entry, page table entry, or process table entry could not be translated		
Any instruction	$LPCR_{VPM}$ = 1 $LPCR_{HR}$ = 1 The guest real address of a page directory entry, page table entry, or process table entry cannot be translated A Hypervisor Instruction Storage interrupt occurs	$HSRR1_{46}$ = 1 HSRR1 bits 34, 43 = 0 HSRR1 bits 0:32, 37:41, 48:63 = corresponding bits in the MSR
III.6.5.17.Act.8 HSRR1 is set correctly when a Hypervisor Instruction Storage interrupt occurs when the next instruction to be executed cannot be fetched because HR=0, VPM=1, and the virtual address of a process table entry or segment table entry group could not be translated		

Instruction sequence	Compliance conditions	Expected result
Any instruction	$LPCR_{VPM}$ = 1 $LPCR_{HR}$ = 0 The virtual address of a process table entry or segment table entry group cannot be translated A Hypervisor Instruction Storage interrupt occurs	$HSRR1_{46}$ = 1 HSRR1 bits 34, 43 = 0 HSRR1 bits 0:32, 37:41, 48:63 = corresponding bits in the MSR
III.6.5.17.Act.9 HSRR1 is set correctly when a Hypervisor Instruction Storage interrupt occurs when the next instruction to be executed cannot be fetched and the operation that caused the exception was attempting to update storage		
Any instruction	The operation that causes the exception is attempting to update storage A Hypervisor Instruction Storage interrupt occurs	$HSRR1_{47}$ = 1 HSRR1 bits 34, 43 = 0 HSRR1 bits 0:32, 37:41, 48:63 = corresponding bits in the MSR
III.6.5.17.Act.10 HDAR is set correctly when a Hypervisor Instruction Storage interrupt occurs when the next instruction to be executed cannot be fetched because HR=0 and a process table entry or segment table entry group virtual address cannot be translated and VPM=1		
Any instruction	$LPCR_{VPM}$ = 1 $LPCR_{HR}$ = 0 The process table entry or segment table entry group virtual address cannot be translated A Hypervisor Instruction Storage interrupt occurs	HDAR is loaded with the virtual address of the table entry or group
III.6.5.17.Act.11 ASDR is set correctly when a Hypervisor Instruction Storage interrupt occurs when the next instruction to be executed cannot be fetched because the process table or process-scoped page directory or page table entry guest real address cannot be translated		
Any instruction	The process table or process-scoped page directory or page table entry guest real address cannot be translated A Hypervisor Instruction Storage interrupt occurs	ASDR is loaded with the guest real page address of the table entry
III.6.5.17.Act.11b ASDR is set correctly when a Hypervisor Instruction Storage interrupt occurs when the next instruction to be executed cannot be fetched because the process or segment table entry virtual address cannot be translated		
Any instruction	The process or segment table entry virtual address cannot be translated A Hypervisor Instruction Storage interrupt occurs	ASDR is loaded with the VSID of the table entry
III.6.5.17.Act.12 ASDR is set correctly when a Hypervisor Instruction Storage interrupt occurs when the next instruction to be executed cannot be fetched because there is an unsupported radix tree configuration in the partition-scoped table		
Any instruction	There is an unsupported radix tree configuration in the partition-scoped table A Hypervisor Instruction Storage interrupt occurs	ASDR is loaded with the guest real address of the instruction, process table entry, page directory entry, or page table entry

19.2.21. Hypervisor Emulation Assistance Interrupt

Architecture sections:

- III.6.5.18 Hypervisor Emulation Assistance Interrupt

Scenario groups:

- Conditions for occurrence of a Hypervisor Emulation Assistance interrupt
- Actions taken when a Hypervisor Emulation Assistance interrupt occurs

19.2.21.1. Conditions for occurrence of a Hypervisor Emulation Assistance interrupt

Instructions potentially causing a Hypervisor Emulation Assistance interrupt: illegal instructions, reserved instructions, instructions not provided by the implementation

Instruction sequence	Compliance conditions	Expected result
III.6.5.18.Cond.1 Occurrence of a Hypervisor Emulation Assistance interrupt when execution is attempted of an instruction that is illegal or reserved or not provided by the implementation		
Representative of *Instructions potentially causing a Hypervisor Emulation Assistance interrupt*		A Hypervisor Emulation Assistance interrupt occurs
III.6.5.18.Cond.2 Occurrence of a Hypervisor Emulation Assistance interrupt when an mtspr or mfspr instruction is executed when MSR_{PR} = 1 if the instruction specifies an SPR with spr_0 = 0 that is not provided by the implementation		
Representative of: mtspr, mfspr	MSR_{PR} = 1 in the encoding of the designated SPR has spr_0 = 0 The designated SPR is not provided by the implementation	A Hypervisor Emulation Assistance interrupt occurs
III.6.5.18.Cond.3 Occurrence of a Hypervisor Emulation Assistance interrupt when an mtspr or mfspr instruction is executed when MSR_{PR} = 0 if the instruction specifies SPR 0, 4, 5, or 6		
Representative of: mtspr, mfspr	MSR_{PR} = 0 The designated SPR is SPR 0, 4, 5, or 6	A Hypervisor Emulation Assistance interrupt occurs
III.6.5.18.Cond.4 Occurrence of a Hypervisor Emulation Assistance interrupt when an mtspr or mfspr instruction is executed when MSR_{PR} = 0 and $LPCR_{EVIRT}$ = 1 if the instruction specifies an SPR other than 0, 4, 5, or 6 that is not provided by the implementation		
mtspr mfspr	MSR_{PR} = 0 $LPCR_{EVIRT}$ = 1 The instruction specifies an SPR other than 0, 4, 5, or 6 that is not provided by the implementation	A Hypervisor Emulation Assistance interrupt occurs
III.6.5.18.Cond.5 Occurrence of a Hypervisor Emulation Assistance interrupt when $MSR_{HV\ PR}$ = 00 and $LPCR_{EVIRT}$ = 1 and execution is attempted of a hypervisor privileged instruction or of an mtspr or mfspr instruction that specifies an SPR that is hypervisor privileged for the operation		
Representative of *Instructions potentially causing a Hypervisor Emulation Assistance interrupt*, mtspr, mfspr	$MSR_{HV\ PR}$ = 00 $LPCR_{EVIRT}$ = 1 Execution is attempted of a hypervisor privileged instruction or of an mtspr or mfspr instruction that specifies an SPR that is hypervisor privileged for the operation	A Hypervisor Emulation Assistance interrupt occurs

19.2.21.2. Actions taken when a Hypervisor Emulation Assistance interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.18.Act.1 HSRR0 and HSRR1 are set correctly when a Hypervisor Emulation Assistance interrupt occurs when $MSR_{HV\ PR}$ = 00 and $LPCR_{EVIRT}$ = 1 and execution is attempted of a hypervisor privileged instruction or of an mtspr or mfspr instruction that specifies an SPR that is hypervisor privileged for the operation		
Representative of *Instructions potentially causing a Hypervisor Emulation Assistance interrupt*, mtspr, mfspr	$MSR_{HV\ PR}$ = 00 $LPCR_{EVIRT}$ = 1 Execution is attempted of a hypervisor privileged instruction or of an mtspr or mfspr instruction that specifies an SPR that is hypervisor privileged for the operation A Hypervisor Emulation Assistance interrupt occurs	HSRR0 = the effective address of the instruction that caused the interrupt $HSRR1_{45}$ = 1 $HSRR1_{33:36}$ = 0 $HSRR1_{42:44}$ = 0 $HSRR1_{46:47}$ = 0 All other bits of HSRR1 = corresponding bits in the MSR
III.6.5.18.Act.2 HEIR is set correctly when a Hypervisor Emulation Assistance interrupt occurs		
Representative of *Instructions potentially causing a Hypervisor Emulation Assistance interrupt*, mtspr, mfspr	A Hypervisor Emulation Assistance interrupt occurs	HEIR = a copy of the instruction that caused the interrupt

19.2.22. Hypervisor Maintenance Interrupt

Architecture sections:

- III.6.5.19 Hypervisor Maintenance Interrupt

Scenario groups:

- Conditions for occurrence of a Hypervisor Maintenance interrupt
- Actions taken when a Hypervisor Maintenance interrupt occurs

Note: this is a system-caused interrupt, not directly caused by the execution of an instruction

19.2.22.1. Conditions for occurrence of a Hypervisor Maintenance interrupt

Hypervisor Maintenance interrupt condition:

$(MSR_{EE} \mid \neg(MSR_{HV}) \mid MSR_{PR}) = 1$

Instruction sequence	Compliance conditions	Expected result
III.6.5.19.Cond.1 Occurrence of a Hypervisor Maintenance interrupt when a Hypervisor Maintenance exception exists		
(No instruction sequence specified)	The *Hypervisor Maintenance interrupt condition* holds A bit in the HMER is set to 1 The corresponding exception is enabled in the HMEER	A Hypervisor Maintenance interrupt occurs

Instruction sequence	Compliance conditions	Expected result
	No higher priority exception exists	

19.2.22.2. Actions taken when a Hypervisor Maintenance interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.19.Act.1 If the thread is in a power-saving mode when the interrupt would have occurred, the thread will exit the power-saving mode		
(No instruction sequence specified)	A Hypervisor Maintenance interrupt occurs The thread is in a power-saving mode	The thread exits the power-saving mode
III.6.5.19.Act.2 HSRR0 and HSRR1 are set correctly when a Hypervisor Maintenance interrupt occurs		
(No instruction sequence specified)	A Hypervisor Maintenance interrupt occurs	HSRR0 = the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present $HSRR1_{33:36} = 0$ $HSRR1_{42:47} = 0$ All other bits of HSRR1 = corresponding bits in the MSR
III.6.5.19.Act.3 The exception bits in the HMER are sticky		
(No instruction sequence specified)	A Hypervisor Maintenance interrupt occurs	HMER unchanged

19.2.23. Directed Hypervisor Doorbell Interrupt

Architecture sections:

- III.6.5.20 Directed Hypervisor Doorbell Interrupt

Scenario groups:

- Conditions for occurrence of a Directed Hypervisor Doorbell interrupt
- Actions taken when a Directed Hypervisor Doorbell interrupt occurs

Note: this is a system-caused interrupt, not directly caused by the execution of an instruction

19.2.23.1. Conditions for occurrence of a Directed Hypervisor Doorbell interrupt

Directed Hypervisor Doorbell interrupt condition:

$(MSR_{EE} \mid \neg(MSR_{HV}) \mid MSR_{PR}) = 1$

Instruction sequence	Compliance conditions	Expected result
III.6.5.20.Cond.1 Occurrence of a Directed Hypervisor Doorbell interrupt when a Directed Hypervisor Doorbell exception exists		

Instruction sequence	Compliance conditions	Expected result
(No instruction sequence specified)	A Directed Hypervisor Doorbell exception exists MSR_{EE} = 1 No higher priority exception exists	A Directed Hypervisor Doorbell interrupt occurs

19.2.23.2. Actions taken when a Directed Hypervisor Doorbell interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.20.Act.1 SRR0 and SRR1 are set correctly when a Directed Hypervisor Doorbell interrupt occurs		
(No instruction sequence specified)	A Directed Hypervisor Doorbell interrupt occurs	HSRR0 = the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present $HSRR1_{33:36}$ = 0 $HSRR1_{42:47}$ = 0 All other bits of HSRR1 = corresponding bits in the MSR

19.2.24. Hypervisor Virtualization Interrupt

Architecture sections:

- III.6.5.21 Hypervisor Virtualization Interrupt

Scenario groups:

- Conditions for occurrence of a Hypervisor Virtualization interrupt
- Actions taken when a Hypervisor Virtualization interrupt occurs

Note: this is a system-caused interrupt, not directly caused by the execution of an instruction

19.2.24.1. Conditions for occurrence of a Hypervisor Virtualization interrupt

Hypervisor Virtualization interrupt condition:

$((MSR_{EE} \mid \neg(MSR_{HV}) \mid MSR_{PR}) \,\&\, HVICE) = 1$

Instruction sequence	Compliance conditions	Expected result
III.6.5.21.Cond.1 Occurrence of a Hypervisor Virtualization interrupt when a Hypervisor Virtualization exception exists		
(No instruction sequence specified)	A Hypervisor Virtualization exception exists MSR_{EE} = 1 $LPCR_{HVICE}$ = 1	A Hypervisor Virtualization interrupt occurs

Instruction sequence	Compliance conditions	Expected result
	No higher priority exception exists	

19.2.24.2. Actions taken when a Hypervisor Virtualization interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.21.Act.1 HSRR0 and HSRR1 are set correctly when a Hypervisor Virtualization interrupt occurs		
(No instruction sequence specified)	A Hypervisor Virtualization interrupt occurs	HSRR0 = the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present $HSRR1_{33:36} = 0$ $HSRR1_{42:47} = 0$ All other bits of HSRR1 = corresponding bits in the MSR

19.2.25. Performance Monitor Interrupt

Architecture sections:

- III.6.5.22 Performance Monitor Interrupt

Scenario groups:

- Conditions for occurrence of a Performance Monitor interrupt
- Actions taken when a Performance Monitor interrupt occurs

Note: this is a system-caused interrupt, not directly caused by the execution of an instruction

19.2.25.1. Conditions for occurrence of a Performance Monitor interrupt

Performance Monitor interrupt condition:

Event-based branches are disabled ($MMCR0_{EBE} = 0$), and $MSR_{EE} = 1$, and either $HFSCR_{PM} = 1$ or the thread is in hypervisor state

Instruction sequence	Compliance conditions	Expected result
III.6.5.22.Cond.1 Occurrence of a Performance Monitor interrupt when a Performance Monitor exception exists		
(No instruction sequence specified)	A Performance Monitor exception exists Event-based branches are disabled $MMCR0_{EBE} = 0$ $MSR_{EE} = 1$ Either $HFSCR_{PM} = 1$ or the thread is in hypervisor state	A Performance Monitor interrupt occurs

Instruction sequence	Compliance conditions	Expected result
	No higher priority exception exists	

19.2.25.2. Actions taken when a Performance Monitor interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.22.Act.1 SRR0 and SRR1 are set correctly when a Performance Monitor interrupt occurs		
(No instruction sequence specified)	A Performance Monitor interrupt occurs	SRR0 = the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present $SRR1_{33:36}$ and $SRR1_{42:47}$ are reserved All other bits of SRR1 = corresponding bits in the MSR
III.6.5.22.Act.2 If multiple Performance Monitor exceptions occur before the first causes a Performance Monitor interrupt, the interrupt reflects the most recent Performance Monitor exception		
(No instruction sequence specified)	Multiple Performance Monitor exceptions occur before the first causes a Performance Monitor interrupt	SRR0, SRR1, MSR reflect the most recent Performance Monitor exception

19.2.26. Vector Unavailable Interrupt

Architecture sections:

- III.6.5.23 Vector Unavailable Interrupt

Scenario groups:

- Conditions for occurrence of a Vector Unavailable interrupt
- Actions taken when a Vector Unavailable interrupt occurs

19.2.26.1. Conditions for occurrence of a Vector Unavailable interrupt

Instruction sequence	Compliance conditions	Expected result
III.6.5.23.Cond.1 Occurrence of a Vector Unavailable interrupt when an attempt is made to execute a Vector instruction and $MSR_{VEC} = 0$		
Any Vector instruction	$MSR_{VEC} = 0$ No higher priority exception exists	A Vector Unavailable interrupt occurs

19.2.26.2. Actions taken when a Vector Unavailable interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.23.Act.1 SRR0 and SRR1 are set correctly when a Vector Unavailable interrupt occurs		
Any Vector instruction	A Vector Unavailable interrupt occurs	SRR0 = the effective address of the instruction that caused the interrupt $SRR1_{33:36}$ = 0 $SRR1_{42:47}$ = 0 All other bits of SRR1 = corresponding bits in the MSR

19.2.27. VSX Unavailable Interrupt

Architecture sections:

- III.6.5.24 VSX Unavailable Interrupt

Scenario groups:

- Conditions for occurrence of a VSX Unavailable interrupt
- Actions taken when a VSX Unavailable interrupt occurs

19.2.27.1. Conditions for occurrence of a VSX Unavailable interrupt

Instruction sequence	Compliance conditions	Expected result
III.6.5.24.Cond.1 Occurrence of a VSX Unavailable interrupt when an attempt is made to execute a VSX instruction and MSR_{VSX} = 0		
Any VSX instruction	MSR_{VSX} = 0 No higher priority exception exists	A VSX Unavailable interrupt occurs

19.2.27.2. Actions taken when a VSX Unavailable interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.24.Act.1 SRR0 and SRR1 are set correctly when a VSX Unavailable interrupt occurs		
Any VSX instruction	A VSX Unavailable interrupt occurs	SRR0 = the effective address of the instruction that caused the interrupt $SRR1_{33:36}$ = 0 $SRR1_{42:47}$ = 0 All other bits of SRR1 = corresponding bits in the MSR

19.2.28. Facility Unavailable Interrupt

Architecture sections:

- III.6.5.25 Facility Unavailable Interrupt

- III.6.2.11 Facility Status and Control Register

Scenario groups:

- Conditions for occurrence of a Facility Unavailable interrupt
- Actions taken when a Facility Unavailable interrupt occurs

19.2.28.1. Conditions for occurrence of a Facility Unavailable interrupt

Instruction sequence	Compliance conditions	Expected result
III.6.5.25.Cond.1 Occurrence of a Facility Unavailable interrupt when a facility is accessed in problem state when it has been made unavailable by the FSCR		
See Section Section 19.1.1, “FSCR Facility Enable (FE)” [247] for scenarios that check this case.		
III.6.5.25.Cond.2 Occurrence of a Facility Unavailable interrupt when a Performance Monitor register is accessed or a clrbhrb or mfbhrbe instruction is executed in problem state when it has been made unavailable by MMCR0		
Any instruction that accesses a Performance Monitor register, or clrbhrb or mfbhrbe	The program is in problem state The designated register has been made unavailable by MMCR0 No higher priority exception exists	A Facility Unavailable interrupt occurs
III.6.5.25.Cond.3 Occurrence of a Facility Unavailable interrupt when the Transactional Memory Facility is accessed in any privilege state when it has been made unavailable by MSR_{TM}		
Any Transactional Memory instruction	MSR_{TM} = 0 No higher priority exception exists	A Facility Unavailable interrupt occurs

19.2.28.2. Actions taken when a Facility Unavailable interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.25.Act.1 SRR0 and SRR1 are set correctly when a Facility Unavailable interrupt occurs		
Any instruction that potentially causes a Facility Unavailable interrupt	A Facility Unavailable interrupt occurs	SRR0 = the effective address of the instruction that caused the interrupt $SRR1_{33:36}$ = 0 $SRR1_{42:47}$ = 0 All other bits of SRR1 = corresponding bits in the MSR
III.6.5.25.Act.2 When a Facility Unavailable interrupt occurs, the IC field of the FSCR contains a binary number indicating the facility for which access was attempted.		
Any instruction that potentially causes a Facility Unavailable interrupt	A Facility Unavailable interrupt occurs	$FSCR_{0:7}$ = the correct value indicating the facility for which access was attempted (as detailed in Section III. 6.2.11)

19.2.29. Hypervisor Facility Unavailable Interrupt

Architecture sections:

- III.6.5.26 Hypervisor Facility Unavailable Interrupt
- III.6.2.12 Hypervisor Facility Status and Control Register

Scenario groups:

- Conditions for occurrence of a Hypervisor Facility Unavailable interrupt
- Actions taken when a Hypervisor Facility Unavailable interrupt occurs

19.2.29.1. Conditions for occurrence of a Hypervisor Facility Unavailable interrupt

Instruction sequence	Compliance conditions	Expected result
III.6.5.26.Cond.1 Occurrence of a Hypervisor Facility Unavailable interrupt when a facility is accessed in problem or privileged non-hypervisor states when it has been made unavailable by the HFSCR		
See Section Section 19.1.2, “HFSCR Facility Enable (FE)” [248] for scenarios that check this case.		
III.6.5.26.Cond.2 Occurrence of a Hypervisor Facility Unavailable interrupt when the stop instruction is executed in privileged non hypervisor state when any of the following conditions exist ($PSSCR_{EC}$ = 1, $PSSCR_{ESL}$ = 1, $PSSCR_{MTL}$ > $PSSCR_{PSLL}$, $PSSCR_{RL}$ > $PSSCR_{PSLL}$)		
stop	$MSR_{HV\ PR}$ = 00 Any of the following conditions exist ($PSSCR_{EC}$ = 1, $PSSCR_{ESL}$ = 1, $PSSCR_{MTL}$ > $PSSCR_{PSLL}$, $PSSCR_{RL}$ > $PSSCR_{PSLL}$)	A Hypervisor Facility Unavailable interrupt occurs

19.2.29.2. Actions taken when a Hypervisor Facility Unavailable interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.26.Act.1 HSRR0 and HSRR1 are set correctly when a Hypervisor Facility Unavailable interrupt occurs		
Any instruction that potentially causes a Hypervisor Facility Unavailable interrupt	A Hypervisor Facility Unavailable interrupt occurs	HSRR0 = the effective address of the instruction that caused the interrupt $HSRR1_{33:36}$ = 0 $HSRR1_{42:47}$ = 0 All other bits of HSRR1 = corresponding bits in the MSR
III.6.5.26.Act.2 When a Hypervisor Facility Unavailable interrupt occurs, the facility that was accessed is indicated in the IC field of the HFSCR		
Any instruction that potentially causes a Hypervisor Facility Unavailable interrupt	A Hypervisor Facility Unavailable interrupt occurs	$HFSCR_{0:7}$ = the correct value indicating the facility for which access was attempted (as detailed in Section III. 6.2.12)

19.2.30. System Call Vectored Interrupt

Architecture sections:

- III.6.5.27 System Call Vectored Interrupt

Scenario groups:

- Conditions for occurrence of a System Call Vectored interrupt
- Actions taken when a System Call Vectored interrupt occurs

19.2.30.1. Conditions for occurrence of a System Call Vectored interrupt

Instruction sequence	Compliance conditions	Expected result
III.6.5.27.Cond.1 Occurrence of a System Call Vectored interrupt when a scv instruction is executed		
scv	No higher priority exception exists	A System Call Vectored interrupt occurs

19.2.30.2. Actions taken when a System Call Vectored interrupt occurs

Note: Setting MSR bits is checked in Section Section 19.2.1, “Setting MSR bits” [250] . Resuming execution at the correct location is checked in Section Section 19.2.2, “Effective address of interrupt vector” [252] .

Instruction sequence	Compliance conditions	Expected result
III.6.5.27.Act.1 LR, CTR, and MSR are set correctly when a System Call Vectored interrupt occurs		
scv	A System Call Vectored interrupt occurs	LR = the effective address of the instruction following the System Call Vectored instruction $CTR_{33:36}$ = undefined $CTR_{42:47}$ = undefined All other bits of CTR = corresponding bits in the MSR

19.3. Interrupt priorities

Architecture sections:

- III.6.7 Exception Ordering
- III.6.9 Interrupt Priorities
- III.6.10 Relationship of Event-Based Branches to Interrupts

Scenario groups:

- Combinations of unordered and ordered interrupts
- Combinations of interrupts caused by the same instruction
- Combinations of interrupts occurring during instruction fetch and instruction execution
- Combinations of instruction-caused interrupts with imprecise or system-caused interrupts
- Combination of event-based branch exceptions with interrupts

Instruction group: one of the instruction groups A-J listed in Section III.6.9, under item 3 Instruction-Caused and Precise.

- For example, *instruction group A* is Fixed-Loads and Stores

Interrupts caused by the instruction group: the interrupts listed for the given *instruction group* in Section III.6.9

- For example, Hypervisor Facility Unavailable is one of the *interrupts caused by instruction group A*

Interrupt is not disabled: either the thread is not in power-saving mode and the interrupt is not disabled, or the thread is in power-saving mode and the exception is enabled to cause exit from the mode

19.3.1. Combinations of unordered and ordered interrupts

Ordered exceptions: exceptions listed in section **III.6.7.2 Ordered Exceptions**

Guideline: each scenario in this group should be tested for a representative *ordered exception*, denoted $I_{ordered}$

Instruction causing $I_{ordered}$: any instruction that causes the exception $I_{ordered}$.

Instruction sequence	Compliance conditions	Expected result
III.6.9.Unordered.1 System Reset exception has the highest priority of all exceptions		
Instruction causing $I_{ordered}$	Exception $I_{ordered}$ is created System Reset exception occurs	A System Reset interrupt is generated All other exceptions are ignored
III.6.9.Unordered.2 Machine Check exception (except for those caused by an invalid attempt to access an accelerator) has higher priority than all of the ordered exceptions		
Instruction causing $I_{ordered}$	Exception $I_{ordered}$ is created Machine Check exception (not due to an invalid attempt to access an accelerator) occurs No System Reset exception occurs	A Machine Check interrupt is generated All other exceptions are ignored
III.6.9.Unordered.3 System Reset has higher priority than Machine Check		
(No instructions required)	System Reset exception occurs Machine Check exception occurs	A System Reset interrupt is generated All other exceptions are ignored

19.3.2. Combinations of interrupts caused by the same instruction

Guidelines:

- The scenario in this group should be tested for every *instruction group*, from *instruction group A* to *instruction group J*.
- For each instruction group, the scenario should be tested for every pair I_{high} , I_{low} of *interrupts caused by the instruction group*, such that I_{high} has higher priority than I_{low}

Instruction sequence	Compliance conditions	Expected result
III.6.9.Instr.1 For exceptions caused by an instruction in *instruction group*, interrupts are generated according to priority		
Representative of *instruction group*	Exception I_{high} is created Exception I_{low} is created	Interrupt I_{high} is generated

Instruction sequence	Compliance conditions	Expected result
	Interrupt I_{high} is not disabled Interrupt I_{low} is not disabled	

19.3.3. Combinations of interrupts occurring during instruction fetch and instruction execution

Interrupts occurring during instruction fetch: Instruction Storage, Instruction Segment, Hypervisor Instruction Storage (the interrupts listed in item K in Section III.6.9)

Guidelines:

- The scenario in this group should be tested for every *instruction group*, from *instruction group A* to *instruction group J*.
- For each *instruction group*, the scenario should be tested for a representative interrupt $I_{execute}$ in *interrupts caused by the instruction group*
- The interrupt I_{fetch} is a representative of *interrupts occurring during instruction fetch*

Instruction sequence	Compliance conditions	Expected result
III.6.9.Fetch.1 Exceptions occurring during instruction execution have higher priority than exceptions occurring during instruction fetch		
Representative of *instruction group*	Exception $I_{execute}$ is created Exception I_{fetch} is created Interrupt $I_{execute}$ is not disabled Interrupt I_{fetch} is not disabled	Interrupt $I_{execute}$ is generated

19.3.4. Combinations of instruction-caused interrupts with imprecise or system-caused interrupts

Interrupt group: one of the interrupt groups 3-6 listed in Section III.6.9

- For example, *interrupt group 4* is Program Imprecise Mode Floating-Point Enabled Exception
- Note that *interrupt group 3* includes all of the interrupts listed in items A-K in Section III.6.9

Guidelines:

- The scenario in this group should be tested for every pair of groups G_{high} , G_{low} , from *interrupt group 3* to *interrupt group 6*, such that G_{high} has higher priority than G_{low}
- For every pair G_{high} , G_{low} , the scenario should be tested for representative interrupts I_{high} in G_{high} and I_{low} in G_{low}

III.6.9.Imprecise.1 For instruction-caused exceptions and imprecise or system-caused exceptions, interrupts are generated according to priority		
Instruction causing I_{high} *and* I_{low}	Exception I_{high} is created Exception I_{low} is created Interrupt I_{high} is not disabled Interrupt I_{low} is not disabled	Interrupt I_{high} is generated

19.3.5. Combination of event-based branch exceptions with interrupts

Guideline: the scenario in this group should be tested for a representative *ordered exception*, denoted $I_{ordered}$, and a representative event-based branch exception, denoted I_{ebb}

Instruction sequence	Compliance conditions	Expected result
III.6.9.Event.1 Event-based exceptions have a priority lower than all exceptions that cause interrupts		
Instruction causing $I_{ordered}$ *and* I_{ebb}	Exception $I_{ordered}$ is created Exception I_{ebb} is created	Interrupt $I_{ordered}$ is generated

20. Timer Facilities (Chapter III.7)

Table of Contents

20.1. Time Base and Virtual Time Base

Architecture sections:

- III.7.2 Time Base (TB)
- III.7.3 Virtual Time Base (TB)

Scenario groups:

- Writing the Time Base
- Incrementing the Virtual Time Base

20.1.1. Writing the Time Base

Instruction sequence for writing the Time Base using TBL and TBU: the instruction sequence specified in section III.7.2.1 of the architecture specification, which uses the mttbu and mttbl extended mnemonics

Instruction sequence for writing the Time Base using TBU40: the instruction sequence specified in section III.7.2.1 of the architecture specification, which uses the mttbu40 extended mnemonic.

Guideline: Each scenario in this group should be tested in both 64-bit and 32-bit computation mode.

Instruction sequence	Compliance conditions	Expected result
III.7.2.TB.1 When a mtspr instruction is executed specifying TBU40, TBU, or TBL, the associated field of the Time Base is altered and the remaining bits of the Time Base are not affected.		
mtspr	The designated SPR is TBU40, TBU, or TBL	The associated field of the Time Base is altered and the remaining bits of the Time Base are not affected
III.7.2.TB.2 Writing the Time Base using TBL and TBU		
Instruction sequence for writing the Time Base using TBL and TBU		Correct value in the Time Base
III.7.2.TB.3 Writing the Time Base using TBU40		
Instruction sequence for writing the Time Base using TBU40		Correct value in the Time Base

20.1.2. Incrementing the Virtual Time Base

Instruction sequence	Compliance conditions	Expected result
III.7.2.VTB.1 Virtual Time Base increments at the same rate as the Time Base		

Instruction sequence	Compliance conditions	Expected result
Any sequence of instructions	Time Base increments during execution of the instruction sequence VTB != 0xFFFF_FFFF_FFFF_FFFF before the increment	VTB is incremented
III.7.2.VTB.2 When the value of Virtual Time Base becomes 0xFFFF_FFFF_FFFF_FFFF (2^{64}- 1), at the next increment its value becomes 0x0000_0000_0000_0000.		
Any sequence of instructions	Time Base increments during execution of the instruction sequence VTB = 0xFFFF_FFFF_FFFF_FFFF before the increment	(VTB) = 0x0000_0000_0000_0000

20.2. Timer facility registers

Architecture sections:

- III.7.4 Decrementer
- III.7.5 Hypervisor Decrementer
- III.7.8 Instruction Counter

Scenario groups:

- Incrementing the Decrementer and Hypervisor Decrementer
- Incrementing the Instruction Counter

20.2.1. Incrementing the Decrementer and Hypervisor Decrementer

Instruction sequence	Compliance conditions	Expected result
III.7.4.Dec.1 The Decrementer is driven at the same frequency as the Time Base (Large Decrementer mode disabled)		
Any sequence of instructions	Time Base increments during execution of the instruction sequence $LPCR_{LD}$ = 0 DEC != 0x0000_0000 before the increment	DEC is decremented
III.7.4.Dec.2 When the Decrementer value becomes 0x0000_0000, at the next decrement its value becomes 0xFFFF_FFFF (Large Decrementer mode disabled)		
Any sequence of instructions	Time Base increments during execution of the instruction sequence $LPCR_{LD}$ = 0 DEC = 0x0000_0000 before the increment	(DEC) = 0xFFFF_FFFF
III.7.4.Dec.3 When the contents of DEC_{32} change from 0 to 1, a Decrementer exception will come into existence within a reasonable period of time (Large Decrementer mode disabled)		
Any sequence of instructions	$LPCR_{LD}$ = 0 The contents of DEC_{32} change from 0 to 1	A Decrementer exception occurs
III.7.4.Dec.4 The Hypervisor Decrementer is driven at the same frequency as the Time Base		
Any sequence of instructions	Time Base increments during execution of the instruction sequence HDEC != 0x0000_0000_0000_0000 before the increment	HDEC is decremented

Instruction sequence	Compliance conditions	Expected result
III.7.4.Dec.5 When the Hypervisor Decrementer value becomes 0x0000_0000_0000_0000, at the next decrement its value becomes 0xFFFF_FFFF_FFFF_FFFF		
Any sequence of instructions	Time Base increments during execution of the instruction sequence HDEC = 0x0000_0000_0000_0000 before the increment	(HDEC) = 0xFFFF_FFFF_FFFF_FFFF
III.7.4.Dec.6 When the contents of $HDEC_0$ change from 0 to 1 and the thread is not in a power-saving mode, a Hypervisor Decrementer exception will come into existence within a reasonable period of time		
Any sequence of instructions	The contents of $HDEC_0$ change from 0 to 1 The thread is not in power-saving mode	A Hypervisor Decrementer exception occurs
III.7.4.Dec.7 The Decrementer is driven at the same frequency as the Time Base (Large Decrementer mode enabled)		
Any sequence of instructions	Time Base increments during execution of the instruction sequence $LPCR_{LD}$ = 1 DEC != 0x0000_0000_0000_0000 before the increment	DEC is decremented
III.7.4.Dec.8 When the Decrementer value becomes 0x0000_0000_0000_0000, at the next decrement its value becomes 0xFFFF_FFFF_FFFF_FFFF (Large Decrementer mode enabled)		
Any sequence of instructions	Time Base increments during execution of the instruction sequence $LPCR_{LD}$ = 1 DEC = 0x0000_0000_0000_0000 before the increment	(DEC) = 0xFFFF_FFFF_FFFF_FFFF
III.7.4.Dec.9 When the contents of DEC_0 change from 0 to 1, a Decrementer exception will come into existence within a reasonable period of time (Large Decrementer mode enabled)		
Any sequence of instructions	$LPCR_{LD}$ = 1 The contents of DEC_0 change from 0 to 1	A Decrementer exception occurs

20.2.2. Incrementing the Instruction Counter

Instruction sequence	Compliance conditions	Expected result
III.7.8.IC.1 The Instruction Counter counts the number of instructions that the thread has completed		
One instruction that completes		IC is incremented

21. Debug Facilities (Chapter III.8)

Table of Contents

21.1. Debug facility mechanisms

Architecture sections:

- III.8.2 Come-From Address Register
- III.8.3 Completed Instruction Address Breakpoint
- III.8.4 Data Address Watchpoint

Scenario groups:

- Setting CFAR
- CIEA and DAW

21.1.1. Setting CFAR

Instruction sequence	Compliance conditions	Expected result
III.8.2.CFAR.1 When an rfebb, rfid, rfscv instruction is executed, CFAR is set to the effective address of the instruction		
Representative of: rfebb rfid rfscv		(CFAR) = effective address of the instruction
III.8.2.CFAR.2 When a Branch instruction is executed and the branch is taken, the register is set to the effective address of an instruction in the instruction cache block containing the Branch instruction		
Any non-B-form Branch instruction A context synchronizing operation	The branch is taken	(CFAR) = the effective address of an instruction in the instruction cache block containing the Branch instruction

21.1.2. CIEA and DAW

Conditions for a Data Address Watchpoint match: the conditions specified in section III.8.4 of the architecture specification.

Instruction sequence	Compliance conditions	Observability Preconditions	Expected result
III.8.3.CIEA.1 A Completed Instruction Address Breakpoint match causes a Trace exception 64-bit mode			
One instruction that completes	The completed instruction address is equal to $CIEA_{0:61}$ || 0b00 The thread run level matches that specified in RLM No higher priority interrupt occurs from the completion of the instruction 64-bit computation mode		A Trace exception occurs

Instruction sequence	Compliance conditions	Observability Preconditions	Expected result
III.8.3.CIEA.2 A Completed Instruction Address Breakpoint match causes a Trace exception 32-bit mode			
One instruction that completes	The completed instruction address is equal to $CIEA_{0:61}$ || 0b00 The thread run level matches that specified in RLM No higher priority interrupt occurs from the completion of the instruction 32-bit computation mode	High-order 32 bits of the EA are not all zeros	A Trace exception occurs
III.8.3.CIEA.3 A Data Address Watchpoint match causes a Data Storage exception 64-bit mode			
One Load or Store instruction (except Store Conditional or dcbz)	The *Conditions for a Data Address Watchpoint match* occur 64-bit computation mode		A Data Storage exception occurs
III.8.3.CIEA.4 A Data Address Watchpoint match causes a Data Storage exception 64-bit mode			
One Load or Store instruction (except Store Conditional or dcbz)	The *Conditions for a Data Address Watchpoint match* occur 32-bit computation mode	High-order 32 bits of the EA are not all zeros	A Data Storage exception occurs
III.8.3.CIEA.5 (removed)			
(removed)	(removed)	(removed)	(removed)
III.8.3.CIEA.6 A Data Address Watchpoint match without modifying the storage operand			
A Store instruction that causes an atomic access	The *Conditions for a Data Address Watchpoint match* occur		A Data Storage exception occurs The storage operand is not modified
III.8.3.CIEA.7 Cache Management instructions other than dcbz never cause a match			
Any Cache Management instruction other than dcbz	The *Conditions for a Data Address Watchpoint match* occur		No Data Storage exception occurs

22. Performance Monitor Facilities (Chapter III.9)

Execution of mtspr and mfspr for the Performance Monitor facility registers (PMC1 - PMC6, MMCR0, MMCR1, MMCR2, MMCRA, SIAR, SDAR, and SIER) is tested in Section Section 17.5.1, “Correct behavior of mtspr and mfspr” [204] of this document.

23. Processor Control (Chapter III.10)

Architecture sections:

- III.10 Processor Control
- III.5.9.2 Synchronize Instruction

Instruction sequence	Compliance conditions	Expected result
III.10.PC.1 P1 performs a message send instruction that is accepted with Broadcast = 00 in which the message is sent to the thread for which $PIR_{44:63}$ is equal to the value of the PROCIDTAG field in the message payload		
P1 msgsnd	MSR_{HV} = 1 The interrupt is enabled (MSR_{EE} = 1) The RB operand defines a message of type 5 The RB operand defines a message payload with Broadcast = 00 The PROCIDTAG field of the message payload (defined by the RB operand) is equal to the $PIR_{44:63}$ of hypervisor thread number of t	The thread t receives the Directed Hypervisor Doorbell message A Directed Hypervisor Doorbell interrupt is generated in thread t
III.10.PC.2 P1 performs a message send instruction that is accepted with Broadcast = 01 in which the message is sent to all threads on the same sub-processor as the thread for which $PIR_{44:63}$ is equal to the value of the PROCIDTAG field in the message payload		
P1 msgsnd	MSR_{HV} = 1 The interrupt is enabled (MSR_{EE} = 1) The RB operand defines a message of type 5 The RB operand defines a message payload with Broadcast = 01 The PROCIDTAG field of the message payload (defined by the RB operand) is equal to the $PIR_{44:63}$ of hypervisor thread number of t	The thread t and all threads on the same sub-processor as t receive the Directed Hypervisor Doorbell message A Directed Hypervisor Doorbell interrupt is generated in thread t and in all threads on the same sub-processor as t
III.10.PC.2b P1 performs a message send instruction that is accepted with Broadcast = 10 in which the message is sent to all threads on the same multi-threaded processor as the thread for which $PIR_{44:63}$ is equal to the value of the PROCIDTAG field in the message payload		
P1 msgsnd	MSR_{HV} = 1 The interrupt is enabled (MSR_{EE} = 1) The RB operand defines a message of type 5 The RB operand defines a message payload with Broadcast = 10 The PROCIDTAG field of the message payload (defined by the RB operand) is equal to the $PIR_{44:63}$ of hypervisor thread number of t	The thread t and all threads on the same multi-threaded processor as t receive the Directed Hypervisor Doorbell message A Directed Hypervisor Doorbell interrupt is generated in thread t and in all threads on the same multi-threaded processor as t
III.10.PC.3 P1 performs a message send privileged instruction that is accepted		
P1 msgsndp	MSR_{HV} = 0 MSR_{PR} = 0	The thread t receives the Directed Privileged Doorbell message

Instruction sequence	Compliance conditions	Expected result
	The interrupt is enabled (MSR_{EE} = 1) The RB operand defines a legal thread number, t, that is less than or equal to the maximum privileged thread number defined for P1 The RB operand defines a message of type 5 The TIRTAG field of the message payload is equal to the privileged thread number of t	The bit corresponding to thread t in DPDES is set to 1 A Directed Privileged Doorbell interrupt is generated in thread t
III.10.PC.4 (removed)		
(removed)	(removed)	(removed)
III.10.PC.5 P1 performs a message send instruction with the interrupt disabled followed by a message clear instruction		
P1 msgsnd msgclr	MSR_{HV} = 1 The interrupt is disabled (MSR_{EE} = 0) The RB operand of the msgsnd defines a message of type 5, defines a message payload with Broadcast = 00, and the PROCIDTAG field of the message payload is equal to the $PIR_{44:63}$ of hypervisor thread number of t The RB operand of the msgclr defines a message of type 5 and defines the hypervisor thread number of t	The thread t receives the Directed Hypervisor Doorbell message Clear the Directed Hypervisor Doorbell exception that exists on thread t A Directed Hypervisor Doorbell interrupt is not generated in thread t
III.10.PC.6 P1 performs a message send privileged instruction with the interrupt disabled followed by a message clear privileged instruction		
P1 msgsndp msgclrp	MSR_{HV} = 0 MSR_{PR} = 0 The interrupt is disabled (MSR_{EE} = 0) The RB operand of the msgsndp defines a legal thread number, t, that is less than or equal to the maximum privileged thread number defined for P1 The RB operand of the msgsndp defines a message of type 5 and the TIRTAG field of the message payload is equal to the privileged thread number of t The RB operand of the msgclrp defines a message of type 5 and defines the privileged thread number of t	The thread t receives the Directed Privileged Doorbell message Clear the Directed Privileged Doorbell exception that exists on thread t and the bit corresponding to thread t in DPDES is set to 0 A Directed Privileged Doorbell interrupt is not generated in thread t
III.10.PC.7 P1 thread t0 performs store, sync, message send to P2 thread T1; P2 thread T1 has a Directed Hypervisor Doorbell Interrupt, message send to P3 thread T2; P3 thread T2 has a Directed Hypervisor Doorbell Interrupt, message sync, load		
P1 T0 std r1,X sync msgsnd to P2 T1 P2 T1 (Directed Hypervisor Doorbell Interrupt) msgsnd to P3 T2 P3 T2 (Directed Hypervisor Doorbell Interrupt) msgsync ld r1,X	MSR_{HV} = 1 The interrupt is enabled (MSR_{EE} = 1) On P1 T0, register r1 is set to contain the value 1 The RB operand of the P1 T0 msgsnd defines a message of type 5, defines a message payload with Broadcast = 00, and the PROCIDTAG field of the	On P3 T2, register r1 contains the value 1

Instruction sequence	Compliance conditions	Expected result
	message payload is equal to the $PIR_{44:63}$ of hypervisor thread number of P2 T1 The RB operand of the P2 T1 msgsnd defines a message of type 5, defines a message payload with Broadcast = 00, and the PROCIDTAG field of the message payload is equal to the $PIR_{44:63}$ of hypervisor thread number of P3 T2	

Appendix A. OpenPOWER Foundation overview

The OpenPOWER Foundation was founded in 2013 as an open technical membership organization that will enable data centers to rethink their approach to technology. Member companies are enabled to customize POWER CPU processors and system platforms for optimization and innovation for their business needs. These innovations include custom systems for large or warehouse scale data centers, workload acceleration through GPU, FPGA or advanced I/O, platform optimization for SW appliances, or advanced hardware technology exploitation. OpenPOWER members are actively pursing all of these innovations and more and welcome all parties to join in moving the state of the art of OpenPOWER systems design forward.

To learn more about the OpenPOWER Foundation, visit the organization website at openpowerfoundation.org.

A.1. Foundation documentation

Key foundation documents include:

- *Bylaws of OpenPOWER Foundation*
- *OpenPOWER Foundation Intellectual Property Rights (IPR) Policy*
- *OpenPOWER Foundation Membership Agreement*
- *OpenPOWER Anti-Trust Guidelines*

More information about the foundation governance can be found at openpowerfoundation.org/about-us/governance.

A.2. Technical resources

Development resouces fall into the following general categories:

- Technical Steering Committee
- Foundation work groups
- OpenPOWER Ready documentation, products, and certification criteria
- Resource Catalog

  To find all OpenPOWER resources of the following types, select the specificied combination of **Resource Type**/**Main Category**/**Sub-category** in the Resource Catalog:

Specifications	`Developer Resources` / `OpenPOWER Documents` / `Specifications`
Work Group Notes	`Developer Resources` / `OpenPOWER Documents` / `Work Group Notes`

Cloud development virtual machines	`Developer Resources` / `Software Developer Cloud Resources` / <empty>
Developer Tools	`Developer Resources` / `Developer Tools` / <empty>

Note

Use the **Search** field to focus your search using key words or phrases for specific resources.

A.3. Contact the foundation

To learn more about the OpenPOWER Foundation, please use the following contact points:

- General information -- `<info@openpowerfoundation.org>`
- Membership -- `<membership@openpowerfoundation.org>`
- Technical Work Groups and projects -- `<tsc-chair@openpowerfoundation.org>`
- Events and other activities -- `<admin@openpowerfoundation.org>`
- Press/Analysts -- `<press@openpowerfoundation.org>`

More contact information can be found at openpowerfoundation.org/get-involved/contact-us.