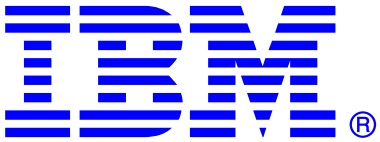


POWER9™ Processor Programming Model Bulletin

September 9, 2019

The specifications in this bulletin are subject to change without notice. Periodic changes to this publication may be incorporated in new additions or supplements to this publication. This publication is provided "AS IS" and IBM Corporation makes no warranty of any kind, expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

IBM® and POWER® are trademarks of IBM Corp., registered in many jurisdictions worldwide.

This material contains some concepts that were developed during research sponsored by the Department of Homeland Security (DHS) Science and Technology Directorate, Cyber Security Division (DHS S&T/CSD) via BAA 11-02; the Department of National Defense of Canada, Defense Research and Development Canada (DRDC); and Air Force Research Laboratory Information Directorate via contract number FA8750-12-C-0243. The U.S. Government and the Department of National Defense of Canada, Defense Research and Development Canada (DRDC) are authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Department of Homeland Security; Air Force Research Laboratory; the U.S. Government; or the Department of National Defense of Canada, Defense Research and Development Canada (DRDC)

POWER9 Processor Programming Model Bulletin

Summary

Some members of the POWER9 Processor family implement the Protected Execution Facility. Together with the Protected Execution Ultravisor and corequisite customizations in the hypervisor (e.g. KVM), the Facility enables the creation of partitions whose memory cannot be accessed by other partitions or by the hypervisor. Hypervisor customizations are for such things as launching a protected partition and providing memory for the ultravisor and protected partitions. The integrity of the protected partitions is not dependent on the hypervisor customizations. The operation of the Facility complies with the architecture for the Secure Memory Facility, a new component of the Power ISA which is described in the following pages. The description is in the form of an RFC (Request for Change) to the Power ISA. The RFC will be included in v3.0C of the Power ISA.

RFC02487: Secure Memory Facility

Date: August 29, 2019
Target Version: 3.0C
Source Version: 2.07B/3.0
Books and sections affected:

Book 1:

- Section 2.7 System Call Instructions
- Section 3.3.20 Move To/From System Register Instructions

Book 2:

- Section 1.6.4 Guarded
- Section 5.3.1 Causes of Transaction Failure
- Section 5.3.2 Recording of Transaction Failure
- Section 5.4.2 Transaction EXception And Status Register
- Section 5.4.3 Transaction Failure Instruction Address Register (TFIAR)

Book 3S:

- Section 1.2.1 Definitions and Notation
- Section 1.4 Exceptions
- Section 2.2 Logical Partitioning Control Register (LPCR)
- Section 2.5 Processor Compatibility Register (PCR)
- Section 2.7 Sharing Hypervisor and Ultravisor Resources
- Section 2.10 Hypervisor Interrupt Little-Endian (HILE) Bit
- Chapter 2+ Ultravisor and Secure Memory Facility (SMF)
- Section 3.2.1 Machine State Register
- Section 3.2.2 State Transitions Associated with the Transactional Memory Facility
- Section 3.2.3 Processor Stop Status and Control Register (PSSCR)
- Section 3.3.1 System Linkage Instructions
- Section 3.3.2 Power-Saving Mode
- Section 3.3.2.1 Power-Saving Mode Instruction
- Section 3.3.2.2 Entering and Exiting Power-Saving Mode
- Section 4.3.10 Software-use SPRs
- Section 4.4.5 Move To/From System Register Instructions
- Section 5.3.2 Address Wrapping Combined with Changing MSR Bit SF
- Section 5.7.3 Ultravisor Real, Hypervisor Real, and Virtual Real Addressing Modes
- Section 5.7.3.1 Ultravisor/Hypervisor Offset Real Mode Address
- Section 5.7.3.2 Storage Control Attributes for Accesses in Ultravisor and Hypervisor Real Addressing Modes
- Section 5.7.3.2.1 Hypervisor Real Mode Storage Control
- Section 5.7.4 Definitions
- Section 5.7.5 Address Ranges Having Defined Uses
- Section 5.7.6.1 Partition Table
- Section 5.7.14 Storage Protection
- Section 5.7.14.5+ Secure Memory Protection
- Section 5.9.2 Synchronize Instruction
- Section 5.10 Page Table Update Synchronization Requirements
- 6.2.2+ Ultravisor Machine Status Save/Restore Registers
- Section 6.2.12 Hypervisor Facility Status and Control Register
- Section 6.3 Interrupt Synchronization
- Section 6.4 Interrupt Classes
- Section 6.4.1 Precise Interrupt
- Section 6.4.3 Interrupt processing
- Section 6.4.4 Implicit alteration of HSRR0 and HSRR1
- Section 6.5 Interrupt Definitions
- Section 6.5.1 System Reset Interrupt
- Section 6.5.3 Data Storage Interrupt (DSI)
- Section 6.5.5 Instruction Storage Interrupt (ISI)
- Section 6.5.9 Program Interrupt
- Section 6.5.14 System Call Interrupt
- Section 6.5.15 Trace Interrupt
- Section 6.5.16 Hypervisor Data Storage Interrupt (HDSI)
- Section 6.5.17 Hypervisor Instruction Storage Interrupt (HISI)
- Secction 6.5.18 Hypervisor Emulation Assistance Interrupt
- Section 6.5.27+ Directed Ultravisor Doorbell Interrupt
- Section 6.7.2 Ordered Exceptions
- Section 6.9 Interrupt Priorities
- Section 8.3 Completed Instruction Address Breakpoint
- Section 8.4 Data Address Watchpoint
- Section 10.1 Overview
- Section 10.2 Programming Model
- Section 10.3.1 Directed Privileged Doorbell Exception State
- Section 10.4 Processor Control Instructions

Chapter 11. Synchronization Requirements for Context Alterations

Book Appendices:

Appendix G. Opcode Maps

Appendices H, I, J. Power ISA AS Instruction Set

Summary:

Adds support for the secure memory facility (SMF) security feature including creation of a new ultravisor privilege mode, checking of a secure property for each page of system memory, ultravisor interception of interrupts from secure partitions for protection of processing state, and ultravisor messages.

Motivation

System software, including both operating system and hypervisor, comprises millions of lines of code developed by large and often disparate teams of programmers. While these components are responsible for isolating executables as well as entire virtual machines from one another to protect sensitive information, they are themselves exposed to security vulnerabilities. In order to provide protection that is independent of the security of these large system components, the secure memory facility (SMF) is added to the processor design. The SMF is implemented in hardware plus a software component that runs at a privilege level above hypervisor privilege.

This RFC provides support for coarse-grained security - preventing the hypervisor from observing the data of secure partitions.

Changes to the Books

Please consider that this RFC was started before v3.0 was published. As a result, there are varying vintages of architecture excerpts present. Please focus on the actual changes, and not the surrounding material.

Some of the changes described herein have nothing to do with SMF.

Editorial Note: There are a large number of random places where urfid, URMOR, and either non-ultravisor or ultravisor-privileged will need to be added to existing text for completeness. Most such cases are deliberately not shown in the RFC to avoid inflating the size to be even more unwieldy.

Book 1:

Section 2.7 System Call Instructions

This section mentions that LEV > 1 for **sc** argument is reserved. We are using LEV = 2 for ultravisor calls. In the description of the **sc** instruction, change the third paragraph.

---------- Begin text ----------

The use of the LEV field is described in Book III. In the first form of the instruction the LEV values greater than 2 are reserved, and bits 0:4 of the LEV field (instruction bits 20:24) are treated as a reserved field.

---------- End text ----------

Section 3.3.20 Move To/From System Register Instructions

For mtspr, add pointer to book 3 for details of loading TEXASR.

---------- Begin text ----------

```
n ← spr5:9 || spr0:4
switch (n)
  case(13): see Book III
  case(130): see Book III
  case(808, 809, 810, 811):
  default:
    if length(SPR(n)) = 64 then
      SPR(n) ← (RS)
    else
      SPR(n) ← (RS)32:63
```

The SPR field denotes a Special Purpose Register, encoded as shown in the table below. If the SPR field contains a value from 808 through 811, the instruction specifies a reserved SPR, and is treated as a no-op; see Section 1.3.3, “Reserved Fields, Reserved Values, and Reserved SPRs”. Otherwise, unless the SPR field contains 13 or 130 (denoting the AMR or the TEXASR), the contents of register RS are placed into the designated Special Purpose Register. For Special Purpose Registers that are 32 bits long, the low-order 32 bits of RS are placed into the SPR.

The AMR (Authority Mask Register) is used for “storage protection.” This use, and operation of ***mtspr*** for the AMR, are described in Book III.

The TEXASR (Transaction Exception and Status Register) is used in the analysis of transaction failures, as described in <crossref to bk2 ch5>. The operation of ***mtspr*** for the TEXASR is described in Book III.

---------- End text ----------

For mfspr, add pointer to book 3 for details of reading TEXASR.

---------- Begin text ----------

```
n ← spr5:9 || spr0:4
switch (n)
  case(129): see Book III
  case(130): see Book III
  case(808, 809, 810, 811):
  default:
    if length(SPR(n)) = 64 then
      RT ← SPR(n)
    else
      RT ← 320 || SPR(n)
```

The SPR field denotes a Special Purpose Register, encoded as shown in the table below. If the SPR field contains 129, the instruction references the Transaction Failure Instruction Address Register (TFIAR) and the result is dependent on the privilege with which it is executed. See Book III. If the SPR field contains a value from 808 through 811, the instruction specifies a reserved SPR, and is treated as a no-op; see Section 1.3.3, “Reserved Fields, Reserved Values, and Reserved SPRs”. Otherwise, unless the SPR field contains 130 (denoting the TEXASR), the contents of the designated Special Purpose Register are placed into register RT. For Special Purpose Registers that are 32 bits long, the low-order 32 bits of RT receive the contents of the Special Purpose Register and the high-order 32 bits of RT are set to zero.

The TEXASR (Transaction Exception and Status Register) is used in the analysis of transaction failures, as described in <crossref to bk2 ch5>. The operation of ***mfspr*** for the TEXASR is described in Book III.

---------- End text ----------

Book 2:

Section 1.6.4 Guarded

Fix the second paragraph to deal with radix and SMF.

---------- Begin text ----------

Except in ultravisor or hypervisor real addressing mode, instructions are not fetched from storage that is Guarded. Except in these addressing modes, if the instruction addressed by the current instruction address is in such storage, the system instruction storage error handler may be invoked (see Section 6.5.5 of Book III).

---------- End text ----------

Section 5.3.1 Causes of Transaction Failure

Add that urfid, msgsndu and msgclru are disallowed.

---------- Begin text ----------

- Execution of the following instructions while in the Transactional state: ***icbi, copy, paste[.], cp_abort, lwat, ldat, stwat, stdat, dcbf, dcbi, dcbst, rfscv, rfid, hrfid, urfid, rfebb, mtmsr[d], msgsnd, msgsndp, msgsndu, msgclr, msgclrp, msgclru, slbie[g], slbia, slbmte, slbfee, stop,*** and ***tlbie[l].*** (These instructions are considered to be *disallowed* in Transactional state.) The disallowed instruction is not executed; failure handling occurs before it has been executed.

---------- End text ----------

Section 5.3.2 Recording of Transaction Failure

Add the secure bit.

---------- Begin text ----------

When transaction failure occurs, information about the cause and circumstances of failure are recorded in SPRs associated with the transactional facility. Failure recording is performed a single time per transaction that fails, controlled by the state of the TEXASR failure summary (FS) bit; when 0, FS indicates that failure recording has not already been performed, and is therefore permissible.

The following RTL function specifies the actions taken during the recording of transaction failure:

```
TMRecordFailure(FailureCause)
                  #FailureCause is 32-bit cause
code
if TEXASR_FS = 0
   if failure IA known then
      TFIAR ← CIA
      TEXASR_37 ← 1
   else
      TFIAR ← approximate instruction address
      TEXASR_37 ← 0
   TEXASR_0:31 ← FailureCause
   if MSR_TS=0b01 then TEXASR_Suspended ← 1
   TEXASR_Privilege ← MSR_HV || MSR_PR
   TFIAR_Privilege ← MSR_HV || MSR_PR
   if MSR_PR=0 then
     TEXASR_Secure ← MSR_S
   TEXASR_FS ← 1
   TDOOMED ← 1
```

When failure recording occurs, the TEXASR and TFIAR SPRs are set indicating the source of failure. When possible, TFIAR is set to the effective address of the instruction that caused the failure, and $TEXASR_{37}$ is set to 1 indicating that the contents of TFIAR are exact. When the instruction address is not known exactly, an approximate value is placed in TFIAR and $TEXASR_{37}$ is set to 0. TEXASR bits 0:31 are set indicating the cause of the failure, and the $TEXASR_{Suspended}$, $TEXASR_{Privilege}$, and $TFIAR_{Privilege}$ fields are set indicating the machine state in which the failure was recorded. If MSR_{PR}=0, $TEXASR_{Secure}$ is also set indicating the machine state in which the failure was recorded. $TEXASR_{TL}$ is unchanged. The TDOOMED bit is set to 1.

Programming Note

TFIAR is intended for use in the debugging of transactional programs by identifying the source of transaction failure. Because TFIAR may not always be set exactly, software should test $TEXASR_{37}$ before use; if zero, the contents of TFIAR are an approximation.

---------- End text ----------

Section 5.4.2 Transaction EXception And Status Register

Add bit 40 to hold a copy of MSR[S].

---------- Begin text ----------

39 Reserved

40 ***Secure (S)***
The thread was in Secure state when the failure was recorded.

Programming Note

This bit is read and written only when MSR_{PR}=0. When MSR_{PR}=1, ***mtspr*** instructions and transaction failure do not modify the bit, and ***mfspr*** instructions return 0 for the bit.

41:51 Reserved

52:63 ***Transaction Level (TL)***
Transaction level (nesting depth + 1) for the active transaction, if any; otherwise 0 if the most recently executed transaction completed successfully, or the transaction level at which

the most recently executed transaction failed if the most recently executed transaction did not complete successfully.

Programming Note

A value of 1 corresponds to an outer transaction. A value greater than 1 corresponds to a nested transaction.

The transaction level in $TEXASR_{TL}$ contains an unsigned integer indicating whether the current transaction is an outer transaction, or is nested, and if nested, its depth. The maximum transaction level supported by a given implementation is of the form 2^t - 1. The value of *t* corresponding to the smallest maximum is 4; the value of *t* corresponding to the largest maximum is 12. This value is tied to the "Maximum transaction level" parameter useful for application programmers, as specified in Section 4.1. The high-order 12-*t* bits of $TEXASR_{TL}$ are treated as reserved.

Transaction failure information is contained in $TEXASR_{0:3740}$. The fields of TEXASR are initialized upon the successful initiation of a transaction from the Non-transactional state, by setting $TEXASR_{TL}$ to 1, indicating an outer transaction, and all other fields to 0.

When transaction failure is recorded, the failure summary bit $TEXASR_{FS}$ is set to 1, indicating that failure has been detected for the active transaction and that failure recording has been performed. $TEXASR_{0:31}$ are set indicating the source of the failure. Exactly one of bits 8 through 31 will be set indicating the instruction or event that caused failure. In the event of failure due to the execution of a ***tabort.***, ***tabortdc.***, ***tabortdci.***, ***tabortwc.***, ***tabortwci.*** or ***treclaim.*** instruction, $TEXASR_{31}$ is set to 1, and, for ***tabort.*** and ***treclaim.***, a software defined failure code is copied from a register operand to $TEXASR_{0:7}$. $TEXASR_{Suspended}$ indicates whether the transaction was in the Suspended state at the time that failure was recorded. The values of MSR_{HV} and MSR_{PR} at the time that failure is recorded are copied to $TEXASR_{34}$ and $TEXASR_{35}$, respectively. If MSR_{PR} is 0 at the time that failure is recorded, the value of MSR_S is copied into $TEXASR_{40}$. In some circumstances, the failure causing instruction address in TFIAR may not be exact. In such circumstances, $TEXASR_{37}$ is set to 0 indicating that the contents of TFIAR are not exact; otherwise $TEXASR_{37}$ is set to 1.

Programming Note

The transaction level contained in $TEXASR_{TL}$ should be interpreted by software as follows:

When in the Transactional or Suspended state, this field contains an unsigned integer representing the transaction level of the active transaction, with 1 indicating an outer transaction, and a number greater than 1 indicating a nested transaction. The nesting depth of the active transaction is $TEXASR_{TL} - 1$.

When in the Non-transactional state, $TEXASR_{TL}$ contains 0 if the last transaction committed successfully, otherwise it contains the transaction level at which the most recent transaction failed.

Programming Note

The Privilege and Secure bits in TEXASR represent the state of the machine at the point when failure is recorded. This information may be used by problem state software to determine whether an unexpected interaction with the operating system or with higher-privilege software (hypervisor or ultravisor) was responsible for transaction failure. (In problem state, ***mftexasr*** returns 0 for the Secure bit.) This information may be useful to operating systems, hypervisors, or ultravisors when restoring register state for failure handling after the transactional facility was reclaimed, to determine which level of software has retained the pre-transactional version of the checkpointed registers.

Note that any transfer of control to the hypervisor during a transaction initiated by a secure partition will cause the transaction to fail because the ultravisor must protect the checkpointed register values from the hypervisor, and therefore must execute ***treclaim.*** before passing control to the hypervisor. Thus if TEXASR reports that the failure was caused by ***treclaim.*** and occurred in ultravisor state, nothing is likely to be gained from additional analysis.

---------- End text ----------

Section 5.4.3 Transaction Failure Instruction Address Register (TFIAR)

Fix for the addition of MSR[S] and explain why it needn't appear in the TFIAR.

---------- Begin text ----------

The Transaction Failure Instruction Address Register is a 64-bit SPR that is set to the exact effective address of the instruction causing the failure, when possible. Bits

62:63 contain the value that was in MSR_{HV} || MSR_{PR} when the failure was recorded.

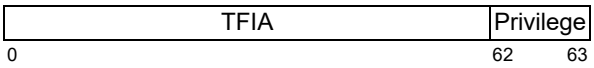


Figure 1. Transaction Failure Instruction Address Register (TFIAR)

In certain cases, the exact address may not be available, and therefore TFIAR will be an approximation. An approximate value will point to an instruction near the instruction that was executing at the time of the failure. TFIAR accuracy is recorded in an Exact bit residing in $TEXASR_{37}$.

Programming Note

The purpose of the Privilege field in TFIAR is to prevent ***mftfiar*** executed in a given privilege state from returning an effective address that was recorded in a higher privilege state; see <xref to section 4.4.4 (mfspr)>. There is no need for ***mftfiar*** to prevent the hypervisor from returning an effective address that was recorded in ultravisor state because the ultravisor, running in Non-transactional state, can use $TEXASR_{FS}$ and $TEXASR_{S\ HV\ PR}$ to determine whether the most recent transaction failure occurred in ultravisor state and, if the most recent transaction failure did occur in ultravisor state, the ultravisor can set TFIAR to all 0s before passing control to the hypervisor. For this reason there is no need for TFIAR to contain the value that was in MSR_S when the failure was recorded.

---------- End text ----------

Book 3S:

Section 1.2.1 Definitions and Notation

Add definitions of ultravisor and hypervisor interrupts. Delete trap interrupt.

---------- Begin text ----------

- **exception**
  An error, unusual condition, or external signal, that may set a status bit and may or may not cause an interrupt, depending upon whether the corresponding interrupt is enabled.
- **interrupt**
  The act of changing the machine state in response to an exception, as described in Chapter 6. “Interrupts” on page 1095.
  - ultravisor interrupt
    An interrupt that forces the thread into ultravisor state by explicitly setting $MSR_{S\ HV\ PR}$ to 0b110 (see <xref to section 3.2.1>).
  - hypervisor interrupt
    An interrupt that forces the thread into hypervisor state by explicitly setting $MSR_{HV\ PR}$ to 0b10 and is not an ultravisor interrupt.

  All interrupts explicitly set MSR_{PR} to 0.

---------- End text ----------

Update the definition of “must” as follows..

---------- Begin text ----------

- **“must”**
  If software that runs in hypervisor state violates a rule that is stated using the word “must” (e.g., “this field must be set to 0”), and the rule pertains to the contents of a hypervisor resource, to executing an instruction that can be executed only in hypervisor state, or to accessing storage in real addressing mode, the results are undefined, and may include altering resources belonging to other partitions, causing the system to “hang”, etc. The same is true for software that runs in ultravisor state and violates a “must” rule pertaining to an ultravisor resource or instruction.

---------- End text ----------

After the hardware bullet, add the following new bullet.

---------- Begin text ----------

- **ultravisor privileged**

  A term used to describe an instruction or facility that is available when and only when the thread is in ultravisor state.

---------- End text ----------

Update the definition of hypervisor privileged as follows to smooth over the addition of ultravisor.

---------- Begin text ----------

- **hypervisor privileged**

  A term used to describe an instruction or facility that is available when and only when the thread is in hypervisor state.

  Programming Note

  Because ultravisor state is also a hypervisor state, hypervisor privileged instructions and facilities are also available when the thread is in ultravisor state. (The distinct privilege states in which a hypervisor privileged instruction or facility is available are: hypervisor non-ultravisor state, and ultravisor state.)

---------- End text ----------

Add the definition of privileged as follows for completeness.

---------- Begin text ----------

- **privileged**

  A term used to describe an instruction or facility that is available when and only when the thread is in privileged state.

  Programming Note

  Because hypervisor state is also a privileged state, privileged instructions and facilities are also available when the thread is in hypervisor state (and when the thread is in ultravisor state). (The distinct privilege states in which a privileged instruction or facility is available are: privileged non-hypervisor state, hypervisor non-ultravisor state, and ultravisor state.)

---------- End text ----------

Section 1.4 Exceptions

After the second bullet, add the following as the third bullet.

---------- Begin text ----------

- an attempt to modify an ultravisor resource when the thread is in privileged but non-ultravisor state (see <crossref to new ch3>), or an attempt to execute an ultravisor-only instruction (e.g., ***urfid***, ***msgsndu***, ***msgclru***) when the thread is in privileged but non-ultravisor state

---------- End text ----------

Section 2.2 Logical Partitioning Control Register (LPCR)

In the AIL definition, add the missing special case of scv. Add the ultravisor interrupts to the list that work as if AIL=0 and also interrupts taken by the ultravisor. Restructure the overrides to improve clarity.

---------- Begin text ----------

The overrides mentioned above are as follows. The list should be read from the top down; the first item matching a given situation applies.

- If the interrupt results in the thread being in ultravisor state, the interrupt is taken as if $LPCR_{AIL}$=0.
- Machine Check, System Reset, and Hypervisor Maintenance interrupts are taken as if $LPCR_{AIL}$=0.
- If the interrupt occurs when MSR_{IR}=0 or MSR_{DR}=0, the interrupt is taken as if $LPCR_{AIL}$=0.
- If the interrupt causes a transition from MSR_{HV}=0 to MSR_{HV}=1 and HR=0, the interrupt is taken as if $LPCR_{AIL}$=0.

---------- End text ----------

Extend the EVIRT definition to cover ultravisor-privileged resources and instructions.

---------- Begin text ----------

42 ***Enhanced Virtualization*** (EVIRT)

Controls whether Enhanced Virtualization is enabled, as specified below.

0 Enhanced virtualization is disabled: attempts to execute hypervisor-privileged instructions or access hypervisor resources, or PTCR, DAWR0, DAWRX0, or CIABR when they are ultravisor resources, in privileged but non-hypervisor state cause a Privileged Instruction type Program interrupt; attempts to access undefined SPR numbers other than 0 for ***mtspr*** and 0, 4, 5, and 6 for ***mfspr*** in privileged state are treated as noops.

1 Enhanced virtualization is enabled: attempts to execute hypervisor-privileged instructions or access hypervisor resources, or PTCR, DAWR0, DAWRX0, or CIABR when they are ultravisor resources, in privileged but non-hypervisor state cause a Hypervisor Emulation Assistance interrupt; attempts to access undefined SPR numbers other than 0 for ***mtspr*** and 0, 4, 5, and 6 for ***mfspr*** in privileged state cause a Hypervisor Emulation Assistance interrupt.

---------- End text ----------

Change HR definition to be consistent with PATE[HR].

---------- Begin text ----------

43 ***Host Radix*** (HR)

Indicates whether the hypervisor uses Radix Tree translation for the partition, as specified below.

0 hypervisor uses HPT translation for this partition.

1 hypervisor uses Radix Tree translation for this partition.

Programming Note

The hypervisor must program HR to match the Host Radix bit in the appropriate Partition Table Entry. If the values do not match when $MSR_{HV\ PR} \neq 0b10$ or $MSR_{IR\ DR} \neq 0b00$, the results are undefined.

HR is duplicated in the LPCR because there are times such as immediately after a partition swap when it is difficult for hardware to quickly access the PATE.

---------- End text ----------

Section 2.5 Processor Compatibility Register (PCR)

Add urfid to [h]rfid as unaffected by the PCR with respect to its setting of the MSR.

---------- Begin text ----------

The PCR has no effect on the setting of the MSR and [H]SRR1 by interrupts (and of the Count Register by the System Call Vectored interrupt), and by the ***rfscv***, ***rfid***, ***hrfid***, ***urfid***, and ***mtmsr[d]*** instructions, except as specified elsewhere in this section.

---------- End text ----------

Section 2.7 Sharing Hypervisor and Ultravisor Resources

Extend the section title. Add URMOR and SMFCTRL to the section. Also correct that PECE may differ by thread in LPCR.

---------- Begin text ----------

Certain additional hypervisor and ultravisor resources, and the PVR, may be shared among threads. Programs that modify these resources must be aware of this sharing, and must allow for the fact that changes to these resources may affect more than one thread.

The following additional resources may be shared among threads.

- HRMOR (see Section 2.3)
- LPIDR (see Section 2.4)
- PCR (see Section 2.5)
- URMOR (see <crossref to ch3>)
- PVR (see Section 4.3.1)
- RPR (see Section 4.3.8)
- PTCR (see Section 5.7.6.1)
- AMOR (see Section 5.7.14.1)
- HMEER (see Section 6.2.10)
- Time Base (see Section 7.2)
- Virtual Time Base (see Section 7.3)
- Hypervisor Decrementer (see Section 7.5)
- certain implementation-specific registers or implementation-specific fields in architected registers

The set of resources that are shared is implementation-dependent.

Threads that share any of the resources listed above, with the exception of the PTCR, the PVR, the URMOR, and the HRMOR, must be in the same partition.

For each field of the LPCR, except the AIL, ONL, LD, PECE, HDICE, and MER fields, software must ensure that the contents of the field are identical among all threads that are in the same partition and are in a state such that the contents of the field could have side effects. (E.g., software must ensure that the contents of $LPCR_{LPES}$ are identical among all threads that are in the same partition and are not in hypervisor state.) For the HDICE field, software must ensure that the contents of the field are identical among all threads that share the Hypervisor Decrementer and are in a state such that the contents of the field could have side effects. There are no identity requirements for the other fields listed in the first sentence of this paragraph.

Software must ensure that the contents of UILE and $SMFCTRL_E$ are identical among all threads in the system that have completed ultravisor initialization. The contents of the D and UDEE fields of SMFCTRL may differ among threads.

---------- End text ----------

Section 2.10 Hypervisor Interrupt Little-Endian (HILE) Bit

Restate the circumstances for using HILE to allow for ultravisor state.

---------- Begin text ----------

The Hypervisor Interrupt Little-Endian (HILE) bit is a bit in an implementation-dependent register or similar mechanism. The contents of the HILE bit are copied into MSR_{LE} by interrupts that result in $MSR_{S\ HV}$ being equal to 0b01 (see Section 6.5), to establish the Endian mode for the interrupt handler. The HILE bit is set, by an implementation-dependent method, only during system initialization.

The contents of the HILE bit must be the same for all threads under the control of a given instance of the hypervisor; otherwise all results are undefined.

---------- End text ----------

Chapter 2+ Ultravisor and Secure Memory Facility (SMF)

Add the following chapter after Chapter 2, Logical Partitioning (LPAR) and Thread Control, to describe the ultravisor functionality and related support for the secure memory facility.

---------- Begin text ----------

3.1 Overview

The Secure Memory Facility (SMF) provides secure isolation of partitions from one another and from higher privilege system software. SMF functionality is implemented using a combination of hardware facilities and firmware that runs at a privilege level above the hypervisor. SMF targets a threat model in which the hypervisor can be compromised such that its inherent isolation capabilities can no longer be counted on. Maintaining the security of data is the sole objective of the ultravisor. It has no role in platform management and is not expected to deal with denial of service attacks. References elsewhere in the Books to "secure systems" apply more generally, and do not necessarily imply that the system uses SMF.

The SMF protection mechanism is based on the assignment of partitions to security domains. The hypervisor is in one security domain, along with all processes that run directly under the hypervisor and all partitions that do not take advantage of the SMF security capabilities. Each of the secure partitions is assigned to its own security domain so that its data and instructions can be protected from access by other security domains. A partition is identified as secure when MSR_S=1. Each location in main storage has an associated Secure Memory property, mem_{SM}. Memory with mem_{SM}=1 may be referred to as "secure memory." Memory with mem_{SM}=0 may be referred to as "ordinary memory." The granularity and method with which main storage is mapped for the Secure Memory property is implementation specific. The Secure Memory property is commonly cached in the TLB and in implementation-specific lookaside buffers. When secure data are to be shared with untrusted software, the standard synchronization associated with PTE updates is used to regulate access. For example, prior to sharing secure data, the PTEs used to access the data are marked invalid and the corresponding TLB entires invalidated by the ultravisor using the standard invalidation sequence. (See <crossref to pte update sequence desc>.) The data are then encrypted and made available in ordinary memory (either mem_{SM} is turned off or the data are moved to ordinary memory). Finally the PTEs that will be used to access the data in ordinary memory are marked valid. (The last step may be done lazily.) Software running with MSR_S=0 is prohibited from accessing secure memory. Software running with MSR_S=1 may access both secure and ordinary memory.

Programming Note

The ultravisor will commonly use a no-execute protection setting to prevent a secure partition from executing instructions from any ordinary memory mapped into its address space.

SMF firmware runs in ultravisor state, a privilege level above that of the hypervisor. That firmware, along with the SMF hardware, is responsible for maintaining isolation of secure partitions from each other and from the hypervisor. This is accomplished by direct ultravisor management of the partition-scoped translation tables in secure memory for secure partitions. The ultravisor itself runs only in (ultravisor) real addressing mode. Security is the result of proper management of the partition-scoped translation together with the hardware enforcement of the access restriction for secure memory. With this hybrid approach, firmware has the ability to enable secure memory sharing between secure partitions and ordinary memory sharing between a given secure partition and the hypervisor, e.g. for system calls. The ultravisor can access any architecture resource or facility.

The hypervisor is expected to cooperate in the management of secure partitions by using ultravisor calls to dispatch them and to manage their storage allocations. To protect against programming errors and malicious hypervisor behavior, ***mtmsr***[***d***], ***rfid***, ***hrfid***, and ***rfscv*** preserve MSR_S and hypervisor interrupts from secure partitions are always received in ultravisor state.

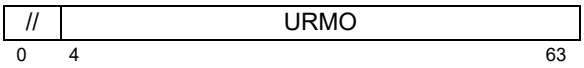
The purpose of intercepting hypervisor interrupts is to protect the state of the secure partition from the hypervisor. The ultravisor's interrupt handler provides a 'shim' that saves and clears the processing state, and then transfers control to the hypervisor to handle the exception condition itself. The ultravisor will restore the secure partition state when it services the ultravisor call to (re-) dispatch the secure partition. Note that the ultravisor's goal is merely to protect the security of data, and not to provide broader system management oversight.

Programming Note

When the ultravisor intercepts an interrupt with a transaction active, it must save and restore the checkpointed registers (causing the transaction to fail).

3.2 Ultravisor Real Mode Offset Register (URMOR)

The layout of the Ultravisor Real Mode Offset Register (URMOR) is shown in Figure 2 below.



Bit(s)	Name	Description
4:63	URMO	Real Mode Offset

Figure 2. Ultravisor Real Mode Offset Register

All other fields are reserved.

The supported URMO values are the non-negative multiples of 2^r, where r is the same implementation-dependent value that constrains the HRMO field of the HRMOR.

The contents of the URMOR affect how some storage accesses are performed as described in <crossref to Real and Virtual Real Addressing modes section> and <crossref to Address Ranges Having Defined Uses section>.

3.3 Ultravisor Interrupt Little-Endian (UILE) Bit

The Ultravisor Interrupt Little-Endian (UILE) bit is a bit in an implementation-dependent register or similar mechanism. The contents of the UILE bit are copied into MSR_{LE} by interrupts that result in $MSR_{S\ HV}$ being equal to 0b11 (see Section 6.5), to establish the Endian mode for the interrupt handler. The UILE bit is set, by an implementation-dependent method, only during system initialization.

The contents of the UILE bit must be the same for all threads in the system; otherwise all results are undefined.

3.4 Secure Memory Facility Control Register (SMFCTRL)

The Secure Memory Facility Control Register (SMFCTRL) is shown in Figure 3 below.

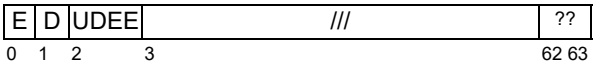


Figure 3. Secure Memory Facility Control Register (SMFCTRL)

Bit Description

0 **SMF Enable (E)**

0 SMF functionality including secure memory checking is disabled.
1 SMF functionality including secure memory checking is enabled.

When $SMFCTRL_E$=1, writing the PTCR is ultravisor privileged.

1 **Debug enable (D)**

0 Ultravisor debug mode is disabled.
1 Ultravisor debug mode is enabled.

In ultravisor debug mode, CIABR, DAWRn, and DAWRXn are ultravisor privileged. See <crossref to ch.8> for a description of how instruction and data address tracing work in ultravisor debug mode.

2 **Ultravisor Doorbell Exit Enable (UDEE)**

0 When the ***stop*** instruction is executed with $PSSCR_{EC}$=1, Directed Ultravisor Doorbell exceptions are not enabled to cause exit from power-saving mode
1 When the ***stop*** instruction is executed with $PSSCR_{EC}$=1, Directed Ultravisor Doorbell exceptions are enabled to cause exit from power-saving mode.

3:61 Reserved

62:63 Implementation-specific use

$SMFCTRL_E$ must be set to 1 prior to exiting ultravisor state if the system will use the SMF facilities. (When $SMFCTRL_E$=0 and MSR_S=0, there is no way to achieve MSR_S=1 without a reboot.)

Programming Note

The two useful runtime states with respect to SMF operation are (1) MSR_S=0 and $SMFCTRL_E$=0 (SMF permanently disabled) and (2) $SMFCTRL_E$=1 (SMF enabled). Very limited verification may be performed on the state with MSR_S=1 and $SMFCTRL_E$=0 and around state changes of $SMFCTRL_E$. Therefore, software should change the value of $SMFCTRL_E$ at most once, making the change prior to the first dispatch of a partition, and spending as little time as possible in the state with MSR_S=1 and $SMFCTRL_E$=0.

If $SMFCTRL_E$=0, $SMFCTRL_D$ and $SMFCTRL_{UDEE}$ must be set to zero. References to $SMFCTRL_D$=1 or $SMFCTRL_{UDEE}$=1 elsewhere in the architecture assume $SMFCTRL_E$=1 unless otherwise stated or obvious from context.

3.4.1 Enabling SMF and Secure Memory Enforcement

The $SMFCTRL_E$ bit enables SMF functionality. When $SMFCTRL_E$=1, certain facilities are ultravisor resources

instead of hypervisor resources and secure memory checking is enabled.

Independent of the basic feature enablement above, SMF has state transition rules that facilitate the protection of security domains. (While these rules are nominally independent of the value of $SMFCTRL_E$, some transitions cannot happen when $SMFCTRL_E$=0. Specifically, ultravisor interrupts cannot occur when $SMFCTRL_E$=0.)

- All interrupts that are not ultravisor interrupts preserve MSR_S. (Ultravisor interrupts necessarily set MSR_S to 1.)
- ***mtmsr*[*d*]**, ***rfid***, ***hrfid***, and ***rfscv*** are not permitted to change MSR_S

Table 1 summarizes the effect of the $SMFCTRL_E$ bit and the $MSR_{S\ HV\ PR}$ bits on various facilities.

facility	$MSR_{S\ HV\ PR}$	$SMFCTRL_E$	$LPCR_{EVIRT}$	behavior
mtspr or ***mfspr*** specifying URMOR, USRR0, USRR1, USPRG0, USPRG1, or SMFCTRL; ***urfid***, ***msgsndu***, ***msgclru***	110	dc	dc	execution allowed
	all xxx except 110**	dc	dc	Privileged Instruction type Program interrupt to xx0
mtspr specifying PTCR	110	dc	dc	execution allowed
	010	0	dc	execution allowed
		1	dc	HEAI to 010
	x00	dc	0	Privileged Instruction type Program interrupt to x00
			1	HEAI to x10
	xx1**	dc	dc	Privileged Instruction type Program interrupt to xx0
mtspr or ***mfspr*** specifying DAWR0, DAWRX0 or CIABR when $SMFCTRL_D$=1	110	1	dc	execution allowed
	010	1	dc	HEAI to 010
	x00	1	0	Privileged Instruction type Program interrupt to x00
			1	HEAI to x10
	xx1**	1	dc	Privileged Instruction type Program interrupt to xx0
sc 2 instruction	dc**	0	dc	hypervisor call, but with SRR1 showing LEV=2
	dc**	1	dc	ultravisor call
mem_{SM} evaluation and match	dc**	0	dc	disabled
	dc**	1	dc	enabled*
* mem_{SM} evaluation may be avoided when MSR_S=1, depending on translation cache design dc = don't care ** The encoding $MSR_{S\ HV\ PR}$=0b111 is reserved and must not be used.				

Table 1: Ultravisor Resource Behavior

Programming Note

Access to memory by mechanisms outside the core must also enforce secure memory access restrictions. Facilities that translate addresses or otherwise use real addresses to access memory must check mem_{SM} against $PATE_S$ for the partition on behalf of which they access memory.

Such mechanisms will require a means to evaluate mem_{SM} and a proxy for $SMFCTRL_E$ to provide the same enablement function for secure memory access enforcement as in the core.

In addition or as an alternative, TCE tables may be managed by the ultravisor and used to identify regions of memory that I/O devices may access.

---------- End text ----------

Section 3.2.1 Machine State Register

Add the S bit and references to **urfid**, as needed. Also specify that SF must be set to 1 in ultravisor state.

---------- Begin text ----------

The Machine State Register (MSR) is a 64-bit register. This register defines the state of the thread. On interrupt, the MSR bits are altered in accordance with Figure 55 on page 1010. The MSR can also be modified by the ***mtmsr***[***d***], ***rfscv***, ***rfid***, ***hrfid*** and ***urfid*** instructions. It can be read by the ***mfmsr*** instruction.

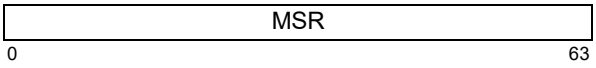


Figure 4. Machine State Register

Below are shown the bit definitions for the Machine State Register.

Bit **Description**

0 ***Sixty-Four-Bit Mode*** (SF)

0 The thread is in 32-bit mode.
1 The thread is in 64-bit mode.

Software must ensure that SF=1 whenever the thread is in ultravisor state.

1:2 Reserved

3 ***Hypervisor State*** (HV)

0 The thread is not in hypervisor state.
1 If MSR_{PR}=0, the thread is in hypervisor state; otherwise the thread is not in hypervisor state.

Programming Note

The privilege state of the thread is determined by MSR_S, MSR_{HV}, and MSR_{PR}, as follows.

S	HV	PR	
0	x	1	problem
1	0	1	problem
x	x	0	privileged
x	1	0	hypervisor
1	1	0	ultravisor
1	1	1	reserved

Hypervisor state is also a privileged state (MSR_{PR} = 0). All references to “privileged state” in the Books include hypervisor state unless otherwise stated or obvious from context. Ultravisor state is also a hypervisor state ($MSR_{HV\ PR}$ = 0b10). All references to “hypervisor state” in the Books include ultravisor state unless otherwise stated or obvious from context.

MSR_{HV} can be set to 1 only by the *System Call* instruction and some interrupts. It can be set to 0 only by ***rfid***, ***hrfid*** and ***urfid.***

It is possible to run an operating system in an environment that lacks a hypervisor, by always having MSR_{HV} = 1 and using $MSR_{HV\ PR}$ = 0b10 for the operating system (effectively, the OS runs in hypervisor state) and $MSR_{HV\ PR}$ = 0b11 for applications. In this use, MSR_S would be 0, and the environment would also lack an ultravisor.

4 Reserved

5 Software must ensure that this bit contains 0; otherwise the results of executing all instructions are boundedly undefined.

Programming Note

This bit is initialized to 0 by hardware at system bringup. The handling of this bit by interrupts and by the ***rfid***, ***hrfid***, ***urfid***, and ***rfscv*** instructions is such that, unless software deliberately sets the bit to 1, the bit will continue to contain 0.

6:28 Reserved

29:30 ***Transaction State*** (TS)

00 Non-transactional
01 Suspended
10 Transactional
11 Reserved

Changes to MSR[TS] that are caused by Transactional Memory instructions, and by invocation of the transaction's failure handler, take effect immediately (even though these instructions and events are not context synchronizing).

31 ***Transactional Memory Available*** (TM)

0 The thread cannot execute any Transactional Memory instructions or access any Transactional Memory registers.

1 The thread can execute Transactional Memory instructions and access Transactional Memory registers unless the Transactional Memory facility has been made unavailable by some other register.

Programming Note

To access Transactional Memory registers and execute Transactional Memory instructions, it must also be true that $HFSCR_{TM}$=1 or the thread is in hypervisor state. See Section 6.2.12 on page 1099 for more information.

32:37 Reserved

38 ***Vector Available*** (VEC)

0 The thread cannot execute any vector instructions, including vector loads, stores, and moves.

1 The thread can execute vector instructions unless they have been made unavailable by some other register.

39 Reserved

40 ***VSX Available*** (VSX)

0 The thread cannot execute any VSX instructions, including VSX loads, stores, and moves.

1 The thread can execute VSX instructions unless they have been made unavailable by some other register.

Programming Note

An application binary interface defined to support Vector-Scalar operations should also specify a requirement that MSR_{FP} and MSR_{VEC} be set to 1 whenever MSR_{VSX} is set to 1.

41 ***Secure*** (S)

0 The thread is not in Secure state. It may not access Secure memory. The thread is not in ultravisor state.

1 The thread is in Secure state. If MSR_{HV}=1 and MSR_{PR}=0, the thread is in ultravisor state; otherwise the value does not affect privilege. The state with MSR_{HV}=1 and MSR_{PR}=1 is reserved. Software must not set $MSR_{S\ HV\ PR}$ = 0b111. References elsewhere in this document to $MSR_{HV\ PR}$=0b11 assume MSR_S=0 unless otherwise stated or obvious from context.

Programming Note

MSR_S can be set to 1 only by the *System Call* instruction and some interrupts. It can be set to 0 only by ***urfid***.

Any instruction or event that causes $MSR_{S\ HV\ PR}$ to be set to 0b110 also causes MSR_{IR} and MSR_{DR} to be set to 0.

42:47 Reserved

...48 ***External Interrupt Enable*** (EE)

0 External, Decrementer, Performance Monitor, and Privileged Doorbell interrupts are disabled.

1 External, Decrementer, Performance Monitor, and Privileged Doorbell interrupts are enabled.

This bit also affects whether Hypervisor Decrementer, Hypervisor Maintenance, Directed Hypervisor Doorbell, and Directed Ultravisor Doorbell interrupts are enabled; see Section 6.5.12 on page 1122, Section 6.5.19 on page 1130, Section 6.5.20 on page 1131, and <crossref to ultravisor doorbell int>.

49 ***Problem State*** (PR)

0 The thread is in privileged state.

1 If $MSR_{S\ HV} \neq$ 0b11, the thread is in problem state.

Programming Note

Any instruction that sets MSR_{PR} to 1 also sets MSR_{EE}, MSR_{IR}, and MSR_{DR} to 1.

The state with $MSR_{S\ HV\ PR}$=0b111 is reserved.

..

51 ***Machine Check Interrupt Enable*** (ME)

0 Machine Check interrupts are disabled.

1 Machine Check interrupts are enabled.

This bit is a hypervisor resource; see Chapter 2., “Logical Partitioning (LPAR) and Thread Control”, on page 879.

Programming Note

The only instructions that can alter MSR_{ME} are ***rfid***, ***hrfid*** and ***urfid***.

52 ***Floating-Point Exception Mode 0*** (FE0)
[Category: Floating-Point]

See below.

53:54 ***Trace Enable*** (TE)

00 Trace Disabled: The thread executes instructions normally.
01 Branch Trace: The thread generates a Branch type Trace interrupt after completing the execution of a branch instruction, whether or not the branch is taken.
10 Single Step Trace: The thread generates a Single-Step type Trace interrupt after successfully completing the execution of the next instruction, unless that instruction is a ***urfid***, ***hrfid, rfid***, ***rfscv***, or a *Power-Saving Mode* instruction, all of which are never traced. Successful completion means that the instruction caused no other interrupt and, if the processor is in the Transactional state, is not one of the instructions that is forbidden in Transactional state (e.g., ***dcbf***; see Section 4.3.1 of Book II).
11 Reserved

Branch tracing need not be supported. If the function is not implemented, the 0b01 bit encoding is treated as reserved.

...

58 ***Instruction Relocate*** (IR)

0 Instruction address translation is disabled.
1 Instruction address translation is enabled.

Programming Note

See the Programming Note in the definition of MSR_S and in the definition of MSR_{PR}.

59 ***Data Relocate*** (DR)

0 Data address translation is disabled. Effective Address Overflow (EAO) (see Book I) does not occur.
1 Data address translation is enabled. EAO causes a Data Storage interrupt.

Programming Note

See the Programming Note in the definition of MSR_S and in the definition of MSR_{PR}.

...

63 ***Little-Endian Mode*** (LE)

0 The thread is in Big-Endian mode.
1 The thread is in Little-Endian mode.

Programming Note

The only instructions that can alter MSR_{LE} are ***rfid***, ***hrfid, urfid***, and rfscv.

...

The initial state of the MSR should be as follows:

...

...

Bit	Name	Value

...

41	S	1

---------- End text ----------

Section 3.2.2 State Transitions Associated with the Transactional Memory Facility

Add urfid to each place in the section where the other *rfid's are listed. The same TM state change restrictions should apply to urfid. (USRR1 will also need to be added to the last one before the e-note.) Special related explanation is added to the last p-note in the section as follows.

---------- Begin text ----------

Programming Note

For ***rfscv, rfid***, ***hrfid***, ***urfid*** and ***mtmsrd***, the attempted transition from S0 to N0 is suppressed in order that interrupt handlers that are "unaware" of transactional memory, and load an MSR value that has not been updated to take account of transactional memory, will continue to work correctly. (If the interrupt occurs when a transaction is running or suspended, the interrupt will set MSR[TS || TM] to S0. If the interrupt handler attempts to load an MSR value that has not been updated to take account of transactional memory, that MSR value will have TS || TM = N0. It is desirable that the interrupt handler remain in state S0, so that it can return normally to the interrupted transaction.)

The problem solved by suppressing this transition does not apply to ***rfebb***, so for ***rfebb*** an attempt to transition from S0 to N0 is not suppressed, and instead causes a TM Bad Thing type Program interrupt.

(The problem solved by suppressing this transition does not apply to ***urfid*** either, since ***urfid*** was added to the architecture after Transactional Memory was added. The transition is suppressed for ***urfid*** because ***urfid*** is very similar to ***[h]rfid***.)

---------- End text ----------

Section 3.2.3 Processor Stop Status and Control Register (PSSCR)

Change “secure environments” to “secure systems” in the second p-note for consistency with other such references.

---------- Begin text ----------

Programming Note

Before dispatching an OS, the hypervisor may initialize this field to 1 in order to prevent the OS from reading the Power-Saving Level Status (PLS) field. This may be necessary in secure systems since an OS may be capable of detecting the presence of another OS on the same processor by observing the state of the PLS field after exiting power-saving mode.

---------- End text ----------

Add behavior for loss of UDEE to the end of the ESL description just before the notes.

---------- Begin text ----------

For power-saving levels that allow loss of SMFCTRL, implementations must provide the means to exit power-saving mode upon the occurrence of a Directed Ultravisor Doorbell exception if $SMFCTRL_{UDEE}$ was set to 1 when ***stop*** was executed. For this case, the implementation is also allowed to exit on the occurrence of a Directed Ultravisor Doorbell exception if $SMFCTRL_{UDEE}$ was set to 0 when ***stop*** was executed.

---------- End text ----------

Add UDEE and ultravisor doorbell to the wakeup description for EC=1.

---------- Begin text ----------

1 If $SMFCTRL_{UDEE}$ was set to 1 when ***stop*** was executed and $SMFCTRL_{UDEE}$ was not lost, hardware will exit power-saving mode when a Directed Ultravisor Doorbell exception occurs. If $LPCR_{PECE}$ is not lost, hardware will exit power-saving mode when a System Reset exception or one of the events specified in $LPCR_{PECE}$ occurs. If the event is a Machine Check exception, then a Machine Check interrupt occurs; otherwise a System Reset interrupt occurs, and the contents of SRR1 indicate the event that caused exit from power-saving mode.

---------- End text ----------

Section 3.3.1 System Linkage Instructions

Adjust the LEV field description in the description of the sc instruction as follows:

---------- Begin text ----------

Then a System Call interrupt is generated. The interrupt causes the MSR to be set as described in Section 6.5, “Interrupt Definitions” on page 1009. The setting of the MSR is affected by the contents of the LEV field. LEV values greater than 2 are reserved. Bits 0:4 of the LEV field (instruction bits 20:24) are treated as a reserved field.

The interrupt causes the next instruction to be fetched from effective address 0x0000_0000_0000_0C00.

This instruction is context synchronizing.

Special Registers Altered:

SRR0 SRR1 MSR

Programming Note

If LEV=1, the hypervisor is invoked.

If LEV=2 and $SMFCTRL_E$ = 1, the ultravisor is invoked.

If LEV=2 and $SMFCTRL_E$ = 0, the hypervisor is invoked. However, such invocation should be considered a programming error.

Executing this instruction with LEV=1 or LEV=2 is the only way that executing an instruction can cause a transition from non-hypervisor state to hypervisor state on the thread that executed the instruction. Executing this instruction with LEV=2 when $SMFCTRL_E$=1 is the only way that executing an instruction can cause a transition from non-ultravisor state to ultravisor state on the thread that executed the instruction.

In correct use, this instruction is used to “call up” one privilege level (application program calls operating system, operating system calls hypervisor, hypervisor calls ultravisor). However, it is possible for a program to call up more than one level (e.g., for an application program to call the hypervisor). An attempt to call up more than one level should be considered a programming error.

---------- End text ----------

In the description of rfscv, don’t allow S (41) to be changed. Also prevent translation from being enabled in ultravisor state. Also eliminate MSR[LE] (bit 63). Add USRR0 to the [H]SRR0s for an enabled pending exception. Make corresponding changes (and fixes) to the p-note following the verbal description.

---------- Begin text ----------

```
if (MSR29:31 ¬= 0b010 | CTR29:31 ¬= 0b000) then
   MSR29:31 ← CTR29:31
MSR48 ← CTR48 | CTR49
MSR58 ← (CTR58 | CTR49)
     & ¬(MSR41 & MSR3 & (¬CTR49))
MSR59 ← (CTR59 | CTR49)
```

& ¬(MSR$_{41}$ & MSR$_3$ & (¬CTR$_{49}$))
MSR$_{0:2\ 4:28\ 32\ 37:40\ 49:50\ 52:57\ 60:63}$ ← CTR$_{0:2\ 4:28\ 32\ 37:40\ 49:50\ 52:57\ 60:63}$
NIA ←$_{iea}$ LR$_{0:61}$ || 0b00

If bits 29 through 31 of the MSR are not equal to 0b010 or bits 29 through 31 of the Count Register are not equal to 0b000, then the value of bits 29 through 31 of the Count Register is placed into bits 29 through 31 of the MSR. The result of ORing bits 48 and 49 of the Count Register is placed into MSR$_{48}$. The result of ANDing bit 41 of the MSR with bit 3 of the MSR and with the complement of bit 49 of the Count Register is complemented and then ANDed with the result of ORing bits 58 and 49 of the Count Register and placed into MSR$_{58}$. The result of ANDing bit 41 of the MSR with bit 3 of the MSR and with the complement of bit 49 of the Count Register is complemented and then ANDed with the result of ORing bits 59 and 49 of the Count Register and placed into MSR$_{59}$. Bits 0:2, 4:28, 32, 37:40, 49:50, 52:57, and 60:63 of the Count Register are placed into the corresponding bits of the MSR.

If the instruction attempts to cause an illegal transaction state transition (see Table 3, “Transaction state transitions that can be requested by rfebb, rfid, rfscv, hrfid, and mtmsrd.,” on page 984), or when TM is disabled by the PCR, a transition to Problem state with an active transaction, a TM Bad Thing type Program interrupt is generated (unless a higher-priority exception is pending). If this interrupt is generated, the value placed into SRR0 by the interrupt processing mechanism (see Section 6.4.3) is the address of the ***rfid*** instruction. Otherwise, if the new MSR value does not enable any pending exceptions, then the next instruction is fetched, under control of the new MSR value, from the address LR$_{0:61}$ || 0b00 (when SF=1 in the new MSR value) or ^{32}0 || LR$_{32:61}$ || 0b00 (when SF=0 in the new MSR value). If the new MSR value enables one or more pending exceptions, the interrupt associated with the highest priority pending exception is generated; in this case the value placed into SRR0, HSRR0, or USRR0 by the interrupt processing mechanism (see Section 6.4.3) is the address of the instruction that would have been executed next had the interrupt not occurred.

This instruction is privileged and context synchronizing.

Special Registers Altered:

MSR

Programming Note

If this instruction sets MSR$_{PR}$ to 1, it also sets MSR$_{EE}$, MSR$_{IR}$, and MSR$_{DR}$ to 1. If this instruction results in MSR$_{S\ HV\ PR}$ being equal to 0b110, it also sets MSR$_{IR}$ and MSR$_{DR}$ to 0.

This instruction does not alter MSR$_{HV}$, MSR$_S$, or MSR$_{ME}$.

---------- End text ----------

In the description of rfid, don’t allow S (41) to be changed. Also don’t allow translation to be enabled in ultravisor state.

---------- Begin text ----------

MSR$_{51}$ ← (MSR$_3$ & SRR1$_{51}$) | ((¬MSR$_3$) & MSR$_{51}$)
MSR$_3$ ← MSR$_3$ & SRR1$_3$
if (MSR$_{29:31}$ ¬= 0b010 | SRR1$_{29:31}$ ¬= 0b000) then
  MSR$_{29:31}$ ← SRR1$_{29:31}$
MSR$_{48}$ ← SRR1$_{48}$ | SRR1$_{49}$
MSR$_{58}$ ← (SRR1$_{58}$ | SRR1$_{49}$)
  & ¬(MSR$_{41}$ & MSR$_3$ & (¬SRR1$_{49}$))
MSR$_{59}$ ← (SRR1$_{59}$ | SRR1$_{49}$)
  & ¬(MSR$_{41}$ & MSR$_3$ & (¬SRR1$_{49}$))
MSR$_{0:2\ 4:28\ 32\ 37:40\ 49:50\ 52:57\ 60:63}$ ← SRR1$_{0:2\ 4:28\ 32\ 37:40\ 49:50\ 52:57\ 60:63}$
NIA ←$_{iea}$ SRR0$_{0:61}$ || 0b00

If MSR$_3$=1 then bits 3 and 51 of SRR1 are placed into the corresponding bits of the MSR. If bits 29 through 31 of the MSR are not equal to 0b010 or bits 29 through 31 of SRR1 are not equal to 0b000, then the value of bits 29 through 31 of SRR1 is placed into bits 29 through 31 of the MSR. The result of ORing bits 48 and 49 of SRR1 is placed into MSR$_{48}$. The result of ANDing bit 41 of the MSR with bit 3 of the MSR and with the complement of bit 49 of SRR1 is complemented and then ANDed with the result of ORing bits 58 and 49 of SRR1 and placed into MSR$_{58}$. The result of ANDing bit 41 of the MSR with bit 3 of the MSR and with the complement of bit 49 of SRR1 is complemented and then ANDed with the result of ORing bits 59 and 49 of SRR1 and placed into MSR$_{59}$. Bits 0:2, 4:28, 32, 37:40, 49:50, 52:57, and 60:63 of SRR1 are placed into the corresponding bits of the MSR.

---------- End text ----------

---------- Begin text ----------

... If the new MSR value enables one or more pending exceptions, the interrupt associated with the highest priority pending exception is generated; in this case the value placed into SRR0, HSRR0, or USRR0 by the interrupt processing mechanism (see Section 6.4.3) is the address of the instruction that would have been executed next had the interrupt not occurred.

This instruction is privileged and context synchronizing.

Special Registers Altered:

MSR

Programming Note

If this instruction sets MSR$_{PR}$ to 1, it also sets MSR$_{EE}$, MSR$_{IR}$, and MSR$_{DR}$ to 1. If this instruction results in MSR$_{S\ HV\ PR}$ being equal to 0b110, it also sets MSR$_{IR}$ and MSR$_{DR}$ to 0.

---------- End text ----------

In the description of hrfid, don't allow S (41) to be changed. Also don't allow translation to be enabled in ultravisor state.

---------- Begin text ----------

```
if (MSR29:31 ¬= 0b010 | HSRR129:31 ¬= 0b000) then
   MSR29:31 ← HSRR129:31
MSR48 ← HSRR148 | HSRR149
MSR58 ← (HSRR158 | HSRR149)
     & ¬(MSR41 & HSRR13 & (¬HSRR149))
MSR59 ← (HSRR159 | HSRR149)
     & ¬(MSR41 & HSRR13 & (¬HSRR149))
MSR0:28 32 37:40 49:57 60:63 ← HSRR10:28 32 37:40 49:57 60:63
NIA ←iea HSRR00:61 || 0b00
```

If bits 29 through 31 of the MSR are not equal to 0b010 or bits 29 through 31 of HSRR1 are not equal to 0b000, then the value of bits 29 through 31 of HSRR1 is placed into bits 29 through 31 of the MSR. The result of ORing bits 48 and 49 of HSRR1 is placed into MSR_{48}. The result of ANDing bit 41 of the MSR with bit 3 of HSRR1 and with the complement of bit 49 of HSRR1 is complemented and then ANDed with the result of ORing bits 58 and 49 of HSRR1 and placed into MSR_{58}. The result of ANDing bit 41 of the MSR with bit 3 of HSRR1 and with the complement of bit 49 of HSRR1 is complemented and then ANDed with the result of ORing bits 59 and 49 of HSRR1 and placed into MSR_{59}. Bits 0:28, 32, 37:40, 49:57, and 60:63 of HSRR1 are placed into the corresponding bits of the MSR.

If the instruction attempts to cause an illegal transaction state transition (see Table 3, "Transaction state transitions that can be requested by rfebb, rfid, rfscv, hrfid, and mtmsrd.," on page 984), or when TM is disabled by the PCR, a transition to Problem state with an active transaction, a TM Bad Thing type Program interrupt is generated (unless a higher-priority exception is pending). If this interrupt is generated, the value placed into SRR0 by the interrupt processing mechanism (see Section 6.4.3) is the address of the ***hrfid*** instruction. Otherwise, if the new MSR value does not enable any pending exceptions, then the next instruction is fetched, under control of the new MSR value, from the address $HSRR0_{0:61}$ || 0b00 (when SF=1 in the new MSR value) or $^{32}0$ || $HSRR0_{32:61}$ || 0b00 (when SF=0 in the new MSR value). If the new MSR value enables one or more pending exceptions, the interrupt associated with the highest priority pending exception is generated; in this case the value placed into SRR0, HSRR0, or USRR0 by the interrupt processing mechanism (see Section 6.4.3) is the address of the instruction that would have been executed next had the interrupt not occurred.

---------- End text ----------

---------- Begin text ----------

Programming Note

If this instruction sets MSR_{PR} to 1, it also sets MSR_{EE}, MSR_{IR}, and MSR_{DR} to 1. If this instruction results in $MSR_{S\ HV\ PR}$ being equal to 0b110, it also sets MSR_{IR} and MSR_{DR} to 0.

---------- End text ----------

After the description of hrfid add urfid:

---------- Begin text ----------

Ultravisor Return From Interrupt Doubleword *XL-form*

urfid

19	///	///	///	306	/
0	6	11	16	21	31

```
if (MSR29:31 ¬= 0b010 | USRR129:31 ¬= 0b000) then
   MSR29:31 ← USRR129:31
MSR48 ← USRR148 | USRR149
MSR58 ← (USRR158 | USRR149)
     & ¬(USRR141 & USRR13 & (¬USRR149))
MSR59 ← (USRR159 | USRR149)
     & ¬(USRR141 & USRR13 & (¬USRR149))
MSR0:28 32 37:41 49:57 60:63 ← USRR10:28 32 37:41 49:57 60:63
NIA ←iea USRR00:61 || 0b00
```

If bits 29 through 31 of the MSR are not equal to 0b010 or bits 29 through 31 of USRR1 are not equal to 0b000, then the value of bits 29 through 31 of USRR1 is placed into bits 29 through 31 of the MSR. The result of ORing bits 48 and 49 of USRR1 is placed into MSR_{48}. The result of ANDing bit 41 of USRR1 with bit 3 of USRR1 and with the complement of bit 49 of USRR1 is complemented and then ANDed with the result of ORing bits 58 and 49 of USRR1 and placed into MSR_{58}. The result of ANDing bit 41 of USRR1 with bit 3 of USRR1 and with the complement of bit 49 of USRR1 is complemented and then ANDed with the result of ORing bits 59 and 49 of USRR1 and placed into MSR_{59}. Bits 0:28, 32, 37:41, 49:57, and 60:63 of USRR1 are placed into the corresponding bits of the MSR.

If the instruction attempts to cause an illegal transaction state transition or, when TM is made unavailable in problem state by the PCR, attempts to cause a transition to problem state and also a transaction state transition that Table 3 on page 987 shows as legal and as resulting in the thread being in Transactional or Suspended state, a TM Bad Thing type Program interrupt is generated (unless a higher-priority exception is pending). If this interrupt is generated, the value placed into SRR0 by the interrupt processing mechanism (see Section 6.4.3) is the address of the ***urfid*** instruction. Otherwise, if the new MSR value does not enable any

pending exceptions, then the next instruction is fetched, under control of the new MSR value, from the address $USRR0_{0:61}$ || 0b00 (when SF=1 in the new MSR value) or $^{32}0$ || $USRR0_{32:61}$ || 0b00 (when SF=0 in the new MSR value). If the new MSR value enables one or more pending exceptions, the interrupt associated with the highest priority pending exception is generated; in this case the value placed into SRR0, HSRR0, or USRR0 by the interrupt processing mechanism (see Section 6.4.3) is the address of the instruction that would have been executed next had the interrupt not occurred.

This instruction is ultravisor privileged and context synchronizing.

Special Registers Altered:
MSR

Programming Note

If this instruction sets MSR_{PR} to 1, it also sets MSR_{EE}, MSR_{IR}, and MSR_{DR} to 1. If this instruction sets $MSR_{S\ HV\ PR}$ to 0b110, it also sets MSR_{IR} and MSR_{DR} to 0.

---------- End text ----------

Section 3.3.2 Power-Saving Mode

Add the ultravisor equivalent in the last bullet.

---------- Begin text ----------

Power-Saving Mode is a mode in which the thread does not execute instructions and may consume less power than it would if it were not in power-saving mode. The thread can be put in power-saving mode by executing the ***stop*** instruction.

There are 16 levels of power savings, designated as levels 0-15. For each power-saving level, the power consumed may be less than or equal to the power consumed in the next-lower level, and the time required for the thread to exit power-saving mode and resume execution may be greater than or equal that of the next-lower level.

When the thread is in power-saving mode, some resource state may be lost. The state that may be lost while in each power-saving level is implementation dependent, with the following restrictions.

- For $PSSCR_{ESL}$ = 0 and power-saving level 0000, no thread state is lost.
- There must be a power-saving level in which the Decrementer and all hypervisor resources are maintained as if the thread was not in power-saving mode, and in which sufficient information is maintained to allow the hypervisor to resume execution.
- The amount of state loss in a given level is less than or equal to the amount of state loss in the next higher level.
- The state of all read-only resources, $SMFCTRL_E$, and the URMOR in an SMF-enabled system or the HRMOR in an SMF-disabled system is always maintained.

Programming Note

For the power-saving level corresponding to the second item above, if the state of the Decrementer were not maintained and updated as if the thread was not in power-saving mode, Decrementer exceptions would not reliably cause exit from this power-saving level even if Decrementer exceptions were enabled to cause exit.

---------- End text ----------

Section 3.3.2.1 Power-Saving Mode Instruction

Add SMFCTRL[UDEE] to the list of controls for power-saving exit and note that the ultravisor must not execute stop. (The latter choice was made because of the difficulty for the design to wake up in the right state in the prevented circumstances.) Move the two p-notes to section 3.3.2.2.

---------- Begin text ----------

The thread remains in power-saving mode until either a System Reset exception or certain other events occur. The events that may cause exit from power-saving mode are specified by $PSSCR_{EC}$, $LPCR_{PECE}$, and $SMFCTRL_{UDEE}$. If the event that causes the exit is a System Reset, Machine Check, or Hypervisor Maintenance exception, resource state that would be lost if the exception occurred when the thread was not in power-saving mode may be lost.

An attempt to execute this instruction in Suspended state will result in a TM Bad Thing type Program interrupt.

This instruction should not be executed in ultravisor state because that scenario may not be thoroughly verified.

---------- End text ----------

Section 3.3.2.2 Entering and Exiting Power-Saving Mode

Add a high level description of how SMF/ultravisor relates to power saving. Include UDEE in the possible exit causes. Note that this also tries to fix integration errors from RFC 2492B by moving p-notes from the stop description to this section. Note that “stop” should be in the appropriate font (fixed here w/o change bars).

---------- Begin text ----------

Before software executes the ***stop*** instruction, the PSSCR is initialized. If the ***stop*** instruction is to be used by the OS, the hypervisor initializes the fields that are accessible only to the hypervisor before dispatching the OS. These fields include the SD, ESL, EC, and PSLL fields. See the Programming Notes for these fields in Section 3.2.3 for additional information.

If the ***stop*** instruction is to be executed by the hypervisor when $PSSCR_{EC}$=1, $LPCR_{PECE}$ and $SMFCTRL_{UDEE}$ must be set to the desired value (see Section 2.2 and <crossref to SMFCTRL section>). Depending on the implementation and the power-saving level to be entered, it may also be necessary to save the state of certain resources and perform synchronization procedures to ensure that all stores have been performed with respect to other threads or mechanisms that use the storage areas before executing the ***stop***. See the the User's Manual for the implementation for details.

Software must also specify the requested and maximum power-saving level limit fields (i.e RL and MTL fields), and the Transition Rate (TR) field in the PSSCR in order to bound the range of power-saving modes that can be entered. If the value of the RL field is greater than or equal to the value of the MTL field, the power-saving level will not increase from the initial level during power-saving mode.

Programming Note

If MSR_{EE}=1 when the stop instruction is executed, then the interrupt corresponding to the exception that was expected to cause exit from power-saving mode may occur immediately prior to execution of the ***stop*** instruction. If this occurs, the result may be a software hang condition since the exception that was expected to cause exit from power-saving mode has already occurred.

The above software hang condition can be prevented by setting MSR_{EE}=0 prior to executing ***stop***.

After the thread has entered power-saving mode with $PSSCR_{EC}$=0, any exception may cause exit from power-saving mode. When an exception occurs, power-saving mode is exited either at the instruction following the stop (if MSR_{EE}=0) or in the corresponding interrupt handler (if MSR_{EE}=1).

Programming Note

If ***stop*** was executed when $PSSCR_{EC}$=0, then $PSSCR_{ESL}$ must also be set to 0 and $PSSCR_{RL\ MTL}$ must be set to values that do not allow state loss. (See the EC bit description in <xref to Section 3.2.2>. This guarantees that the state of MSR_{EE} is not lost.)

Programming Note

If ***stop*** was executed when $PSSCR_{EC}$=0 and MSR_{EE}=0 (in order to avoid the hang condition described in a preceding Programming Note), MSR_{EE} should be set to 1 after power-saving mode is exited in order to take the interrupt corresponding to the exception that caused exit from power-saving mode.

After the thread has entered power-saving mode with $PSSCR_{EC}$=1, only the System Reset or Machine Check exceptions and the exceptions enabled in $LPCR_{PECE}$ and $SMFCTRL_{UDEE}$ will cause exit. If the event that causes exit is a Machine Check exception, then a Machine Check interrupt occurs; otherwise a System Reset interrupt occurs, and the contents of SRR1 indicate the exception that caused exit from power-saving mode. If state loss has occurred in an SMF-enabled system, the interrupt is taken in ultravisor state.

If the hypervisor has set $PSSCR_{SD}$=0 prior to when the ***stop*** instruction is executed, the instruction following the ***stop*** may typically be a ***mfspr*** in order to read the contents of $PSSCR_{PLS}$ to determine the maximum power-saving level that was entered during power-saving mode.

Programming Note

The ultravisor does not initiate power-saving.

If a secure partition attempts to execute ***stop*** with parameters that allow state loss, the ultravisor gets control via the Hypervisor Facility Unavailable interrupt. It saves secure state and gives control to the hypervisor's Hypervisor Facility Unavailable interrupt handler.

Upon exit from a state-losing power-saving mode in an SMF-enabled system, the ultravisor gets control at its Machine Check or System Reset interrupt handler. It restores any ultravisor state that was lost, and then services the Directed Ultravisor Doorbell exception if that caused the wakeup. It then restores the HRMOR and transfers control to the hypervisor at the hypervisor's Machine Check interrupt handler if the ultravisor got control at the ultravisor's Machine Check interrupt handler, and to the hypervisor's System Reset interrupt handler otherwise. The hypervisor restores any lost hypervisor state, and then handles the exception (other than Directed Ultravisor Doorbell exception) that caused the wakeup. For this process to work, the ultravisor must have stored a record of its state in some known location prior to transferring control to the hypervisor to execute ***stop***. The hypervisor in turn must have stored its HRMOR value in a location known to the ultravisor. It must also have stored a record of its state in some known location.

The only other function the ultravisor may need to perform for a given power-saving mode transition is to be a proxy accessing hypervisor state in the platform that is mixed with ultravisor state and lacking independent access control.

---------- End text ----------

Section 4.3.10 Software-use SPRs

Clarify the wording of the p-notes.

---------- Begin text ----------

Programming Note

Neither the contents of the SPRGs, nor accessing them using ***mtspr*** or ***mfspr***, has a side effect on the operation of the thread. One or more of the registers is likely to be needed by interrupt handlers that run in privileged non-hypervisor state (e.g., as scratch registers and/or pointers to per thread save areas).

Operating systems must ensure that no sensitive data are left in SPRG3 when a problem state program is dispatched, and operating systems for secure systems must ensure that SPRG3 cannot be used to implement a "covert channel" between problem state programs. These requirements can be satisfied by clearing SPRG3 before passing control to a program that will run in problem state.

...

Programming Note

Neither the contents of the HSPRGs, nor accessing them using ***mtspr*** or ***mfspr***, has a side effect on the operation of the thread. One or more of the registers is likely to be needed by interrupt handlers that run in hypervisor non-ultravisor state (e.g., as scratch registers and/or pointers to per thread save areas).

---------- End text ----------

Add the following description of the new USPRG registers at the end of this subsection.

---------- Begin text ----------

USPRG0 and USPRG1 are 64-bit registers provided for use by ultravisor programs.

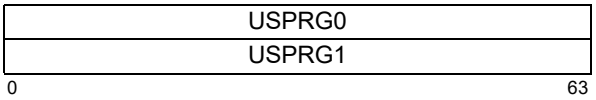


Figure 5. SPRs for use by ultravisor programs

Programming Note

Neither the contents of the USPRGs, nor accessing them using ***mtspr*** or ***mfspr***, has a side effect on the operation of the thread. One or both of the registers is likely to be needed by interrupt handlers that run in ultravisor state (e.g., as scratch registers and/or pointers to per thread save areas).

---------- End text ----------

Section 4.4.5 Move To/From System Register Instructions

Add ultravisor SPRs and note SPRs that can become ultravisor privileged.

---------- Begin text ----------

decimal	SPR[1]	Register Name	Privileged		Length (bits)	Extended Mnemonics*	
	$spr_{5:9}$ $spr_{0:4}$		mtspr	mfspr		mtspr	mfspr
176	00101 10000	DPDES	hypv[2]	yes	64	mtdpdes Rx	mfdpdes Rx
180	00101 10100	DAWR0	hyp/ult[15]	hyp/ult[15]	64	mtdawr0 Rx	mfdawr0 Rx
186	00101 11010	RPR	hypv[2]	hypv[2]	64	mtrpr Rx	mfrpr Rx
187	00101 11011	CIABR	hyp/ult[15]	hyp/ult[15]	64	mtciabr Rx	mfciabr Rx
188	00101 11100	DAWRX0	hyp/ult[15]	hyp/ult[15]	32	mtdawrx0 Rx	mfdawrx0 Rx
190	00101 11110	HFSCR	hypv[2]	hypv[2]	64	mthfscr Rx	mfhfscr Rx
...							
446	01101 11110	TIR	-	yes	64	-	mftir Rx
464	01110 10000	PTCR	hyp/ult[14]	hypv[2]	64	mtptcr Rx	mfptcr Rx
496	01111 10000	USPRG0	ultv	ultv	64	mtusprg0 Rx	mfusprg0 Rx
497	01111 10001	USPRG1	ultv	ultv	64	mtusprg1 Rx	mfusprg1 Rx
505	01111 11001	URMOR	ultv	ultv	64	mturmor Rx	mfurmor Rx
506	01111 11010	USRR0	ultv	ultv	64	mtusrr0 Rx	mfusrr0 Rx
507	01111 11011	USRR1	ultv	ultv	64	mtusrr1 Rx	mfusrr1 Rx
511	01111 11111	SMFCTRL	ultv	ultv	64	mtsmfctrl Rx	mfsmfctrl Rx
768	11000 00000	SIER	-	no[6]	64	-	mfusier Rx mfsier Rx

- This register is not defined for this instruction.

1 Note that the order of the two 5-bit halves of the SPR number is reversed.

2 This register is a hypervisor resource, and can be accessed by this instruction only in hypervisor state (see Chapter 2).

3 This register cannot be directly written. Instead, bits in the register corresponding to 0 bits in (RS) can be cleared using *mtspr SPR,RS*.

4 The value specified in register RS may be masked by the contents of the [U]AMOR before being placed into the AMR; see the ***mtspr*** instruction description.

5 The value specified in register RS may be ANDed with the contents of the AMOR before being placed into the UAMOR; see the ***mtspr*** instruction description.

6 $MMCR0_{PMCC}$ controls the availability of this SPR, and its contents depend on the privilege state in which it is accessed. See Section 9.4.4 for details.

7 The value specified in Register RS may be masked by the contents of the AMOR before being placed into the IAMR; see the ***mtspr*** instruction description.

8 Accesses to these SPRs are noops; see Section 1.3.3, “Reserved Fields, Reserved Values, and Reserved SPRs” in Book I.

9 The length of the GSR is undefined. An access to this SPR affects synchronization of subsequent ***mtspr*** instructions. See the introductory text in this section for more details

10 SPR numbers 777-778, 783, 793-794, and 799 are reserved for the Performance Monitor. All other SPR numbers that are not shown above and are not implementation-specific are reserved.

11 The ***mftb*** instruction is Phased-Out. Assemblers targeting Version 2.03 or later of the architecture should generate an ***mfspr*** instruction for the ***mftb*** and ***mftbu*** extended mnemonics; see the corresponding Assembler Note in the ***mftb*** instruction description (see Section 6.1 of Book II).

12 ***mfspr*** specifying the GSR has no meaningful use. It is treated as a noop. As a result, no extended mnemonic is assigned for it.

13 No extended mnemonic is provided because previous versions of the architecture defined the obvious extended mnemonic as resolving to the non-privileged SPR number, and because there is no software benefit in using the privileged SPR number, rather than the non-privileged SPR number, for this function.

14 ***mtspr*** specifying this register is ultravisor privileged when $SMFCTRL_E$=1; otherwise it is hypervisor privileged.

15 This register is ultravisor privileged when $SMFCTRL_D$=1; otherwise it is hypervisor privileged.

---------- End text ----------

For mtspr, add the secure bit write suppression to TEXASR for problem state and repackage the paragraph about privilege violations for mtspr to cover hypervisor access to ultravisor privileged SPRs, as follows. Also replace “n” in “SPR(n)” with the appropriate number where it stands for a single number, including cases not shown below.

---------- Begin text ----------

```
case(48):  SPR(48) ← (RS)
           if PATE_HR=1 for the partition then
             All implementation-specific
             lookaside information that was
             created when address translation
             was enabled and for which effPID≠0
             is invalidated.
case (130): if MSR_PR = 1 then
              SPR(130)_0:39 41:63 ← (RS)_0:39 41:63
            else
              SPR(130) ← (RS)
case (157): if MSR_HV PR = 0b10 then
              SPR(157) ← (RS)
            else
              SPR(157) ← (RS) & AMOR
```

...

The SPR field denotes a Special Purpose Register, encoded as shown in Figure 18. If the SPR field contains the value 158, the instruction indicates the start of a sequence of ***mtspr*** instructions that may be synchronized as a group. See the introductory material in this section for more information. If the SPR field contains a value from 808 through 811, the instruction specifies a reserved SPR, and is treated as a no-op; see Section 1.3.3, “Reserved Fields, Reserved Values, and Reserved SPRs” in Book I. Otherwise, the contents of register RS are placed into the designated Special Purpose Register, except as described in the next six paragraphs. For Special Purpose Registers that are 32 bits long, the low-order 32 bits of RS are placed into the SPR.

...

When the designated SPR is the UAMOR and $MSR_{HV\ PR}$=0b00, the contents of register RS are ANDed with the contents of the AMOR and the result is placed into the UAMOR.

When the designated SPR is the TEXASR and MSR_{PR}=1, bit 40 of the TEXASR is not modified.

When the designated SPR is the PIDR and the partition uses Radix Tree translation, the implementation specific lookaside invalidation specified by ***slbia*** with IH=0b011 is performed along with the SPR update. When the designated SPR is the LPIDR and both the originating and destination partitions use Radix Tree translation, the implementation specific lookaside invalidation specified by ***slbia*** with IH=0b110 is performed along with the SPR update.

...

spr_0=1 if and only if writing the register is privileged. Execution of this instruction specifying an SPR number with spr_0=1 when the privilege state of the thread does not permit the access causes one of the following.

- MSR_{PR}=1: Privileged Instruction type Program interrupt
- $MSR_{HV\ PR}$=0b00 or $MSR_{S\ HV\ PR}$=0b010 and the SPR is always an ultravisor resource (independent of the contents of SMFCTRL): Privileged Instruction type Program interrupt
- $MSR_{HV\ PR}$=0b00 and the SPR is a hypervisor resource (see Figure 17) or is PTCR, DAWR0, DAWRX0, or CIABR when they are ultravisor privileged for the operation:
  - $LPCR_{EVIRT}$=0: Privileged Instruction type Program interrupt
  - $LPCR_{EVIRT}$=1: Hypervisor Emulation Assistance interrupt
- $MSR_{S\ HV\ PR}$=0b010 and the SPR is PTCR, DAWR0, DAWRX0, or CIABR when they are ultravisor privileged for the operation: Hypervisor Emulation Assistance interrupt

---------- End text ----------

For mfspr, add that problem state reads zero from the secure bit of TEXASR and clarify that MSR[S] has no effect on the zeroing of the value read from TFIAR. Repackage the paragraph about privilege violations for mfspr to cover hypervisor access to ultravisor privileged SPRs, as follows:

---------- Begin text ----------

```
case(129):
  if (MSR_HV PR = 0b10) | (TFIAR_PR=MSR_PR=1) |
   ((MSR_HV PR = 0b00) & (TFIAR_HV PR ≠ 0b10)) then
     RT ← SPR(n)
  else
     RT ← 0
case(130):
  RT ← SPR(n)
  if MSR_PR = 1 then
     RT_40 ← 0
case(808, 809, 810, 811):
```

...

The SPR field denotes a Special Purpose Register, encoded as shown in Figure 17. If the designated Special Purpose Register is the TFIAR and TFIAR indicates the failure was recorded in a state more privileged than the current state, register RT is set to zero; ultravisor and hypervisor states are not differentiated (MSR_S is ignored) for this purpose. If the designated Special Purpose Register is the TEXASR and MSR_{PR}=1, the contents of the TEXASR are placed into register RT, but with bit 40 of RT set to 0. If the SPR field contains 158, the instruction specifies the GSR, and is treated as a noop. If the SPR field contains a value from 808 through 811, the instruction specifies a reserved SPR, and is treated as a no-op; see Section 1.3.3, “Reserved Fields, Reserved Values, and

Reserved SPRs" in Book I. Otherwise, the contents of the designated Special Purpose Register are placed into register RT. For Special Purpose Registers that are 32 bits long, the low-order 32 bits of RT receive the contents of the Special Purpose Register and the high-order 32 bits of RT are set to zero.

Programming Note

Note that when a problem state transaction's failure is recorded in hypervisor state and there is a subsequent need for a context switch in privileged, non-hypervisor state, an attempt to save TFIAR will result in zeros being saved. This is harmless because if the original application ever tries to read the TFIAR, it would read zeros anyway, since the failure took place in hypervisor state.

spr_0=1 if and only if reading the register is privileged. Execution of this instruction specifying an SPR number with spr_0=1 when the privilege state of the thread does not permit the access causes one of the following.

- MSR_{PR}=1: Privileged Instruction type Program interrupt
- $MSR_{HV\ PR}$=0b00 or $MSR_{S\ HV\ PR}$=0b010 and the SPR is always an ultravisor resource (independent of the contents of SMFCTRL): Privileged Instruction type Program interrupt
- $MSR_{HV\ PR}$=0b00 and the SPR is a hypervisor resource (see Figure 17) or is DAWR0, DAWRX0, or CIABR when they are ultravisor privileged for the operation:
  - $LPCR_{EVIRT}$=0: Privileged Instruction type Program interrupt
  - $LPCR_{EVIRT}$=1: Hypervisor Emulation Assistance interrupt
- $MSR_{S\ HV\ PR}$=0b010 and the SPR is DAWR0, DAWRX0, or CIABR when they are ultravisor privileged for the operation: Hypervisor Emulation Assistance interrupt

---------- End text ----------

Prevent mtmsr from changing S(41) and prevent translation from being enabled in ultravisor state. The corresponding changes are required for mtmsrd, but are not shown here.

---------- Begin text ----------

```
if L = 0 then
   MSR48 ← (RS)48 | (RS)49
   MSR58 ← ((RS)58 | (RS)49)
     & ¬(MSR41 & MSR3 & (¬(RS)49))
   MSR59 ← ((RS)59 | (RS)49)
     & ¬(MSR41 & MSR3 & (¬(RS)49))
   MSR32:40 42:47 49:50 52:57 60:62
          ←(RS)32:40 42:47 49:50 52:57 60:62
else
   MSR48 62 ← (RS)48 62
```

The MSR is set based on the contents of register RS and of the L field.

L=0:

The result of ORing bits 48 and 49 of register RS is placed into MSR_{48}. The result of ANDing bit 41 of the MSR with bit 3 of the MSR and with the complement of bit 49 of register RS is complemented and then ANDed with the result of ORing bits 58 and 49 of register RS and placed into MSR_{58}. The result of ANDing bit 41 of the MSR with bit 3 of the MSR and with the complement of bit 49 of register RS is complemented and then ANDed with the result of ORing bits 59 and 49 of register RS and placed into MSR_{59}. Bits 32:40, 42:47, 49:50, 52:57, and 60:62 of register RS are placed into the corresponding bits of the MSR.

---------- End text ----------

Add a reference to the MSR_S bit in the first programming note for the mtmsr instruction, as follows:

---------- Begin text ----------

Programming Note

If this instruction sets MSR_{PR} to 1, it also sets MSR_{EE}, MSR_{IR}, and MSR_{DR} to 1. If this instruction results in $MSR_{S\ HV\ PR}$ being equal to 0b110, it also sets MSR_{IR} and MSR_{DR} to 0.

This instruction does not alter MSR_S, MSR_{ME} or MSR_{LE}. (This instruction does not alter MSR_{HV} because it does not alter any of the high-order 32 bits of the MSR.)

If the only MSR bits to be altered are $MSR_{EE\ RI}$, to obtain the best performance L=1 should be used.

---------- End text ----------

Add a reference to the MSR_S bit in the first programming note for the mtmsrd instruction, as follows:

---------- Begin text ----------

Programming Note

If this instruction sets MSR_{PR} to 1, it also sets MSR_{EE}, MSR_{IR}, and MSR_{DR} to 1. If this instruction results in $MSR_{S\ HV\ PR}$ being equal to 0b110, it also sets MSR_{IR} and MSR_{DR} to 0.

This instruction does not alter MSR_{HV}, MSR_S, MSR_{ME}, or MSR_{LE}.

If the only MSR bits to be altered are $MSR_{EE\ RI}$, to obtain the best performance L=1 should be used.

---------- End text ----------

Section 5.3.2 Address Wrapping Combined with Changing MSR Bit SF

Add USRR0 to the p-note.

---------- Begin text ----------

Programming Note

If the thread is in 32-bit mode, the current instruction is at effective address 2^{32} - 4, and an interrupt occurs that is defined to set SRR0, HSRR0, or USRR0 (or LR, for the System Call Vectored interrupt) to the effective address of the next sequential instruction, the contents of SRR0, HSRR0, or USRR0 (or LR), as appropriate to the interrupt, are undefined.

---------- End text ----------

Section 5.7.3 Ultravisor Real, Hypervisor Real, and Virtual Real Addressing Modes

Extend real mode discussion to cover UV mode.

---------- Begin text ----------

If a storage access is an instruction fetch performed when instruction address translation is disabled, or if the access is a data access performed when data address translation is disabled, it is said to be performed in “ultravisor real addressing mode” if the thread is in ultravisor state, in “hypervisor real addressing mode” if the thread is in hypervisor non-ultravisor state, and in “virtual real addressing mode” if the thread is in privileged non-hypervisor state. Storage accesses in ultravisor real, hypervisor real, and virtual real addressing modes are performed in a manner that depends on the contents of $MSR_{S\ HV}$, VPM, $PATE_{PS}$, URMOR (see <crossref to UV chapter>), HRMOR (see Chapter 2), bit 0 of the effective address (EA0), and the state of the Real Mode Storage Control Facility as described below. Bits 1:3 of the effective address are ignored.

$MSR_{S\ HV}$=0b11

- If EA_0=0, the Ultravisor Offset Real Mode Address mechanism, described in <crossref to UV Offset real mode addressing section>, controls the access.
- If EA_0=1, bits 4:63 of the effective address are used as the real address for the access.

$MSR_{S\ HV}$=0b01

- If EA_0=0, the Hypervisor Offset Real Mode Address mechanism, described in Section 5.7.3.1, controls the access.
- If EA_0=1, bits 4:63 of the effective address are used as the real address for the access.

MSR_{HV}=0

- If $PATE_{HR||GR}$=0b00, the Virtual Real Mode Addressing mechanism, described in Section 5.7.3.3, controls the access.
- If $PATE_{HR||GR}\neq$0b00, partition-scoped translation is performed on the effective address. (See Section 5.7.12.3, “Obtaining Host Real Address, Radix on Radix”.)

---------- End text ----------

Section 5.7.3.1 Ultravisor/Hypervisor Offset Real Mode Address

Add URMOR description, etc.

---------- Begin text ----------

If MSR_{HV} = 1 and EA_0 = 0, the access is controlled by the contents of the Ultravisor Real Mode Offset Register or the Hypervisor Real Mode Offset Register, depending on the value of MSR_S, as follows.

Ultravisor Real Mode Offset Register (URMOR)

When MSR_S=1, bits 4:63 of the effective address for the access are ORed with the 60-bit offset represented by the contents of the URMOR, and the 60-bit result is used as the real address for the access.

Hypervisor Real Mode Offset Register (HRMOR)

When MSR_S=0, bits 4:63 of the effective address for the access are ORed with the 60-bit offset represented by the contents of the HRMOR, and the 60-bit result is used as the real address for the access.

For each of the two registers, the supported offset values are all values of the form $i\times2^r$, where $0 \le i < 2^j$, and j and r are implementation-dependent values having the properties that $12 \le r \le 26$ (i.e., the minimum offset granularity is 4 KB and the maximum offset granularity is 64 MB) and j+r = m, where the real address size supported by the implementation is m bits.

Programming Note

$EA_{4:63-r}$ should equal $^{60-r}0$. If this condition is satisfied, ORing the effective address with the offset produces a result that is equivalent to adding the effective address and the offset.

If m<60, $EA_{4:63-m}$, $URMOR_{4:63-m}$, and $HRMOR_{4:63-m}$ must be zeros.

---------- End text ----------

Section 5.7.3.2 Storage Control Attributes for Accesses in Ultravisor and Hypervisor Real Addressing Modes

Add UV.

---------- Begin text ----------

Storage accesses in ultravisor and hypervisor real addressing modes are performed as though all of storage had the following storage control attributes, except as modified by the Hypervisor Real Mode Storage Control facility (see Section 5.7.3.2.1). (The storage control attributes are defined in Book II.)

- not Write Through Required
- not Caching Inhibited, for instruction fetches
- not Caching Inhibited, for data accesses except those caused by the *Load/Store Caching Inhibited* instructions; Caching Inhibited, for data accesses caused by the *Load/Store Caching Inhibited* instructions
- Memory Coherence Required, for data accesses
- Guarded
- not SAO

Additionally, storage accesses in ultravisor and hypervisor real addressing modes are performed as though all storage was not No-execute.

Programming Note

Because storage accesses in ultravisor and hypervisor real addressing modes do not use the SLB or the Page Table, accesses in this mode bypass all checking and recording of information contained therein (e.g., storage protection checks that use information contained therein are not performed, and reference and change information is not recorded).

---------- End text ----------

Section 5.7.3.2.1 Hypervisor Real Mode Storage Control

Add a couple more allowances for ultravisor. Delete the e-note that was really intended for the old high water mark method.

---------- Begin text ----------

The Hypervisor Real Mode Storage Control facility provides a means of specifying portions of real storage that are treated as non-Guarded in ultravisor and hypervisor real addressing modes ($MSR_{HV\ PR}$=0b10, and MSR_{IR}=0 or MSR_{DR}=0, as appropriate for the type of access). The remaining portions are treated as Guarded in ultravisor and hypervisor real addressing modes. The means is a hypervisor resource (see Chapter 2), and may also be system-specific.

The facility divides real storage into history blocks, in implementation-specific sizes. The history for instruction fetches is tracked separately from that for data accesses. If there is no instruction fetch history for a block and it is the target of an instruction fetch, the access is performed as though the block is Guarded, but the block is treated as non-Guarded for subsequent instruction fetches on a best effort basis, limited by the amount of history that the facility can maintain. If there is no data access history for a block and it is accessed using a *Load/Store Caching Inhibited* instruction, the access is performed as though the block is Guarded, and the block is treated as Guarded for subsequent accesses on a best effort basis, limited by the amount of history that the facility can maintain. If there is no data access history for a block and it is accessed using any other *Load* or *Store* instruction, the access is performed as though the block is Guarded, but the block is treated as non-Guarded for subsequent accesses on a best effort basis, limited by the amount of history that the facility can maintain.

The storage location specified by a *Load/Store Caching Inhibited* instruction must not be in storage that is specified by the Hypervisor Real Mode Storage Control facility to be treated as non-Guarded. The storage location specified by any other *Load* or *Store* instruction must not be in storage that is specified by the Hypervisor Real Mode Storage Control facility to be treated as Guarded. ("specified by the Hypervisor Real Mode Storage Control facility" means "specified in a history block".) The history can be erased using an slbia instruction; see Section 5.9.3.2.

Programming Note

There are two cautions about mixing different types of accesses (i.e. *Load/Store Caching Inhibited* instructions vs. any other *Load* or *Store* instruction vs. instruction fetches). The first, as indicated above, is to avoid confusing the history mechanism, and the granularity for concern is a history block. For this caution, instruction fetches are irrelevant because they have their own history mechanism and are always intended to be non-guarded.

The second caution is to avoid storage paradoxes that result from a Caching Inhibited access to a location that is held in a cache. The nature of this caution and its solution are described in Section 5.8.2.2, “Altering the Storage Control Bits”. The minimum granularity for concern is the history block, but may be larger, depending on extant translations to the storage in question. Since the consistency of instruction storage is managed by software and ultravisor and hypervisor real mode instruction fetches are always not Caching Inhibited, instruction fetches are also irrelevant to this caution.

The facility does not apply to implicit accesses to the Page Table performed during address translation or in recording reference and change information. These accesses are performed as described in Section 5.7.3.4.

Programming Note

The preceding capability can be used to improve the performance of software that runs in ultravisor and hypervisor real addressing modes, by causing accesses to instructions and data that occupy well-behaved storage to be treated as non-Guarded.

---------- End text ----------

Section 5.7.4 Definitions

Add translation mode as a new first defintion. Add S to the MSR states for the adjunct.

---------- Begin text ----------

translation mode: Refers to either HPT translation or Radix Tree translation. The translation mode is specified by the HR field in the Partition Table Entry corresponding to the contents of the LPIDR.

...

adjunct: An adjunct is a software entity that resides in a partition along with an operating system and its applications in order to efficiently provide services (e.g. device drivers) for the partition. The adjunct is managed by the hypervisor. It runs in problem state with $MSR_{S\ HV\ PR}$=0b011, thereby restricting the resources it can modify (MSR_{PR}=1) and causing its interrupts to go to the hypervisor ($MSR_{S\ HV}$=0b01). It shares an HPT with the partition it serves. The adjunct's storage is kept separate from the client partition's storage using Virtual Page Class Key protection. (The adjunct's lightness of weight derives from not requiring a full partition context switch (SLB flush, TLB flush, LPID/PID change, etc.) when the client partition invokes the services of the adjunct.) Each hardware thread may have its own unique translations for an adjunct. As a result, adjunct segment descriptors cannot exist in the process's Segment Table and must instead be bolted in the SLB manually. The adjunct construct exists only with an HR=0 hypervisor and only for LPID≠0. The adjunct has its own 64-bit EA space. Entry to an adjunct is only possible from hypervisor state. Prior to dispatching the adjunct, the hypervisor must invalidate SLB entries that map the effective address range that will be used by the adjunct. Similarly, on exit from the adjunct, the hypervisor must invalidate its SLB entries

---------- End text ----------

Section 5.7.5 Address Ranges Having Defined Uses

Add URMOR to the offset real mode description. Correct the omission that HV change is conditional on HPT translation and add ending in ultravisor state to the AIL=0 cases.

---------- Begin text ----------

- Offset Real Mode interrupt vectors

  The real pages beginning at the real addresses specified by the URMOR and the HRMOR are used similarly to the page for the fixed interrupt vectors.

- Relocated interrupt vectors

  Depending on the values of $LPCR_{AIL}$ and $MSR_{IR\ DR}$ and on the kind of interrupt, and on whether the interrupt will cause MSR_{HV} to change from 0 to 1 when HR=0 or will result in $MSR_{S\ HV}$ being equal to 0b11, either the virtual page containing the byte addressed by effective address 0x0000_0000_0001_8000 or the virtual page containing the byte addressed by effective address 0xC000_0000_0000_4000 may be used similarly to the page for the fixed interrupt vectors. (See Section 2.2.)

---------- End text ----------

Section 5.7.6.1 Partition Table

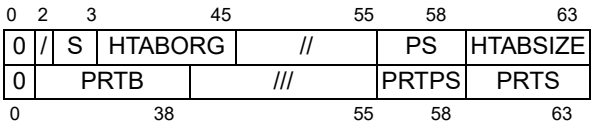
Add the S bit in the Partition Table Entry layouts, for use by outboard translation mechanisms.

---------- Begin text ----------

The Partition Table is composed of a pair of doublewords per partition. The first doubleword indicates whether the host uses HPT or Radix Tree translation and whether the partition is secure, and contains the base of the host's translation table structure in host real memory. The first doubleword also contains the size of the table structure and the size of the Root Page Directory for a hypervisor using Radix Tree translation, or the base page size for the VRMA for Paravirtualized HPT translation. Additional details about the parameters for HPT translation follow.

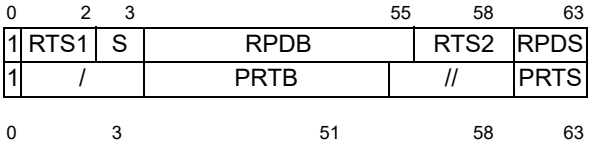
---------- End text ----------

---------- Begin text ----------.



Paravirtualized HPT Partition Table Entry

Bit(s)	Name	Description
0	HR	Host Radix 0b0- hypervisor uses HPT translation for this partition 0b1- hypervisor uses Radix Tree translation for this partition
3	S	Partition is Secure
4:45	HTABORG	Hashed Page Table Base
56:58	PS	Page Size (uses L||LP encoding as in current SLBE)
59:63	HTABSIZE	HPT size = $2^{HTABSIZE+18}$ HTABSIZE ≤ 28
0	GR	Guest Radix 0b0- partition uses HPT 0b1- partition uses Radix Tree
1:38	PRTB	Process Table Base (when UPRT=1)
56:58	PRTPS	Process Table Page SIze (when UPRT=1) (uses L||LP encoding as in current SLBE)
59:63	PRTS	Process Table Size = $2^{12+PRTS}$ PRTS≤24 (when UPRT=1)



Radix on Radix Partition Table Entry

Bit(s)	Name	Description
0	HR	Host Radix 0b0- hypervisor uses HPT translation for this partition 0b1- hypervisor uses Radix Tree translation for this partition
1:2	RTS1	Radix Tree Size[0:1]
3	S	Partition is Secure
4:55	RPDB	Root Page Directory Base
56:58	RTS2	Radix Tree Size[2:4] (number of address bits mapped), size=2^{RTS+31}
59:63	RPDS	Root Page Directory Size = 2^{RPDS+3}, RPDS≥5
0	GR	Guest Radix 0b0- partition uses HPT 0b1- partition uses Radix Tree
4:51	PRTB	Process Table Base

Radix on Radix Partition Table Entry

Bit(s)	Name	Description
59:63	PRTS	Process Table Size = $2^{12+PRTS}$ PRTS≤24 (when UPRT=1)

All other fields are reserved.

---------- End text ----------

---------- Begin text ----------

Programming Note

The S bit in Partition Table Entries is provided for use by outboard mechanisms that access storage. The processor uses MSR_S, not $PATE_S$, to determine partition security.

The size of the Process Table is provided to simplify hardware design and testing. The size enables the hardware to mask address bits instead of providing an adder. No size checking is provided. (An out-of-range PID will not produce an exception simply because of its size.) Hypervisor software may help detect such errors by the OS by not providing a translation for virtual / guest real addresses for a page or two beyond the end of the Process Table.

Similarly, no size checking is provided for the Partition Table. (An out-of-range LPID will not produce an exception simply because of its size.)

---------- End text ----------

Section 5.7.14 Storage Protection

Add to the intro to include secure memory. Also redo the storage protection cases to cover some omissions and improve clarity.

---------- Begin text ----------

The storage protection mechanism provides a means for selectively granting instruction fetch access, granting read access, granting write access, and prohibiting access to areas of storage based on a number of control criteria.

The operation of the storage protection mechanism depends on the value of one or more of the following.

- MSR bits HV, S, IR, DR, PR
- the key bits in the associated SLB entry
- the page protection bits and key bits in the associated PTE
- the AMR, IAMR, AMOR, and UAMOR
- the Secure Memory property

The storage protection mechanism consists of the Virtual Page Class Key Protection mechanism described in Section 5.7.14.1, the Basic Storage Protection mechanism described in Section 5.7.14.3 and Section

5.7.14.4, the Radix Tree Translation Storage Protection mechanism described in Section 5.7.14.5, and the Secure Memory Protection mechanism described in <crossref to SM section>.

In order for a storage access to be permitted, it must be permitted by all of the mechanisms that apply to it. If $SMFCTRL_E$=1, each storage access is subject to Secure Memory Protection independent of the translation mode of the access. In addition, each access is subject to other protection mechanisms depending on its translation mode, as listed below.

- MSR_{HV}=1 and address translation is disabled: Basic Storage Protection mechanism
- HR=0
  - access to instruction or data when address translation is enabled: Virtual Page Class Key Protection mechanism and Basic Storage Protection mechanism
  - all other cases (access to Process Table Entry or Segment Table Entry when address translation is enabled; access to instruction or data when MSR_{HV}=0 and address translation is disabled): Basic Storage Protection mechanism

Programming Note

Because the assumed K_s and K_p values are either 0 or irrelevant, these accesses are always permitted by the Basic Storage Protection mechanism.

- HR=1
  - access to instruction or data when address translation is enabled and effLPID≠0: Radix Tree Translation Storage Protection mechanisms of both the process-scoped and partition-scoped PTEs
  - access to instruction or data when address translation is enabled and effLPID=0: Radix Tree Translation Storage Protection mechanism of the process-scoped PTE
  - all other cases (access to Process Table Entry when address translation is enabled; access to process-scoped PDE or process-scoped PTE when address translation is enabled and effLPID≠0; access to instruction or data when MSR_{HV}=0 and address translation is disabled): Radix Tree Translation Storage Protection mechanism of the partition-scoped PTE

If an access associated with an instruction fetch is not permitted, an Instruction Storage exception or a Hypervisor Instruction Storage exception is generated. If an access associated with a data access is not permitted, a Data Storage exception or a Hypervisor Data Storage exception is generated.

---------- End text ----------

Section 5.7.14.5+ Secure Memory Protection

Add the following section after radix tree translation storage protection.

---------- Begin text ----------

When $SMFCTRL_E$=1, Secure Memory Protection is enabled. Each location in main storage has a Secure Memory property mem_{SM}. mem_{SM}=1 indicates secure memory. mem_{SM}=0 indicates ordinary memory. Generally, only secure partitions and the ultravisor may access secure memory for explicit and implicit accesses. The one exception is that the Partition Table is commonly located in secure memory, but may be accessed implicitly as part of the translation process for software running with MSR_S=0. The granularity and method with which main storage is mapped for the Secure Memory property is implementation specific.

For each kind of access to a host real address that can cause a violation of Basic or Radix Tree Translation Storage Protection, a Secure Memory Protection exception is reported by the same type of interrupt as its Basic or Radix Tree Translation Storage Protection counterpart, except setting [H]DSISR or [H]SRR1 bit 43 instead of 36, as follows. For HPT translation, the exception is reported as an ISI or DSI if the thread is in hypervisor state, or if the thread is in non-hypervisor state when IR or DR is 1 for the appropriate type of access and VPM=0; otherwise as HISI or HDSI. For Radix Tree translation, the exception is reported as an ISI or DSI if effLPID=0; otherwise as HISI or HDSI. The same reporting approach is used for accesses which require translation but for which no Basic Storage Protection exception is possible. This includes accesses to the Segment Table Entry Group and Process Table Entry when HPT translation is in use.

In the preceding cases the host real address for the access is a result of address translation. A Secure Memory Protection exception can also be caused by accesses to a host real address that is not the result of address translation. (Such accesses cannot cause a violation of Basic or Radix Tree Translation Storage Protection.) These additional cases are reported as follows. For a hypervisor real mode access the exception is reported as an ISI or DSI. For a process-scoped radix tree access for effLPID=0 the exception is reported as an ISI or DSI. For a PTEG access the exception is reported as an ISI or DSI if $MSR_{HV\ PR}$=0b10; otherwise as HISI or HDSI. For a partition-scoped radix tree access the exception is reported as an HISI or HDSI unless effLPID=0, in which case the exception is reported as an ISI or DSI. These cases also set [H]DSISR or [H]SRR1 bit 43 to 1.

---------- End text ----------

Section 5.8.2.2 Altering the Storage Control Bits

Change “ordinary storage” to “normal storage” (referring to WIMG=0010 equivalent storage).

---------- Begin text ----------

Programming Note

The storage control bit alterations described above are examples of cases in which the directives for application of statements about the W and I bits to SAO given in the third paragraph of the preceding subsection must be applied. A transition from the typical WIMG=0b0010 for normal storage to WIMG=0b1110 for SAO storage does not require the flush described above because both WIMG combinations indicate storage that is not Caching Inhibited.

---------- End text ----------

Section 5.9.2 Synchronize Instruction

Add msgsndu with the same requirements as msgsnd.

---------- Begin text ----------

The *Synchronize* instruction is described in Section 4.6.3 of Book II, but only at the level required by an application programmer. This section describes properties of the instruction that are relevant only to operating system, hypervisor, and ultravisor software programmers.

The *Synchronize* instruction provides an ordering function for stores that are in set A of the memory barrier created by the *Synchronize* instruction, relative to data accesses caused by instructions that are executed on other threads after the occurrence of the interrupt that is caused by a ***msgsndp***, ***msgsnd***, or ***msgsndu*** instruction that follows the *Synchronize* instruction. The thread that is the target of the ***msgsndp***, ***msgsnd***, or ***msgsndu*** instruction is here called the "target thread".

- For ***msgsndp***, and L = 0, 1, or 2 for the *Synchronize* instruction, the stores are performed with respect to the target thread before any data accesses caused by instructions that are executed on the target thread after the corresponding Directed Privileged Doorbell interrupt has occurred.
- For ***msgsnd*** or ***msgsndu***, and L = 0 or 2 for the *Synchronize* instruction (***sync*** or ***ptesync***), the stores are performed with respect to any given other thread before any data accesses caused by instructions that are executed on the given thread after a ***msgsync*** instruction is executed on that thread after the corresponding Directed Hypervisor or Ultravisor Doorbell interrupt has occurred on the target thread.

Programming Note

Synchronize with L=1 (l***wsync***) should not be used with ***msgsnd*** or ***msgsndu***. (If used, it will not have the desired ordering effect.)

Programming Note

The ***msgsync*** instruction, which is needed when ***msgsnd*** or ***msgsndu*** is used, is not needed when ***msgsndp*** is used because ***msgsndp*** targets only threads on the same multi-threaded processor as the thread executing the ***msgsndp***, while ***msgsnd*** and ***msgsndu*** can target any thread in the system. (If the target thread for ***msgsnd*** or ***msgsndu*** is on the same multi-threaded processor as the thread executing the ***msgsnd*** or ***msgsndu***, in principle the ***msgsync*** can be omitted. This optimization is practical only when the ***msgsnd***/***msgsndu*** topology is appropriately constrained, however, because the Directed Hypervisor or Ultravisor Doorbell interrupt provides no indication of which thread executed the ***msgsnd*** or ***msgsndu*** that caused the interrupt, so there is no easy way for the interrupt handler to determine whether the ***msgsync*** can be omitted.) ***msgsync*** is not needed or defined in V. 2.07 for a similar reason: ***msgsnd*** in V. 2.07 can target only threads on the same multi-threaded processor as the thread executing the ***msgsnd***.

The ordering done by ***sync*** (and ***ptesync***) provides the appearance of "causality" across a sequence of ***msgsnd*** (or ***msgsndu***) instructions, as in the following example. "***msgsnd***->T1" means "***msgsnd*** instruction targetting thread T1". "<DHDI 0>" means "occurrence of Directed Hypervisor Doorbell interrupt caused by ***msgsnd*** executed on T0". On T0, register r1 is assumed to contain the value 1.

```
T0              T1              T2
std  r1,X       <DHDI 0>        <DHDI 1>
sync            msgsnd->T2      msgsync
msgsnd->T1                      ld   r1,X
```

In this example, T2's load from X must return 1.

---------- End text ----------

Comment: Please disregard the change bar at the end of the preceding p-note.

Section 5.10 Page Table Update Synchronization Requirements

Make the following change in the Programming Note:

---------- Begin text ----------

Programming Note

In many cases this context synchronization will occur naturally; for example, if the sequence is executed within an interrupt handler the ***rfscv***, ***rfid,*** ***hrfid*** or ***urfid*** instruction that returns from the interrupt handler may provide the required context synchronization.

---------- End text ----------

6.2.2+ Ultravisor Machine Status Save/Restore Registers

---------- Begin text ----------

When a Directed Ultravisor Doorbell interrupt occurs, the state of the machine is saved in the Ultravisor Machine Status Save/Restore Registers (USRR0 and USRR1).

.

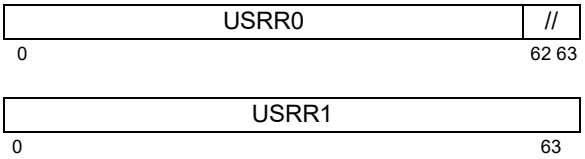


Figure 6. Ultravisor Save/Restore Registers

USRR1 bits may be treated as reserved in a given implementation if they correspond to MSR bits that are reserved or are treated as reserved in that implementation and, for USRR1 bits in the range 33:36 and 42:47, they are specified as being set either to 0 or to an undefined value by the Directed Ultravisor Doorbell interrupt.

The USRR0 and USRR1 are ultravisor resources; see <ultravisor chapter>.

---------- End text ----------

Section 6.2.12 Hypervisor Facility Status and Control Register

Add urfid to go with [h]rfid in the first p-note.

---------- Begin text ----------

Programming Note

Notice that ***rfebb***, ***rfscv***, ***rfid***, ***hrfid***, ***urfid***, and ***mtmsrd*** instructions can cause a TM Bad Thing type Program interrupt even when executed in a privilege state in which TM is made unavailable by the HFSCR. Here are two examples. Both assume that $HFSCR_{TM}$=0; the second assumes that $HFSCR_{EBB}$=1.

- An operating system, running with $MSR_{TS\ TM}$ = 0b000 (N0), sets $SRR1_{29:31}$ to 0b101 (T1) then executes ***rfid***. The attempted illegal transaction state transition will cause a TM Bad Thing type Program interrupt, despite the fact that TM is made unavailable in privileged non-hypervisor state by the HFSCR.
- An application program, running with $MSR_{TS\ TM}$ = 0b000 (N0), sets $BESCR_{TS}$ to 0b01 (S) then executes ***rfebb***. The attempted illegal transaction state transition will cause a TM Bad Thing type Program interrupt, despite the fact that TM is made unavailable in problem state by the HFSCR.

This anomaly cannot be caused by the PCR.

- ***rfscv***, ***rfid***, ***hrfid***, ***urfid***, and ***mtmsrd*** cannot be executed in the privilege state (problem state) in which TM is made unavailable by the PCR.
- ***rfebb*** can be executed in the privilege state in which TM is made unavailable by the PCR, but the PCR bit that makes TM unavailable (the v2.06 bit) also makes ***rfebb*** unavailable.

Another difference between the HFSCR and the PCR is that $PCR_{v2.06}$=1 prevents a thread from being simultaneously in problem state and in Transactional or Suspended state and $HFSCR_{TM}$=0 does not. However, if the hypervisor always returns to the partition in Non-transactional state when $HFSCR_{TM}$=0, the partition will be unable to enter Transactional or Suspended state.

---------- End text ----------

Section 6.3 Interrupt Synchronization

Add the USRRs along with the others.

---------- Begin text ----------

When an interrupt occurs, in general SRR0, HSRR0 or USRR0 is set to point to an instruction such that all preceding instructions have completed execution, no subsequent instruction has begun execution, and the instruction addressed by SRR0, HSRR0 or USRR0 may or may not have completed execution, depending on the interrupt type. The only exception is that if an ***mtspr*** sequence started by ***mtgsr*** is active when the interrupt occurs, some of the sequence's ***mtspr***s

beyond the instruction pointed to by SRR0, HSRR0, or USRR0 may have been executed; see Chapter 11.

---------- End text ----------

Section 6.4 Interrupt Classes

Add ultravisor doorbell to the list of system-caused interrupts.

---------- Begin text ----------

nterrupts are classified by whether they are directly caused by the execution of an instruction or are caused by some other system exception. Those that are “system-caused” are:

- System Reset
- Machine Check
- External
- Decrementer
- Directed Privileged Doorbell
- Hypervisor Decrementer
- Hypervisor Maintenance
- Hypervisor Virtualization
- Directed Hypervisor Doorbell
- Directed Ultravisor Doorbell
- Performance Monitor

External, Decrementer, Hypervisor Decrementer, Directed Privileged Doorbell, Directed Hypervisor Doorbell, Directed Ultravisor Doorbell, Hypervisor Maintenance, and Hypervisor Virtualization interrupts are maskable interrupts. Therefore, software may delay the generation of these interrupts. System Reset and Machine Check interrupts are not maskable.

“Instruction-caused” interrupts are further divided into two classes, *precise* and *imprecise*.

---------- End text ----------

Section 6.4.1 Precise Interrupt

Add USRR0 to bullet 1. HSRR0 was not previously included but should be.

---------- Begin text ----------

When the fetching or execution of an instruction causes a precise interrupt, the following conditions exist at the interrupt point.

1. SRR0, HSRR0 or USRR0 addresses either the instruction causing the exception or the immediately following instruction. Which instruction is addressed can be determined from the interrupt type and status bits.

---------- End text ----------

Section 6.4.3 Interrupt processing

Add USRR0 to bullet 1 and USRR1 to bullets 2 and 3 and USRR at the end of the last Programming Note.

---------- Begin text ----------

1. SRR0, HSRR0 or USRR0 is loaded with an instruction address that depends on the type of interrupt; see the specific interrupt description for details.
2. Bits 33:36 and 42:47 of SRR1, HSRR1 or USRR1 are loaded with information specific to the interrupt type.
3. Bits 0:32, 37:41, and 48:63 of SRR1, HSRR1 or USRR1 are loaded with a copy of the corresponding bits of the MSR.

...

Programming Note

Because interrupts that set the HSRRs preserve MSR_{RI} instead of setting it to 0 as is done by interrupts that set the SRRs, handlers for interrupts that set the HSRRs must prevent additional such interrupts from occurring until enough state has been saved that another such interrupt can be recovered from, and also when the HSRRs have been restored prior to executing ***hrfid***. Required behavior during those intervals includes the following.

- Keep $MSR_{HV\ PR\ EE}$=0b100. (This state prevents many such interrupts from occurring.)
- Execute only defined instructions that are not in invalid form.
- Pin the first page of the hypervisor’s Process Table
- Ensure that the PTE mapping the first page of the hypervisor’s Process Table has the Reference bit set and has no other reason to cause an exception.

Similarly, because the Directed Ultravisor Doorbell interrupt preserves MSR_{RI} instead of setting it to 0, the Directed Ultravisor Doorbell interrupt handler must prevent additional such interrupts from occurring until enough state has been saved that another such interrupt can be recovered from, and also when the USRRs have been restored prior to executing ***urfid***. This can be accomplished by keeping $MSR_{S\ HV\ PR\ EE}$=0b1100 during those intervals.

---------- End text ----------

Section 6.4.4 Implicit alteration of HSRR0 and HSRR1

Add urfid to the list of instructions that cannot be emulated, and limit the prohibition on emulation of hrfid to hypervisor state (a correction).

---------- Begin text ----------

Executing some of the more complex instructions may have the side effect of altering the contents of HSRR0 and HSRR1. The instructions listed below are guaranteed not to have this side effect. Any omission of instruction suffixes is significant; e.g., ***add*** is listed but ***add.*** is excluded.

1. *Branch* instructions

   b[***l***][***a***], ***bc***[***l***][***a***], ***bclr***[***l***], ***bcctr***[***l***]

2. *Fixed-Point Load* and *Store* Instructions

   lbz, ***lbzx***, ***lhz***, ***lhzx***, ***lwz***, ***lwzx***, ***ld***, ***ldx***, ***stb***, ***stbx***, ***sth***, ***sthx***, ***stw***, ***stwx***, ***std***, ***stdx***

   Execution of these instructions is guaranteed not to have the side effect of altering HSRR0 and HSRR1 only if the storage operand is aligned and $MSR_{HV\ DR}$=0b10.

3. *Arithmetic* instructions

   addi, ***addis***, ***add***, ***subf***, ***neg***

4. *Compare* instructions

   cmpi, ***cmp***, ***cmpli***, ***cmpl***

5. *Logical* and *Extend Sign* instructions

   ori, ***oris***, ***xori***, ***xoris***, ***and***, ***or***, ***xor***, ***nand***, ***nor***, ***eqv***, ***andc***, ***orc, extsb***, ***extsh***, ***extsw***

6. *Rotate and Shift* instructions

   rldicl, ***rldicr***, ***rldic***, ***rlwinm***, ***rldcl***, ***rldcr***, ***rlwnm***, ***rldimi***, ***rlwimi***, ***sld***, ***slw***, ***srd***, ***srw***

7. Other instructions

   isync

   rfid, ***urfid***

   hrfid in hypervisor state

   mtspr, ***mfspr***, ***mtmsrd***, ***mfmsr***

---------- End text ----------

Section 6.5 Interrupt Definitions

Add the S column on the right, add the new ultravisor interrupt type, and add various new entries to the footnotes in the MSR setting figure. Also add the new ultravisor interrupt type to the effective address figure and that offsets are also affected by MSR[S].

---------- Begin text ----------

Interrupt Type	**MSR Bit**								
	IR	DR	FE0	FE1	EE	RI	ME	HV	S
System Reset	0	0	0	0	0	0	p	1	t
Machine Check	0	0	0	0	0	0	0	1	t
Data Storage	r	r	0	0	0	0	-	-	-
Data Segment	r	r	0	0	0	0	-	-	-
Instruction Storage	r	r	0	0	0	0	-	-	-
Instruction Segment	r	r	0	0	0	0	-	-	-
External	r	r	0	0	0	h	-	e	-
Alignment	r	r	0	0	0	0	-	-	-
Program	r	r	0	0	0	0	-	-	-
FP Unavailable	r	r	0	0	0	0	-	-	-
Decrementer	r	r	0	0	0	0	-	-	-
Directed Privileged Doorbell	r	r	0	0	0	0	-	-	-
Hypervisor Decrementer	r	r	0	0	0	-	-	1	-
System Call	r	r	0	0	0	0	-	s	u
Trace	r	r	0	0	0	0	-	-	-
Hypervisor Data Storage	r	r	0	0	0	-	-	1	-
Hypervisor Instr. Storage.	r	r	0	0	0	-	-	1	-
Hypv Emulation Assistance	r	r	0	0	0	-	-	1	-
Hypervisor Maintenance	0	0	0	0	0	-	-	1	-
Directed Hypervisor Doorbell	r	r	0	0	0	-	-	1	-
Hypervisor Virtualization	r	r	0	0	0	0	-	1	-
Performance Monitor	r	r	0	0	0	0	-	-	-
Vector Unavailable	r	r	0	0	0	0	-	-	-
VSX Unavailable	r	r	0	0	0	0	-	-	-
Facility Unavailable	r	r	0	0	0	0	-	-	-
Hypervisor Facility Unavailable	r	r	0	0	0	-	-	1	-
Directed Ultravisor Doorbell	0	0	0	0	0	-	-	1	1
System Call Vectored	r	r	0	0	-	-	-	-	-

Interrupt Type	MSR Bit IR DR FE0 FE1 EE RI ME HV S
0 bit is set to 0 1 bit is set to 1 - bit is not altered r for interrupts that are taken as if $LPCR_{AIL}$= 3, and for interrupts for which $LPCR_{AIL}$ applies, if $LPCR_{AIL}$=2 or 3, set to 1; otherwise set to 0 p if the interrupt occurred while the thread was in power-saving mode, set to 1; otherwise not altered e if LPES=0, set to 1; otherwise not altered h if LPES=1, set to 0; otherwise not altered s if LEV=1 or LEV=2, set to 1; otherwise not altered t if the interrupt caused exit from a state-losing power-saving mode and $SMFCTRL_E$=1, set to 1; if the interrupt caused exit from a state-losing power-saving mode and $SMFCTRL_E$=0, set to 0; otherwise not altered u if $SMFCTRL_E$ =1 and LEV=2, set to 1; otherwise not altered *Settings for Other Bits* Bits BE, FP, PR, SE, TM, VEC, VSX, PMM, and bit 5 are set to 0. TM, FP, SLE, VEC, and VSX are set to 0. If the interrupt results in $MSR_{S\ HV}$ being equal to 0b11, the LE bit is copied from the UILE bit; otherwise, if the interrupt results in $MSR_{S\ HV}$ being equal to 0b01, the LE bit is copied from the HILE bit; otherwise the LE bit is copied from the $LPCR_{ILE}$ bit. The SF bit is set to 1. If the TS field contained 0b10 (Transactional) when the interrupt occurred, the TS field is set to 0b01 (Suspended); otherwise the TS field is not altered. Reserved bits are set as if written as 0.	

Figure 7. MSR setting due to interrupt

...

Effective Address[1]	Interrupt Type
00..0000_0100	System Reset
00..0000_0200	Machine Check
00..0000_0300	Data Storage
00..0000_0380	Data Segment
00..0000_0400	Instruction Storage
00..0000_0480	Instruction Segment
00..0000_0500	External
00..0000_0600	Alignment
00..0000_0700	Program
00..0000_0800	Floating-Point Unavailable
00..0000_0900	Decrementer
00..0000_0980	Hypervisor Decrementer
00..0000_0A00	Directed Privileged Doorbell
00..0000_0B00	Reserved
00..0000_0C00	System Call
00..0000_0D00	Trace
00..0000_0E00	Hypervisor Data Storage
00..0000_0E20	Hypervisor Instruction Storage
00..0000_0E40	Hypervisor Emulation Assistance
00..0000_0E60	Hypervisor Maintenance
00..0000_0E80	Directed Hypervisor Doorbell
00..0000_0EA0	Hypervisor Virtualization
00..0000_0EC0	Reserved
00..0000_0EE0	Reserved for implementation-dependent interrupt for performance monitoring
00..0000_0F00	Performance Monitor
00..0000_0F20	Vector Unavailable
00..0000_0F40	VSX Unavailable
00..0000_0F60	Facility Unavailable
00..0000_0F80	Hypervisor Facility Unavailable
00..0000_0FA0	Directed Ultravisor Doorbell
00..0000_0FC0	Reserved
. . .	. . .
00..0000_0FFF	Reserved
00..0001_7000	System Call Vectored
00..0001_7020	System Call Vectored
. . .	. . .
00..0001_7FE0	System Call Vectored
00..0001_7FFF	(end of ***scv*** interrupt vectors)

Effective Address[1]	Interrupt Type

[1] The values in the Effective Address column are interpreted as follows.

- 00...0000_0nnn *means* 0x0000_0000_0000_0nnn unless the values of HR, $LPCR_{AIL}$, and $MSR_{S\ HV\ IR\ DR}$ cause the application of an effective address offset. See the description of $LPCR_{AIL}$ in Section 2.2 for more details.
- 0...00_0001_7nnn *means* 0x0000_0000_0001_7nnn unless the values of HR, $LPCR_{AIL}$, and $MSR_{S\ HV\ IR\ DR}$ cause the usage of an alternate effective address. See the description of $LPCR_{AIL}$ in Section 2.2 for details.

Effective addresses 0x0000_0000_0000_0000 through 0x0000_0000_0000_00FF are used by software and will not be assigned as interrupt vectors.

Figure 8. Effective address of interrupt vector by interrupt type

---------- End text ----------

Section 6.5.1 System Reset Interrupt

Add directed ultravisor doorbell to the bulleted list near the beginning of the section. Add an SRR1 code for directed ultravisor doorbell wakeup.

---------- Begin text ----------

- External
- Decrementer
- Directed Privileged Doorbell
- Directed Hypervisor Doorbell
- Directed Ultravisor Doorbell
- Hypervisor Maintenance
- Hypervisor Virtualization exception
- Implementation-specific

---------- End text ----------

Delete the Architecture Note in the definition of SRR1[33]. (The Note, added in V. 2.07, says bit 33 will be required to be set to 0 “in the next version of the architecture”, and has been proven incorrect.)

---------- Begin text ----------

SRR1

33 Implementation-dependent.

34:36 Set to 0.

42:45 If the interrupt did not occur when the thread was in power-saving mode, set to an

implementation-specific value. If the interrupt occurred when the thread was in power-saving mode, set to indicate the exception that caused exit from power-saving mode as shown below:

$SRR1_{42:45}$	Exception
0000	Reserved
0001	Directed Ultvsr Doorbell
0010	Implementation specific
0011	Directed Hypvsr Doorbell
0100	System Reset
0101	Directed Privlgd Doorbell
0110	Decrementer
0111	Reserved
1000	External
1001	Hypervisor Virtualization
1010	Hypervisor Maintenance
1011	Reserved
1100	Implementation specific
1101	Reserved
1110	Implementation specific
1111	Reserved

---------- End text ----------

Editorial note: The abbreviations of the names of the storage interrupts are added to the section names.

Section 6.5.3 Data Storage Interrupt (DSI)

Fix words for hypervisor real mode.

---------- Begin text ----------

A Data Storage interrupt occurs when no higher priority exception exists and either

(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)) and the data access cannot be performed, or

(c) HPT translation is being performed, the value of the expression

$((MSR_{HV\ PR}=0b10) \mid ((\neg VPM \mid \neg PRTE_V) \& MSR_{DR}))$

is 1, and a data access cannot be performed,

except for the case of $MSR_{HV\ PR} \neq 0b10$,

VPM=0, $LPCR_{KBV}$=1, and a Virtual Page Class Key Storage 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, a problem other than page fault occurs attempting to access the LPID=0 process table, or process-scoped translation prevents the data access from being performed

for any of the following reasons that can occur in the respective translation state except for a PTEG access causing a secure memory exception when VPM=0. (In the expression for (c) above, "$\neg PRTE_V$" is shorthand representing the case of an invalid segment table descriptor stopping the translation process.)

---------- End text ----------

Add radix storage protection (an oversight) and secure memory protection.

---------- Begin text ----------

- The access violates Basic Storage Protection.
- The access violates Virtual Page Class Key Storage Protection and $LPCR_{KBV}$=0.
- The access violates Radix Tree Translation Storage Protection.
- The access violates Secure Memory Protection.
- The process- and partition-scoped page attributes conflict.

---------- End text ----------

---------- Begin text ----------

42 Set to 1 if the access is not permitted by virtual page class key protection; otherwise set to 0.

43 Set to 1 if the access is not permitted by Secure Memory Protection; otherwise set to 0.

44 Set to 1 if an unsupported radix tree configuration is found during the translation process; otherwise set to 0.

---------- End text ----------

Make a miscellaneous repair.

---------- Begin text ----------

If multiple Data Storage exceptions occur for a given effective address, any one or more of the bits corresponding to these exceptions may be set to 1 in the DSISR. However, if one or more Data Storage exceptions occur together with a Virtualized Page Class Key Storage Protection exception that occurs when $LPCR_{KBV}$=1 and Virtualized Partition Memory is disabled by VPM=0, an HDSI results, and all of the exceptions are reported in the HDSISR.

---------- End text ----------

Section 6.5.5 Instruction Storage Interrupt (ISI)

Add the steering for hypervisor real mode.

---------- Begin text ----------

An Instruction Storage interrupt occurs when no higher priority exception exists and either

(a) ($MSR_{HV\ PR}$=0b10) & (MSR_{IR}=0)) and the next instruction to be executed cannot be fetched, or

(b) 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

(c) Radix Tree translation is being performed and either a problem other than page fault occurs attempting to access the LPID=0 process table or process-scoped translation prevents the next instruction to be executed from being fetched

for any of the following reasons that can occur in the respective translation state except for a PTEG access causing a secure memory exception when VPM=0. (In the expression for (b) above, "¬PRTE$_V$" is shorthand representing the case of an invalid segment table descriptor stopping the translation process.)

---------- End text ----------

Add secure memory protection. (The presentation is made more consistent with that for DSI.)

---------- Begin text ----------

- The address of the appropriate process table entry or segment table entry group cannot be translated when HR=0 and either VPM=0 or the process table entry is invalid (independent of VPM).
- The access violates Basic Storage Protection.
- The access violates Virtual Page Class Key Storage Protection.
- The access violates Radix Tree Translation Storage Protection.
- The access violates Secure Memory Protection.
- The process- and partition-scoped page attributes conflict.

---------- End text ----------

---------- Begin text ----------

42 Set to 1 if the access is not permitted by virtual page class key protection; otherwise set to 0.

43 Set to 1 if the access is not permitted by Secure Memory Protection; otherwise set to 0.

44 Set to 1 if an unsupported radix tree configuration is found during the translation process; otherwise set to 0.

---------- End text ----------

Section 6.5.9 Program Interrupt

Update the privileged instruction type of program interrupt to account for ultravisor privileged resources.

---------- Begin text ----------

Privileged Instruction

The following applies if the instruction is executed when MSR_{PR} = 1.

A Privileged Instruction type Program interrupt is generated when execution is attempted of a privileged instruction, or of an ***mtspr*** or ***mfspr*** instruction with an SPR field that contains a value having spr_0=1.

The following applies if the instruction is executed when $MSR_{HV\ PR}$ = 0b00 and $LPCR_{EVIRT}$=0.

A Privileged Instruction type Program interrupt is generated when 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 or that specifies PTCR, DAWR0, DAWRX0, or CIABR when those SPRs are ultravisor privileged for the operation.

The following applies if the instruction is executed when $MSR_{HV\ PR}$ = 0b00 or when $MSR_{S\ HV\ PR}$ = 0b010.

A Privileged Instruction type Program interrupt is generated when execution is attempted of an ultravisor privileged instruction, or of an ***mtspr*** or ***mfspr*** instruction that specifies an SPR, other than PTCR, DAWR0, DAWRX0, and CIABR, that is ultravisor privileged for the operation.

---------- End text ----------

Update the last p-note in the section to include ultravisor-privileged resources.

---------- Begin text ----------

Programming Note

When $LPCR_{EVIRT}$=1, some of the conditions that cause a Privileged Instruction type Program interrupt when $LPCR_{EVIRT}$=0 (attempted execution, in privileged but non-hypervisor state, of a hypervisor privileged instruction or of an ***mtspr*** or ***mfspr*** instruction specifying an SPR that is hypervisor privileged for the operation or PTCR, DAWR0, DAWRX0, or CIABR when they are ultravisor privileged for the operation) instead cause a Hypervisor Emulation Assistance interrupt. Having these cases cause a Hypervisor Emulation Assistance interrupt permits support of nested hypervisors through virtualization of hypervisor facilities, and simplifies creation of a common kernel for the OS and the hypervisor. Some operating systems may still have code to handle these conditions, at the Program interrupt vector location. For this reason, if a Hypervisor Emulation Assistance interrupt occurs with $HSRR1_{45}$=1 and the hypervisor is not providing either of these functions, the hypervisor should pass control to the operating system at the operating system's Program interrupt vector location, with all registers (SRR0, SRR1, MSR, GPRs, etc.) set as if the instruction had caused a Privileged Instruction type Program interrupt, including setting $SRR1_{3\ 49}$ to 0b00.

---------- End text ----------

Section 6.5.14 System Call Interrupt

Add SRR1 bits to distinguish ucall from hcall from syscall.

---------- Begin text ----------

A System Call interrupt occurs when a *System Call* instruction is executed.

The following registers are set:

SRR0 Set to the effective address of the instruction following the System Call instruction.

SRR1

33:36 Set to 0.

42:43 Set to indicate the LEV value specified by the *System Call* instruction that caused the interrupt, as follows.

LEV	$SRR1_{42:43}$
0	00
1	01
2	10
3*	undefined
* reserved LEV value	

44:47 Set to 0.

Others Loaded from the MSR.

MSR See Figure 7 on page 35.

Execution resumes at effective address 0x0000_0000_0000_0C00, possibly offset as specified in Figure 7.

Programming Note

An attempt to execute an ***sc*** instruction with LEV=1 or LEV=2 in problem state, or an attempt to execute an ***sc*** instruction with LEV=2 in privileged non-hypervisor state, should be treated as a programming error.

An attempt to execute an ***sc*** instruction with LEV=2 when $SMFCTRL_E$=0 should be treated as a programming error.

---------- End text ----------

Section 6.5.15 Trace Interrupt

Add urfid to join the other *rfid's.

---------- Begin text ----------

A Trace interrupt occurs when no higher priority exception exists and any instruction except ***rfid***, ***hrfid, urfid, rfscv,*** or a *Power-Saving Mode* instruction is successfully completed, provided any of the following is true:

...

Execution resumes at effective address 0x0000_0000_0000_00D0, possibly offset as specified in Figure 70. For a Trace interrupt resulting from execution of an instruction that modifies the value of MSR_{IR}, MSR_{DR}, $MSR_{S\ HV}$, or $LPCR_{AIL\ HR}$, the Trace interrupt vector location is based on the modified values.

Programming Note

The following instructions are not traced.

- ***rfid***
- ***hrfid***
- ***urfid***
- ***rfscv***
- ***sc***, ***scv***, and *Trap* instructions that trap
- *Power-Saving Mode* instructions
- other instructions that cause interrupts (other than Trace interrupts)
- the first instructions of any interrupt handler
- instructions that are emulated by software
- instructions, executed in Transactional state, that are disallowed in Transactional state
- instructions, executed in Transactional state, that cause types of accesses that are disallowed in Transactional state
- ***mtspr***, executed in Transactional state, specifying an SPR that is not part of the Transactional Memory checkpointed registers
- ***tbegin.*** executed at maximum nesting depth

In general, interrupt handlers can achieve the effect of tracing these instructions.

---------- End text ----------

Section 6.5.16 Hypervisor Data Storage Interrupt (HDSI)

Make editorial wording improvement.

---------- Begin text ----------

(b) HPT translation is being performed and either a PTEG access causes a secure memory exception or 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 either a page fault occurs on the LPID=0 process table or

partition-scoped translation other than for the LPID=0 process table prevents an access from being performed

---------- End text ----------

Add secure memory protection.

---------- Begin text ----------

- The access violates storage protection. In addition to the legacy VPM cases (including those for Secure Memory Protection), this includes mismatches in access authority in which the process-scoped PTE permits the access but the partition-scoped PTE does not and Secure Memory Protection for a radix guest. It also includes lack of necessary authority for accesses to process-scoped tables, for example lack of write authority to set a reference bit in the process-scoped PTE (and Secure Memory Protection here as well). (In such a case, the "access" reported as failing would be the access to the process-scoped table. The HDAR would provide the guest real / (abbreviated) virtual address of the table entry.)

---------- End text ----------

---------- Begin text ----------

42 Set to 1 if the access is not permitted by virtual page class key protection; otherwise set to 0.

43 Set to 1 if the access is not permitted by Secure Memory Protection; otherwise set to 0.

44 Set to 1 if an unsupported MMU configuration is found during the translation process.

---------- End text ----------

Section 6.5.17 Hypervisor Instruction Storage Interrupt (HISI)

Make editorial wording improvement.

---------- Begin text ----------

A Hypervisor Instruction Storage interrupt occurs when no higher priority exception exists, either the 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 and either a PTEG access causes a secure memory exception or 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 for any of the following reasons, or

(b) Radix Tree translation is being performed and either a page fault occurs on the LPID=0 process table or

partition-scoped translation other than for the LPID=0 process table prevents the next

instruction to be executed from being fetched for any

of the following reasons that can occur in the respective translation state.

---------- End text ----------

Add secure memory protection.

---------- Begin text ----------

- The access violates storage protection. In addition to the legacy VPM cases (including those for Secure Memory Protection), this includes mis-

matches in access authority in which the process-scoped PTE permits the access but the partition-scoped PTE does not and Secure Memory Protection for a radix guest. It also includes lack of necessary authority for accesses to process-scoped tables, for example lack of write authority to set a reference bit in the process-scoped PTE (and Secure Memory Protection here as well). (In such a case, the “access” reported as failing would be the access to the process-scoped table. The HDAR would provide the guest real / (abbreviated) virtual address of the table entry.)

---------- End text ----------

---------- Begin text ----------

42 Set to 1 if the access is not permitted by virtual page class key protection; otherwise set to 0.

43 Set to 1 if the access is not permitted by Secure Memory Protection; otherwise set to 0.

44 Set to 1 if an unsupported MMU configuration is found during the translation process.

---------- End text ----------

Secction 6.5.18 Hypervisor Emulation Assistance Interrupt

---------- Begin text ----------

A Hypervisor Emulation Assistance interrupt is generated when execution is attempted of an illegal instruction, or of a reserved instruction or an instruction that is not provided by the implementation. It is also generated under the following conditions.

- When $MSR_{HV\ PR}$=0b00 and $LPCR_{EVIRT}$=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 or that specifies PTCR, DAWR0, DAWRX0, or CIABR when those SPRs are ultravisor privileged for the operation.
- When $MSR_{S\ HV\ PR}$ = 0b010, execution is attempted of an ***mtspr*** or ***mfspr*** instruction that specifies PTCR, DAWR0, DAWRX0, or CIABR when those SPRs are ultravisor privileged for the operation.
- When MSR_{PR}=1, execution is attempted of an ***mtspr*** or ***mfspr*** instruction that specifies an SPR with spr_0=0 that is not provided by the implementation.

...

HSRR1

33:36 Set to 0.

42:44 Set to 0.

45 Set to 1 for an attempt, when $MSR_{HV\ PR}$ = 0b00 and $LPCR_{EVIRT}$=1, to execute a hypervisor privileged instruction or an ***mtspr*** or ***mfspr*** instruction that specifies an SPR that is hypervisor privileged for the operation or that specifies PTCR, DAWR0, DAWRX0, or CIABR when they are ultravisor privileged for the operation, or for an attempt when $MSR_{S\ HV\ PR}$ = 0b010 to execute an ***mtspr*** or ***mfspr*** instruction that specifies PTCR, DAWR0, DAWRX0, or CIABR when they are ultravisor privileged for the operation; otherwise set to 0.

46:47 Set to 0.

Others Loaded from the MSR.

---------- End text ----------

Admit that the big p-note ignores SMF Extend the first paragraph as follows.

---------- Begin text ----------

This Programming Note illustrates how Hypervisor Emulation Assistance interrupts should be handled by software, including in environments that support nested hypervisors. For simplicity, this Programming Note ignores effects of the SMF facility (equivalently, assumes that $SMFCTRL_E$=0).

---------- End text ----------

Section 6.5.27+ Directed Ultravisor Doorbell Interrupt

Add ultravisor IPIs.

---------- Begin text ----------

A Directed Ultravisor Doorbell interrupt occurs when no higher priority exception exists, $SMFCTRL_E$=1, a Directed Ultravisor Doorbell exception is present, and the value of the following expression is 1.

(MSR_{EE} | ¬($MSR_{S\ HV\ PR}$=0b110)

Directed Ultravisor Doorbell exceptions are generated when Directed Ultravisor Doorbell messages (see Chapter 10) are received and accepted by the thread.

The following registers are set:

USRR0 Set to the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present.

USRR1

33:36 Set to 0.

42:47 Set to 0.

Others Loaded from the MSR.

MSR See Figure 7 on page 35.

Execution resumes at effective address 0x0000_0000_0000_0FA0.

---------- End text ----------

Section 6.7.2 Ordered Exceptions

Include new ultravisor exceptions in the ordering. The ultravisor doorbell goes above HMI and the other doorbells and security goes with “other [H]DSI and with all [H]ISI.

---------- Begin text ----------

The exceptions listed here are ordered with respect to the state of the interrupt processing mechanism. With one exception, in the following list the hypervisor forms of the Data Storage and Instruction Storage exceptions can be substituted for the non-hypervisor forms since the hypervisor forms cannot be caused by the same instruction and have the same ordering. The exception is that Virtual Page Class Key Storage Protection exceptions that occur when $LPCR_{KBV}$=1 and Virtualized Partition Memory is disabled by VPM_1=0 cause only a Hypervisor Data Storage exception (and never a Data Storage exception).

System-Caused or Imprecise

1. Program
   - Imprecise Mode Floating-Point Enabled Exception
2. Directed Ultravisor Doorbell
3. Hypervisor Maintenance
4. Hypervisor Virtualization, External, [Hypervisor] Decrementer, Performance Monitor, Directed Privileged Doorbell, Directed Hypervisor Doorbell

Instruction-Caused and Precise

1. Instruction Segment
2. [Hypervisor] Instruction Storage
3.a Hypervisor Emulation Assistance
3.b Program
   - Privileged Instruction
4. Function-Dependent
   4.a Fixed-Point and Branch
      1 Hypervisor Facility Unavailable
      2 Facility Unavailable
      3a Program
         - Trap
         - TM Bad Thing
      3b System Call or System Call Vectored
      3c.1 Data Storage for the case of *Fixed-Point Load or Store Caching Inhibited* instructions with MSR_{DR}=1 or the case of an invalid function code for an Atomic Memory Operation
      3c.2 all other Data Storage, Hypervisor Data Storage, [Hypervisor] Data Segment, or Alignment
      4 Trace
   4.b Floating-Point
      1 Hypervisor Facility Unavailable
      2 FP Unavailable
      3a Program
         - Precise Mode Floating-Pt Enabled Excep’n
      3b [Hypervisor] Data Storage, [Hypervisor] Data Segment, or Alignment
      4 Trace
   4.c Vector
      1 Hypervisor Facility Unavailable
      2 Vector Unavailable
      3a [Hypervisor] Data Storage, [Hypervisor] Data Segment, or Alignment
      4 Trace
   4.d VSX
      1 Hypervisor Facility Unavailable
      2 VSX Unavailable
      3a Program
         - Precise Mode Floating-Pt Enabled Excep’n
      3b [Hypervisor] Data Storage, [Hypervisor] Data Segment, or Alignment
      4 Trace
   4.e Other Instructions
      1 Hypervisor Facility Unavailable
      2 Facility Unavailable
      3a [Hypervisor] Data Storage, [Hypervisor] Data Segment, or Alignment
      4 Trace

For implementations that execute multiple instructions in parallel using pipeline or superscalar techniques, or combinations of these, it can be difficult to understand the ordering of exceptions.To understand this ordering it is useful to consider a model in which each instruction is fetched, then decoded, then executed, all before the next instruction is fetched. In this model, the exceptions a single instruction would generate are in the order shown in the list of instruction-caused exceptions. Exceptions with different numbers have different ordering. Exceptions with the same numbering but different lettering are mutually exclusive and cannot be caused by the same instruction. The Hypervisor Virtualization, External, [Hypervisor] Decrementer, Performance Monitor, Directed Privileged Doorbell, and Directed Hypervisor Doorbell interrupts have equal ordering. Similarly, where Data Storage, Data Segment, and Alignment exceptions are listed in the same item they have equal ordering.

Even on threads that are capable of executing several instructions simultaneously, or out of order, instruction-caused interrupts (precise and imprecise) occur in program order.

---------- End text ----------

Section 6.9 Interrupt Priorities

Add urfid to the other *rfid’s, add ultravisor doorbell (list numbers are wrong ...sorry). Security faults are silently added to other [H]DSIs and [H]ISIs.

Also, cleanup old oversights: add rfebb and mtmsr and restore rfscv (which was removed in error) to HEAI, add HEAI to the TM instructions,

---------- Begin text ----------

H. TM instruction, ***mt/fspr*** specifying TM SPR

a. These exceptions are mutually exclusive and have the same priority:
   - Program - Privileged Instruction (only for ***treclaim.*** and ***trechkpt.***)
   - Hypervisor Emulation Assistance

b. Hypervisor Facility Unavailable
c. Facility Unavailable
d. Program - TM Bad Thing (only for ***treclaim.***, ***trechkpt.***, and ***mtspr***)
e. Trace

I. ***rfebb***, ***rfscv***, ***rfid, hrfid, urfid,*** and ***mtmsr***[***d***]
   a. These exceptions are mutually exclusive and have the same priority:
      - Program - Privileged Instruction, for all except ***rfebb***
      - Hypervisor Emulation Assistance, for ***rfebb***, ***rfscv***, ***hrfid*** and ***mtmsr***

   b. Hypervisor Facility Unavailable (***rfebb*** only)
   c. Facility Unavailable (***rfebb*** only)
   d Program - TM Bad Thing for all except ***mtmsr***.
   e. Program - Floating-Point Enabled Exception or all except ***rfebb***
   f. Trace, for ***rfebb*** and ***mtmsr***[***d***] only

J. Other Instructions
   a. These exceptions are mutually exclusive and have the same priority:
      - Program - Trap
      - System Call
      - System Call Vectored
      - Program - Privileged Instruction
      - Hypervisor Emulation Assistance

   b. Hypervisor Facility Unavailable
   c. Facility Unavailable
   d. Trace

K. [Hypervisor] Instruction Storage and Instruction Segment

These exceptions have the lowest priority in this category. They are recognized only when all instructions prior to the instruction causing one of these exceptions appear to have completed and that instruction is the next instruction to be executed. The two exceptions are mutually exclusive.

The priority of these exceptions is specified for completeness and to ensure that they are not given more favorable treatment. It is acceptable for an implementation to treat these exceptions as though they had a lower priority.

5. Program - Imprecise Mode Floating-Point Enabled Exception

   This exception is the fourth highest priority exception. When this exception is created, the interrupt mechanism waits for all other possible exceptions to be reported. It then generates this interrupt if no higher priority exception exists when the interrupt is to be generated.

6. Directed Ultravisor Doorbell

   This exception is the fifth highest priority exception. When this exception is created, the interrupt mechanism waits for all other possible exceptions to be reported. It then generates this interrupt if no higher priority exception exists when the interrupt is to be generated.

Programming Note

Some platform implementations may depend on timely servicing of Hypervisor Maintenance interrupts, e.g. to prevent physical damage. The Directed Ultravisor Doorbell interrupt handler may test the HMER to identify such circumstances and take appropriate action.

5. Hypervisor Maintenance

   This exception is the sixth highest priority exception. When this exception is created, the interrupt mechanism waits for all other possible exceptions to be reported. It then generates this interrupt if no higher priority exception exists when the interrupt is to be generated.

   If a Hypervisor Maintenance exception exists and each attempt to execute an instruction when the Hypervisor Maintenance interrupt is enabled causes an exception (see the Programming Note below), the Hypervisor Maintenance interrupt is not delayed indefinitely.

6. Hypervisor Virtualization, Direct External, Mediated External, and [Hypervisor] Decrementer, Performance Monitor, Directed Privileged Doorbell, Directed Hypervisor Doorbell

   These exceptions are the lowest priority exceptions. All have equal priority (i.e., the hardware may generate any one of the corresponding interrupts for which an exception exists). When one of these exceptions is created, the interrupt processing mechanism waits for all other possible exceptions to be reported. It then generates the corresponding interrupt if no higher priority exception exists when the interrupt is to be generated.

---------- End text ----------

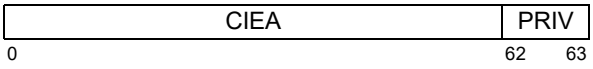
Section 8.3 Completed Instruction Address Breakpoint

Add UV debug control.

---------- Begin text ----------

The Completed Instruction Address Breakpoint mechanism provides a means of detecting an instruction completion at a specific instruction address. The address comparison is done on an effective address (EA).

The Completed Instruction Address Breakpoint mechanism is controlled by the Completed Instruction Address Breakpoint Register (CIABR) shown in Figure 9, except that if $SMFCTRL_D$=1 when PRIV≠0, the Privilege specification in the PRIV field is ignored and the facility detects instruction address matches in ultravisor state.



Bit(s)	Name	Description
0:61	CIEA	Completed Instruction Effective Address
62:63	PRIV	Privilege (PRIV > 0b00 ignored when $SMFCTRL_D$=1) 00: Disable matching 01: Match in problem state 10: Match in privileged non-hypervisor state 11: Match in hypervisor non-ultravisor state

Figure 9. Completed Instruction Address Breakpoint Register

A Completed Instruction Address Breakpoint match occurs upon instruction completion if all of the following conditions are satisfied. The values of CIABR, SMFCTRL, and the MSR that are used for the comparisons are those that exist at the time the instruction is initiated.

- the completed instruction address is equal to $CIEA_{0:61}$ || 0b00.
- $SMFCTRL_D$=0 and the thread privilege matches that specified in PRIV or $SMFCTRL_D$=1, PRIV≠0, and $MSR_{S\ HV\ PR}$=0b110.

In 32-bit mode the high-order 32 bits of the EA are treated as zeros for the purpose of detecting a match.

A Completed Instruction Address Breakpoint match causes a Trace exception provided that no higher priority interrupt occurs from the completion of the instruction (see Section 6.5.15).

---------- End text ----------

Section 8.4 Data Address Watchpoint

Add UV debug control.

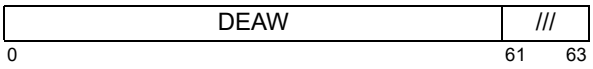
---------- Begin text ----------

The Data Address Watchpoint mechanism provides a means of detecting load and store accesses to a range of addresses starting at a designated doubleword. The address comparison is done on an effective address (EA).

Programming Note

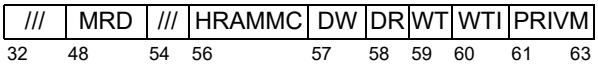
The Data Address Watchpoint mechanism employs a simple EA compare. It makes no attempt to take the radix table translation quadrants (keyed off $EA_{0:1}$) into account to enable a single setting to work in all privilege levels.

The Data Address Watchpoint mechanism is controlled by a single set of SPRs, numbered with n=0: the Data Address Watchpoint Register (DAWRn), shown in Figure 10, and the Data Address Watchpoint Register Extension (DAWRXn), shown in Figure 11. $SMFCTRL_D$ functions as an extension to the PRIVM field: when $SMFCTRL_D$=1, the facility detects data address watchpoint matches in ultravisor state in addition to states enabled by the PRIVM field.



Bit(s)	Name	Description
0:60	DEAW	Data Effective Address Watchpoint

Figure 10. Data Address Watchpoint Register



Bit(s)	Name	Description
48:53	MRD	Match Range in Doublewords biased by -1. (0b000000 = 1 DW, 0b111111 = 64 DW)
56	HRAMMC	Hypervisor Real Addressing Mode Match Control 0: $DEAW_0$ and EA_0 are used during matching in ultravisor or hypervisor real addressing mode 1: $DEAW_0$ and EA_0 are ignored during matching in ultravisor or hypervisor real addressing mode
57	DW	Data Write
58	DR	Data Read
59	WT	Watchpoint Translation
60	WTI	Watchpoint Translation Ignore
61:63	PRIVM	Privilege Mask
61	HYP	Hypervisor non-ultravisor state
62	PNH	Privileged Non-Hypervisor state
63	PRO	Problem state

All other fields are reserved.

Figure 11. Data Address Watchpoint Register Extension

The supported PRIVM values are 0b000, 0b001, 0b010, 0b011, 0b100, and 0b111 when $SMFCTRL_D$=0 and 0b000, 0b001, 0b010, and 0b011 when $SMFCTRL_D$=1. If the combination of $SMFCTRL_D$ and the PRIVM field does not contain one of the supported values, then whether a match occurs for a given storage

access is undefined. Elsewhere in this section it is assumed that the PRIVM field contains one of the supported values.

Programming Note

When $SMFCTRL_D$=0, PRIVM value 0b000 causes matches not to occur regardless of the contents of other DAWRn and DAWRXn fields. PRIVM values 0b101 and 0b110 are not supported because a storage location that is shared between the hypervisor and non-hypervisor software is unlikely to be accessed using the same EA by both the hypervisor and the non-hypervisor software. (PRIVM value 0b111 is supported primarily for reasons of software compatibility with respect to emulation of the DABR facility as described in a subsequent Programming Note.)

$SMFCTRL_D$=1 is provided for ultravisor debugging and also for ultravisor supervision of secure partition debugging. When $SMFCTRL_D$=1, exceptions due to matches that occur in hypervisor non-ultravisor state are unlikely to be desirable.

A Data Address Watchpoint match occurs for a *Load* or *Store* instruction if, for any byte accessed, all of the following conditions are satisfied. For the first condition, chk_DEAW and chk_EA are defined as follows. If $MSR_{HV\ DR}$=0b10 and HRAMMC=1 then

chk_DEAW = 0b0 || $DEAW_{1:60}$ and
chk_EA = 0b0 || $EA_{1:63}$;

otherwise

chk_DEAW = DEAW and
chk_EA = EA.

- the access is
  - a quadword access and located in the range ($chk\_DEAW_{0:59}$ || 0b0) ≤ ($chk\_EA_{0:59}$ || 0b0) ≤ (($chk\_DEAW_{0:59}$ || 0b0) + ($^{55}0$ || $MRD_{0:4}$|| 0b0)) such that ($chk\_EA_{0:60}$ AND ($^{55}1$ || $^{6}0$)) = ($chk\_DEAW_{0:60}$ AND ($^{55}1$ || $^{6}0$)).
  - not a quadword access and located in the range $chk\_DEAW_{0:60}$ ≤ $chk\_EA_{0:60}$ ≤ ($chk\_DEAW_{0:60}$ + ($^{55}0$ || $MRD_{0:5}$)) such that ($chk\_EA_{0:60}$ AND ($^{55}1$ || $^{6}0$)) = ($chk\_DEAW_{0:60}$ AND ($^{55}1$ || $^{6}0$)).
- (MSR_{DR} = $DAWRXn_{WT}$) | $DAWRXn_{WTI}$
- the thread is in
  - ultravisor state and $SMFCTRL_D$=1, or
  - hypervisor non-ultravisor state and $DAWRXn_{HYP}$ = 1, or
  - privileged non-hypervisor state and $DAWRXn_{PNH}$ = 1, or
  - problem state and $DAWRXn_{PR}$ = 1
- the instruction is a *Store* and $DAWRXn_{DW}$ = 1, or the instruction is a *Load* and $DAWRXn_{DR}$ = 1.

In 32-bit mode the high-order 32 bits of the EA are treated as zeros for the purpose of detecting a match.

If the above conditions are satisfied, it is undefined whether a match occurs in the following cases.

- The instruction is *Store Conditional* but the store is not performed
- The instruction is ***dcbz***. (For the purpose of determining whether a match occurs, ***dcbz*** is treated as a *Store*.)

The *Cache Management* instructions other than ***dcbz*** never cause a match.

A Data Address Watchpoint match causes a Data Storage exception or a Hypervisor Data Storage exception (see Section 6.5.3, “Data Storage Interrupt” on page 1114 and Section 6.5.16, “Hypervisor Data Storage Interrupt” on page 1124). If a match occurs, some or all of the bytes of the storage operand may have been accessed; however, if a *Store* instruction causes the match, the storage operand is not modified if the instruction is one of the following:

- any *Store* instruction that causes an atomic access

---------- End text ----------

Section 10.1 Overview

---------- Begin text ----------

The Processor Control facility provides a mechanism for the ultravisor or hypervisor to send messages to other threads in the system. Privileged non-hypervisor programs are able to send messages to other threads on the same multi-threaded processor; however if the processor is configured into sub-processors, privileged non-hypervisor programs can only send messages to other threads on the same sub-processor.

---------- End text ----------

Section 10.2 Programming Model

---------- Begin text ----------

Ultravisor-level, hypervisor-level, and privileged-level messages can be sent. Ultravisor-level messages are sent using the ***msgsndu*** instruction and cause ultravisor-level exceptions when received. Hypervisor-level messages are sent using the ***msgsnd*** instruction and cause hypervisor-level exceptions when received. Privileged-level messages are sent using the ***msgsndp*** instruction and cause privileged-level exceptions when received. For all three instructions, the message type and destination threads are specified in a General Purpose Register.

If a message is received by a thread, the exception corresponding to the message type is generated. When the exception is generated, the corresponding interrupt occurs when no higher priority exception exists and the interrupt is enabled (MSR_{EE}=1 for the Directed Privileged Doorbell interrupt, MSR_{EE}=1 or MSR_{HV}=0 for the Directed Hypervisor Doorbell interrupt, and MSR_{EE}=1

or $MSR_{S\ HV\ PR}$≠0b110 for the Directed Ultravisor Doorbell interrupt).

A Directed Privileged Doorbell exception remains until the corresponding interrupt occurs, or the exception is cleared by execution of a ***mtspr***(DPDES) or ***msgclrp*** instruction.

A Directed Hypervisor Doorbell exception remains until the corresponding interrupt occurs, or the exception is cleared by execution of a ***msgclr*** instruction.

A Directed Ultravisor Doorbell exception remains until the corresponding interrupt occurs, or the exception is cleared by execution of a ***msgclru*** instruction.

If a Doorbell exception of a given privilege is present and the corresponding interrupt is pended because MSR_{EE}=0, additional Doorbell exceptions of that privilege are ignored until the exception is cleared.

---------- End text ----------

Section 10.3.1 Directed Privileged Doorbell Exception State

Update the p-note for ultravisor.

---------- Begin text ----------

Programming Note

The primary use of the DPDES is to provide the means for the hypervisor to save a [sub-]processor's Directed Privileged Doorbell exception state when the set of programs running on the [sub-]processor is swapped out or moved from one [sub-]processor to another. Since there is no such need for a similar function for the hypervisor or ultravisor, there is no similar register for the hypervisor or ultravisor. Privileged programs are able to read the DPDES in order to poll for Directed Privileged Doorbell exceptions when the corresponding interrupt is disabled (MSR_{EE}=1).

---------- End text ----------

Section 10.4 Processor Control Instructions

Add msgsndu and msgclru to the introduction.

---------- Begin text ----------

msgsndu, msgsnd, msgsndp, msgclru, msgclr, and ***msgclrp*** instructions are provided for sending and clearing messages. ***msgsync*** is provided to enable the thread that is target of a ***msgsndu*** or ***msgsnd*** instruction to ensure that stores performed by the message-sending thread before it executed ***msgsndu*** or ***msgsnd*** have been performed with respect to the target thread. ***msgsndp*** and ***msgclrp*** are privileged instructions; ***msgsnd***, ***msgclr***, and ***msgsync*** are hypervisor privileged instructions; ***msgsndu*** and ***msgclru*** are ultravisor privileged instructions.

---------- End text ----------

Add msgsndu and msgclru instructions for interprocessor interrupt / system synchronization purposes as the first two instructions in the section.

---------- Begin text ----------

Message Send Ultravisor *X-form*

msgsndu RB

31	///	///	RB	78	/
0	6	11	16	21	31

```
msgtype ← GPR(RB)32:36
payload ← GPR(RB)37:63
if (msgtype = 0x05) then
   send_msg(msgtype, payload)
```

msgsndu sends a message to other threads in the system. The message type and destination thread(s) are specified in RB.

RB

<-Message Payload->

/ / /	TYPE	B	/ / /	PROCIDTAG
0	32	37	39	44 63

Figure 12. RB Contents for *msgsndu*

The contents of RB are defined below. Bits 37:63 are referred to as the message payload.

Field	Description
0:31	Reserved
32:36	**Type** If Type=0x05, then a Directed Ultravisor Doorbell message is to be sent to the thread(s) specified in the Message Payload field. All other values of the Type field are reserved; if the instruction is executed with this field set to a reserved value, the instruction is treated as a no-op.
37:38	**Broadcast** (B) 00 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. 01 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. 10 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.

11 Reserved

39:43 Reserved

44:63 **PROCIDTAG**

This field indicates the recipient thread(s) as specified in the B field. If this field set to a value that is not the same as bits $PIR_{44:63}$ of any thread in the system, then the instruction behaves as if it were a no-op.

The actions taken on receipt of a message are defined in Section 10.2.

This instruction is ultravisor privileged.

Special Registers Altered:

None

Programming Note

If ***msgsndu*** is used to notify the receiver that updates have been made to storage, a ***sync*** should be placed between the stores and the ***msgsndu***. See Section 5.9.2.

Message Clear Ultravisor *X-form*

msgclru RB

31	///	///	RB	110	/
0	6	11	16	21	31

```
t ← hypervisor thread number of executing thread
if (msgtype = 0x05) then
   clear any Directed Ultravisor Doorbell exception
   for thread t
```

msgclru clears a message previously accepted by the thread executing the ***msgclru***.

Let msgtype be $(RB)_{32:36}$, and let t be the hypervisor thread number of the thread executing the ***msgclru*** instruction.

If msgtype = 0x05, then clear any Directed Ultravisor Doorbell exception that exists on thread t; otherwise, this instruction is treated as a no-op.

This instruction is ultravisor privileged.

Special Registers Altered:

None

Programming Note

msgclru is typically issued only when MSR_{EE}=0. If ***msgclru*** is executed when MSR_{EE}=1 when a Directed Ultravisor Doorbell interrupt is about to occur, the corresponding interrupt may or may not occur.

---------- End text ----------

Make minor editorial fixes to the msgsnd description.

---------- Begin text ----------

```
msgtype ← GPR(RB)32:36
payload ← GPR(RB)37:63
if (msgtype = 0x05) then
   send_msg(msgtype, payload)
```

...

Programming Note

If ***msgsnd*** is used to notify the receiver that updates have been made to storage, a ***sync*** should be placed between the stores and the ***msgsnd***. See Section 5.9.2.

---------- End text ----------

Make minor editorial fixes to the msgclr description.

---------- Begin text ----------

```
t ← hypervisor thread number of executing thread
if (msgtype = 0x05) then
   clear any Directed Hypervisor Doorbell exception
   for thread t
```

---------- End text ----------

Fix up msgsync to include msgsndu.

---------- Begin text ----------

Message Synchronize *X-form*

msgsync

31	///	///	///	886	/
0	6	11	16	21	31

In conjunction with the *Synchronize* and ***msgsndu*** or ***msgsnd*** instructions, the ***msgsync*** instruction provides an ordering function for stores that have been performed with respect to the thread executing the *Synchronize* and ***msgsndu*** or ***msgsnd*** instructions, relative to data accesses by other threads that are performed after a Directed Ultravisor Doorbell or Directed Hypervisor Doorbell interrupt has occurred, as described in the *Synchronize* instruction description on p. 1067.

This instruction is hypervisor privileged.

Special Registers Altered:

None

Programming Note

When used in conjunction with ***msgsndu*** or ***msgsnd***, *Synchronize* with L = 0 or 2 is executed on the thread that will execute the ***msgsndu*** or ***msgsnd***, and ***msgsync*** is executed on another thread -- typically the thread that is the target of the ***msgsndu*** or ***msgsnd***, but possibly any other thread (partly because the software that services the Directed Ultravisor Doorbell or Directed Hypervisor Doorbell interrupt may ultimately run on a thread other than that which received the exception). The *Synchronize* precedes the ***msgsndu*** or ***msgsnd***; the ***msgsync*** is executed after the Directed Ultravisor Doorbell or Directed Hypervisor Doorbell interrupt occurs, and precedes all instructions that need to "see" the values stored by the stores that are in set A of the memory barrier created by the *Synchronize*; see Section 5.9.2, “Synchronize Instruction”.

---------- End text ----------

Chapter 11. Synchronization Requirements for Context Alterations

Add **urfid**, urmor, and SMFCTRL to the tables. Remove reference to note 11 from both LPCR entries.

---------- Begin text ----------

Instruction or Event	Required Before	Required After	Notes
event-based branch and ***rfebb***	none	none	21
interrupt	none	none	
rfid	none	none	
hrfid	none	none	
urfid	none	none	
rfscv	none	none	
sc	none	none	
scv	none	none	
Trap	none	none	
mtspr (AMR)	CSI	CSI	13
mtspr (PIDR)	CSI	CSI	6
mtspr (DAWRn)	CSI	CSI	
mtspr (DAWRXn)	CSI	CSI	
mtspr (HRMOR)	CSI	CSI	11,17
mtspr (URMOR)	CSI	CSI	11,17
mtspr (LPCR)	CSI	CSI	14
mtspr (PTCR)	***ptesync***	CSI	3
mtspr (SMFCTRL)	CSI	CSI	
mtmsrd (SF)	none	none	
mtmsrd (TS)	none	none	
mtmsrd (TM)	none	none	
mtmsr[***d***] (PR)	none	none	
mtmsr[***d***] (DR)	none	none	
mtspr (PIDR)	CSI	CSI	6
slbie	CSI	CSI	4
slbieg	CSI	CSI	4,6
slbia	CSI	CSI	4
slbmte	CSI	CSI	4,10
tlbie	CSI	CSI	4,6
tlbiel	CSI	***ptesync***	4
Store(PTE)	none	{***ptesync***, CSI}	5,6
Store(STE)	none	{***ptesync***, CSI}	5,6
Store(PRTE)	none	{***ptesync***, CSI}	5,6
Store(PATE)	none	{***ptesync***, CSI}	5,6
transaction failure and all TM instructions except ***tcheck***	none	none	21

Table 2: Synchronization requirements for data access

Instruction or Event	Required Before	Required After	Notes
event-based branch and ***rfebb***	none	none	21
interrupt	none	none	
rfid	none	none	
hrfid	none	none	
urfid	none	none	
rfscv	none	none	
sc	none	none	
scv	none	none	
Trap	none	none	
mtmsrd (SF)	none	none	7
mtmsrd (TS)	none	none	
mtmsrd (TM)	none	none	
mtmsr[***d***] (EE)	none	none	1
mtmsr[***d***] (PR)	none	none	8
mtmsr[***d***] (FP)	none	none	
mtmsr[***d***](FE0,FE1)	none	none	
mtmsr[***d***] (SE, BE)	none	none	
mtmsr[***d***] (IR)	none	none	8
mtmsr[***d***] (RI)	none	none	
mtspr (DEC)	none	none	9
mtspr (PIDR)	CSI	CSI	6
mtspr (IAMR)	none	CSI	
mtspr (TFHAR)	none	none	
mtspr (TEXASR)	none	none	
mtspr (CTRL)	none	none	
mtspr (FSCR)	none	CSI	
mtspr (DPDES)	none	CSI	17
mtspr (CIABR)	none	CSI	
mtspr (HFSCR)	none	CSI	
mtspr (HDEC)	none	none	9
mtspr (HRMOR)	none	CSI	8, 11,17
mtspr (URMOR)	none	CSI	8, 11,17
mtspr (LPCR)	none	CSI	12
mtspr (LPIDR)	CSI	CSI	6,14,17
mtspr (PCR)	none	CSI	17
mtspr (PTCR)	***ptesync***	CSI	3,17
mtspr (SMFCTRL)	none	CSI	
mtspr (Perf. Mon.)	none	CSI	15,18
mtspr (BESCR)	none	CSI	16,18
slbie	none	CSI	4
slbieg	none	CSI	4,6
slbia	none	CSI	4
slbmte	none	CSI	4,8,10
tlbie	none	CSI	4,6
tlbiel	none	CSI	4
Store(PTE)	none	{***ptesync***, CSI}	5,6,8
Store(STE)	none	{***ptesync***, CSI}	5,6,8
Store(PRTE)	none	{***ptesync***, CSI}	5,6,8
Store(PATE)	none	{***ptesync***, CSI}	5,6,8
transaction failure and all TM instructions except ***tcheck***	none	none	21

Table 3: Synchronization requirements for instruction fetch and/or execution

---------- End text ----------

Add URMOR to the text of footnote 11 and remove LPCR[VC].

---------- Begin text ----------

When the URMOR or the HRMOR is modified, software must invalidate all implementation-specific lookaside information used in address translation that depends on the old contents of the register or field (i.e., the contents immediately before the modification). The ***slbia*** instruction can be used to invalidate all such implementation-specific lookaside information.

---------- End text ----------

Book Appendices:

Appendix G. Opcode Maps

Add the following entry in Table 9 (Right), column '10010', row '01001:

306
urfid
S XL

Add msgsndu and msgclru.

Appendices H, I, J. Power ISA AS Instruction Set

Add **urfid, msgsndu, and msgclru** to the instruction listings, as appropriate. Add ultravisor priority to the key.

---------- End RFC ----------