POWER9 Processor User's Manual

OpenPOWER

Version 2.1
10 October 2019

Printed in the United States of America October 2019

IBM Systems and Technology Group
2070 Route 52, Bldg. 330
Hopewell Junction, NY 12533-6351

The IBM home page can be found at **ibm.com®**.

Version 2.1
10 October 2019

Contents

List of Figures

List of Tables

Revision Log

Each release of this document supersedes all previously released versions. The revision log lists all significant changes made to the document since its initial release. In the rest of the document, change bars in the margin indicate that the adjacent text was modified from the previous release of this document.

Revision Date	Description
10 October 2019	Version 2.1. • Revised *Related Documents* on page 28. • Revised *Section 4 Power Architecture Compliance* on page 51. • Revised *Section 4.2 Fixed-Point Invalid Forms and Undefined Conditions* on page 58. • Revised *Section 4.3.7 Floating-Point Invalid Forms and Undefined Conditions* on page 62. • Revised *Section 4.6.2.4 Vector Category Prefetch Instructions (**dss, dst, and dstst)*** on page 65. • Revised *Section 4.6.2.5 Data Cache Block Touch Instructions (**dcbt and dcbtst)*** on page 66. • Added *Section 4.6.2.6 Data Cache Block Touch Instructions (**dcbt and dcbtst) - Single Cache Line (TH = '00000')*** on page 66. • Added *Section 4.6.2.7 Data Cache Block Touch - Invalid TH Forms (TH = '00001' through TH = '00111')* on page 66. • Revised *Section 4.6.2.8 Data Cache Block Touch Data Stream (TH = '01000')* on page 67. • Added *Section 4.6.2.9 Data Cache Block Touch Data Stream Descriptor (TH = '01010')* on page 67. • Added *Section 4.6.2.10 Data Cache Block Touch Data Stream Stride Descriptor (TH = '01011')* on page 67. • Revised *Section 4.6.2.11 Data Cache Block Touch - Transient (TH = '10000')* on page 67. • Revised *Section 4.6.2.12 Data Cache Block Touch - No Access Needed Anymore (TH = '10001')* on page 68. • Added *Section 4.6.4.1 TDOOMED* on page 71. • Revised *Section 4.7.3.4 Move To/From Special Purpose Register Instructions* on page 82. • Revised *Table 4-8 SPR Table* on page 83. • Revised *Table 4-16 Address Bit Range Checking by Hardware* on page 103. • Revised *Section 4.10.9.1 D-ERAT I and G Bit Setting* on page 115 (removed additional condition descriptions for *Table 4-28 D-ERAT I and G Bit Setting* on page 115). • Added a bullet describing secure memory facility support to *Section 10.1 EMC Major Features* on page 184. • Changed bullet to 24 cores x 4 threads in *Section 17.3.3 P3 Presentation Controller (P3PC)* on page 262. • Revised the introduction in *Section 24 Specific Security Features* on page 321. • Added *Section 24.3 Secure Memory Facility* on page 326. • Added **mfspr_smfctrl/mtspr_smfctrl**, **mfspr_urmor/mtspr_urmor**, **mfspr_usprg0/mtspr_usprg0**, **mfspr_usprg1/mtspr_usprg1**, **mfspr_usrr0/mtspr_usrr0**, and **mfspr_usrr1/mtspr_usrr1** to *Table A-1. Instruction Properties* on page 375. • Revised the *Glossary* on page 495.
9 April 2018	Version 2.0.
29 September 2016	Version 1.0.

About this Document

This user's manual describes the IBM® POWER9™ processor and provides information about the registers, facilities, initialization, and use of the POWER9 processor.

This document provides information about the POWER9 processor that is visible from a programming model point of view, and is intended to be a companion to the baseline architecture documentation (see *Related Documents* on page 28). While there are some programming model considerations associated with chips and subsystems outside of the Central Electronics Complex (CEC), this document focuses primarily on the microprocessor core and the storage subsystem. For information about other chips that might appear in POWER9 systems, see the functional specifications for these individual chips.

Who Should Read this Document

This manual is intended for system software and hardware developers and application programmers who want to develop products for the POWER9 processor. It is assumed that the reader understands operating systems, microprocessor system design, basic principles of reduced instruction set computer (RISC) processing, and details of the Power ISA.

Conventions Used in This Document

This section explains numbers, bit fields, instructions, and signals that are in this document.

Representation of Numbers

Numbers are generally shown in decimal format, unless designated as follows:

- Hexadecimal values are preceded by an "x" and enclosed in single quotation marks.
  For example: x'0A00'.
- Binary values in sentences are shown in single quotation marks.
  For example: '1010'.

Note: A bit value that is immaterial, which is called a "don't care" bit, is represented by an "X."

Bit Significance

In the POWER9 documentation, the smallest bit number represents the most significant bit of a field, and the largest bit number represents the least significant bit of a field.

Other Conventions

Instruction mnemonics are shown in lower-case, bold text. For example: **tlbie**. I/O signal names are shown in upper case.

Related Documents

The following documents can be helpful when reading this specification. Contact your IBM representative to obtain any documents that are not available through the IBM Portal for OpenPOWER or the OpenPOWER foundation.

Power ISA User Instruction Set Architecture - Book I (Version 3.0B)

Power ISA Virtual Environment Architecture - Book II (Version 3.0B)

Power ISA Operating Environment Architecture - Book III (Version 3.0B)

POWER9 Processor Programming Model Bulletin

Linux on Power Architecture Platform Reference

PCI Express Base Specification, Revision 4.0

IBM EnergyScale for POWER8 Processor-Based Systems

Manual for Using WBEM CLI to Fetch Flexible Service Processor CIM Data

POWER9 Processor Programming Guide for the 25G Link with NVLink 2.0 Compliant Devices

Coherent Accelerator Interface Architecture (CAIA)

1. POWER9 Processor Overview

The POWER9 processor is a superscalar symmetric multiprocessor designed for use in servers and large-cluster systems. It uses CMOS 14 nm technology with 17 metal layers.

The POWER9 processor supports direct-attach memory. It supports a maximum symmetric multiprocessing (SMP) size of two sockets and is targeted for scale-out workloads. The POWER9 processor offers superior cost and performance. The target market segments are:

- Technical Computing: The POWER9 processor provides superior floating-point performance and high-memory bandwidth to address this market segment. It also supports off-chip floating-point acceleration.
- Cloud Operating Environments: The POWER9 processor enables efficient cloud management software, enforces service-level agreements, and provide facilities for charge-back accounting based on resources consumed.
- Big Data Analytics: The POWER9 processor with CAPI attach, large caches, and on chip accelerators provides a robust platform for analytics and big-data applications.
- Enterprise: Robust cache/memory for in-memory database applications.

From a logical perspective, the POWER9 processor consists of six main components:

- POWER9 processing core including the L1 cache
- L2/L3 caches and noncacheable unit (NCU)
- Processor bus fabric interconnect
- Memory subsystem
- PCIe I/O subsystem
- Accelerator subsystem

1.1 General Features

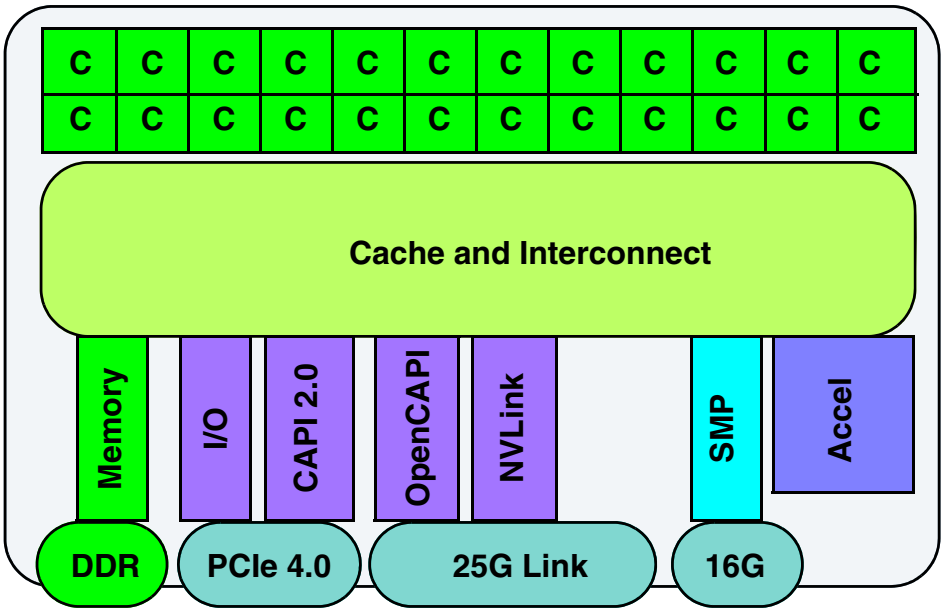
The POWER9 processor can have up to 24 cores enabled on a single chip and is offered with a direct-attached memory for scale-out computers. Each core has four threads that use simultaneous multithreading (SMT).

The POWER9 processor supports the following architectural features:

- Power ISA Architecture (Books I, II, and III), version 3.0B
- Linux on Power Architecture Platform Requirements
- I/O Design Architecture v2 (IODA2) Specification, Version 2.4+
- IEEE P754-2008 floating-point compliant
- Big-endian, little-endian, strong-ordering support extension
- 56-bit real address, 68-bit virtual address

Figure 1-1 provides a block diagram of the POWER9 processor.

Figure 1-1. POWER9 Processor Block Diagram



The following features describe the main components of the 24-core POWER9 processor chip:

- POWER9 core and cache
  - Up to 24 processor cores
  - Four-slice design plus
    - Branch unit
    - Decimal floating-point unit
    - Crypto unit
  - Each slice can perform one FX or VSX operation per cycle and one LS operation
  - Four SMT, O-o-O
  - 32 GPR, 32 FPR, and 64 VSR registers per thread
  - 20-deep primary and 96-deep secondary history buffer per slice
  -
- Core pairs share
  - 32 KB per core (not shared) instruction cache (I-cache)
  - 32 KB per core (not shared) data cache (D-cache)
  - 512 KB private L2 cache
  - 10 MB eDRAM L3 cache

- DDR4 memory controllers
  - Eight DDR4 ports, 1.2 V, two DIMMs per port
  - Supports ×4, ×8, 4 - 16 Gb, R/LR, 3D devices
  - DDR4 support: maximum 2667 MHz, one DIMM per channel
  - DDR4 support: maximum 2400 MHz, two DIMMs per channel
  - High-throughput atomic memory operations
  - 133+ GBps streaming bandwidth at 2667 MHz
  - 128-byte line with 64-byte sectoring
- POWER9 SMP on-chip interconnect
  - 1600 - 2400 MHz frequency
  - Eight 32-byte data buses
  - Four address snoop buses
  - 12 or 24 core ramps
  - Fifteen nest ramps
- POWER9 SMP off-chip interconnect
  - Two 30-bit + 2 spare electrical X buses at 16 GHz
  - Maximum two socket SMP
- PCIe GEN4 support
  - 16 GHz differential PCIe Generation 4 buses: 48 lanes grouped in three sets of 16 lanes
    - 1 × 16 + 1× 16 that bifurcates to 2 × 8
    - 1 × 8 + 1 × 8 that bifurcates to 2 × 4
  - Six separate PCI host bridges (PHB)
  - P3Virtualization controller
- NVIDIA® NVLink™ protocol over the 25G Link interface:
  - Six bricks
  - Eight lanes per brick
  - 25 Gbps transfer rate per lane
  - Coherent memory operations
  - GPU direct
  - GPU-to-GPU connections
  - Address translation services (ATS)
- Power management support
  - Core/L2/L3 instant on/off
  - Halt state support
  - Controlled by 17 on-chip programmable PPE engines
  - Hypervisor-directed power change requests using a Pstate mechanism
  - Dynamic lane width reduction (SMP interconnect, PCI)

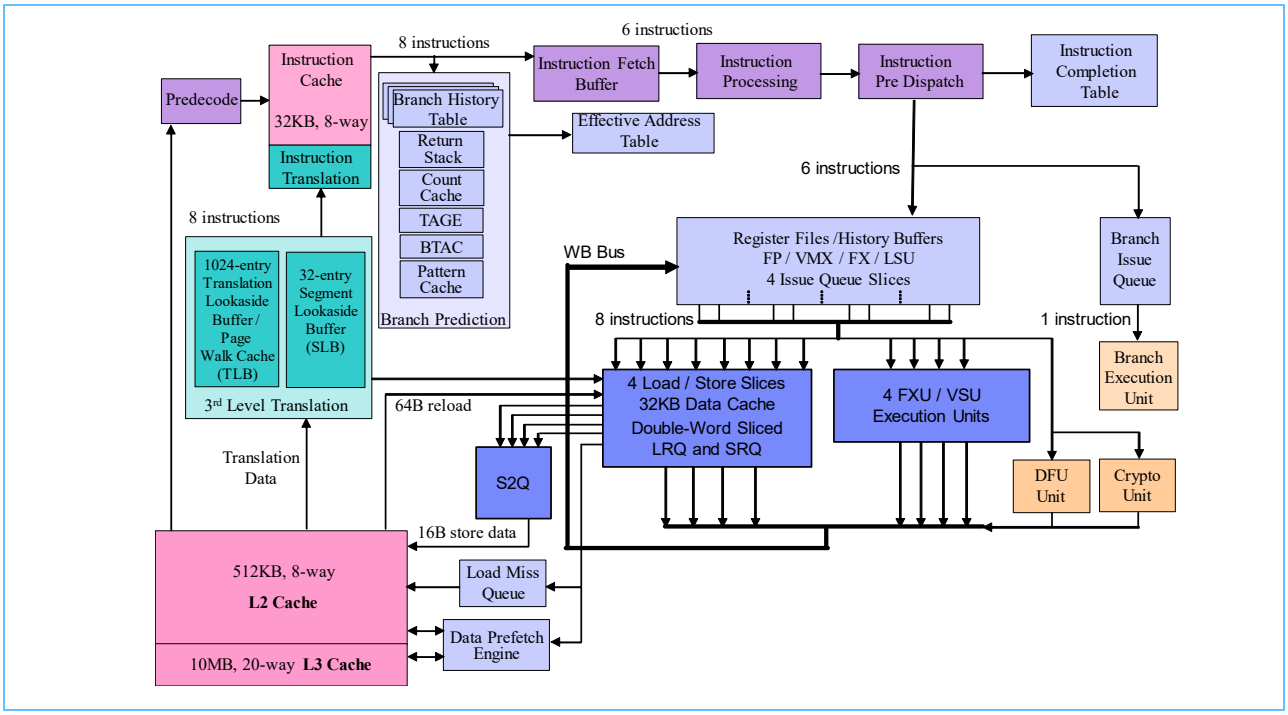
  - Sensors
    - Digital thermal sensor (DTS2) ±5°C
    - On-chip analog thermal diode ±1 - 2°C
    - Voltage drop monitor
    - Dedicated performance, microarchitecture, and event counters
  - On-chip controller (OCC)
    - On-chip PowerPC 405 for thermal management control
    - On-chiplet hardware assist (automated core chiplet management)
    - On-chip power management controls automated communications to the voltage regulation modules (VRMs) and voltage and frequency sequencers for automated Pstate and idle state support
  - Actuators
    - Per-chiplet frequency control through the DPLL
    - Architected idle states: nap, sleep, and winkle; each with increasing power savings capability (and latency)
    - SPR power management control registers (PMCR, PMICR, PMSR) for hypervisor support
  - Memory/DIMM throttling for memory subsystem power and thermal management
- Clocking
  - Reference clocks
    - 133 MHz core/nest
    - 100 MHz PCI
    - 156 MHz optics
    - 16 MHz TOD
  - 26 PLLs and DDR DLLs
    - Six core quads, six X bus, six PCI, three reference clock filters, two optics, two DDR, one nest
- Accelerators
  - CAPI attachment options
    - Legacy POWER8® CAPI adapter support
    - 2 × 16 PCIe Gen4
  - Single GZIP engine
  - Two 842 compression engines
  - Three AES/SHA engines
  - Atomic memory operations (AMO)
  - Nest MMU to enable user access to all accelerators
- Pervasive interface
  - Two FSI slaves
  - 2 × 8 FSI master
  - 2 × I^2C SEEPROM
  - Quad SPI
  - Low-pin count (LPC) connection to baseboard management control (BMC)
  - Processor serial interface (PSI) for connection to flexible service processor (FSP)
- Cloud management quality of service (QoS) support

- Features
  - On-chip accelerators
    - CAPI allows an FPGA or ASIC to connect coherently to the POWER9 processor SMP interconnect via the PCIe.
    - On-chip: compression, encryption, data move initiated by hypervisor, GZIP engine, nest MMU to enable user access to all accelerators
    - In-core: user invocation encryption (AES, SHA)
    - OpenCAPI: industry-standard, high-speed, low-latency acceleration
  - Cloud computing enhancements: page replacement/affinity assist, IPL time reduction, four concurrent LPARs per core
  - Transactional memory
  - Random number generator
  - RAID6 support in VMX
  - Support for industry standard BMC
  - Multi-level TCE support
  - Turbo mode support
  - Details of CAPI can be found in the *Coherent Accelerator Interface Architecture (CAIA)* document and the associated *CAPI User Handbook* document.

2. POWER9 Processor Core

This section provides an overview of the POWER9 microprocessor core, including key design fundamentals, an overview of the master pipeline operation, and a detailed summary of key design features.

Figure 2-1. POWER9 Processor Core



2.1 Key Design Fundamentals

This section describes the key design fundamentals of the POWER9 processor core.

2.1.1 64-bit Implementation of the Power ISA (Version 3.0)

- Compatibility for all Power ISA application-level code (problem state).
  - Architecturally supports POWER8 mode.
  - Supports partition mobility.
- Supports IEEE standard P754.
- Linux support.
- AIX support with backward operating-system capability. Backward compatible up to AIX 5.3, with PCR-based compatibility mode.

2.1.2 Layered Implementation Strategy for High-Frequency Operation

- Reduced pipelined design.
  - 11 stages from I-cache access to writeback for most fixed-point register-to-register operations.
  - 13 stages for most load/store operations (assuming an L1 D-cache hit) from I-cache to writeback.
  - 17 stages for most floating-point operations from I-cache access to writeback.
- Dynamic instruction cracking[1] for some instructions allows for simpler inner-core dataflow.
  - Dedicated dataflow for cracking one instruction into two or more internal operations.
  - Microcoded templates for longer emulation sequences.

2.1.3 Speculative Superscalar Inner Core Organization

- Multi-threaded core design.
  - Single thread (ST), 2-way multi-thread (SMT2), and 4-way multi-thread (SMT4).
  - Four logical partitions (LPARs) supported at a time.
- Aggressive branch prediction.
  - Prediction for up to eight branches per cycle.
  - Support for up to 40 predicted taken branches in-flight ST mode. Twenty predicted taken branches per thread in SMT2 mode and 10 predicted taken branches per thread in SMT4 mode.
  - Prediction support for branch direction and branch target addresses.
- In-order dispatch of up to six internal operations (iops) into five distributed issue queues per cycle.
  - Up to two branches dispatched per cycle.
  - Up to six non-branch instructions dispatched per cycle.
- Out-of-order issue of up to nine operations.
  - Four load or store agen operations.
  - Four 64-bit execution/computational operations, 128-bit operations are issued as a pair of 64-bit issues.
  - One branch operation.
- Register renaming on GPRs, FPRs, CR fields, XER (parts), FPSCR, VSCR, Link, TAR, and Count.
- Eleven execution units.
  - Four symmetric load/store units (LSU).
  - Four symmetric 64-bit VMX execution units capable of executing fixed point ALU, simple FX, complex FX, permute, 128-bit fixed-point, single, double-precision, floating-point operations. Two execution units are tied together to perform 128-bit execution.
    - Four floating-point units (FPU). Each FPU supports a double-precision operation or up to two single-precision operations each for SIMD and also supports fixed-point multiply and complex FX operations.
    - For each symmetric unit, only one operation per cycle can be issued.
  - One decimal floating-point and quad-precision floating-point unit (DFU).

1. Process by which some complex instructions are broken into multiple simpler, more RISC-like instructions.

  - One crypto unit.
  - One branch execution unit (BR).
- Large number of instructions in flight.
  - 96 instructions deep, instruction-fetch buffer, split equally across the threads in SMT2 and SMT4 mode.
  - Up to 24 instructions in four dispatch pipe stages.
  - Up to 256 instructions from dispatch through instruction completion.
  - Up to 64 stores queued in the SRQ (available for forwarding), shared by the available threads and buffered en-route to the L2 cache through a 16-entry S2Q.
- Fast, selective flush of incorrect speculative instructions and results.

2.1.4 Specific Focus on Storage Latency Management

- Out-of-order and speculative issue of load operations.
- Support for up to eight outstanding L1 cache-line misses with critical data forwarding; critical sector first.
- Hardware-initiated instruction prefetching.
- Hardware-initiated or software-initiated data-stream prefetching. Support for up to eight active streams.

2.2 Pipeline Structure

The pipeline structure for the microprocessor can be subdivided into a master pipeline and several different execution unit pipelines. The master pipeline presents speculative in-order instructions to the mapping, sequencing, and dispatch functions, and ensures an orderly completion of the real execution path (throwing away any other potential speculative results associated with mispredicted paths). The execution unit pipelines allow out-of-order issuing of both speculative and non-speculative operations. The execution unit pipelines progress independently from the master pipeline and from one another.

Figure 2-2. Pipeline Structure

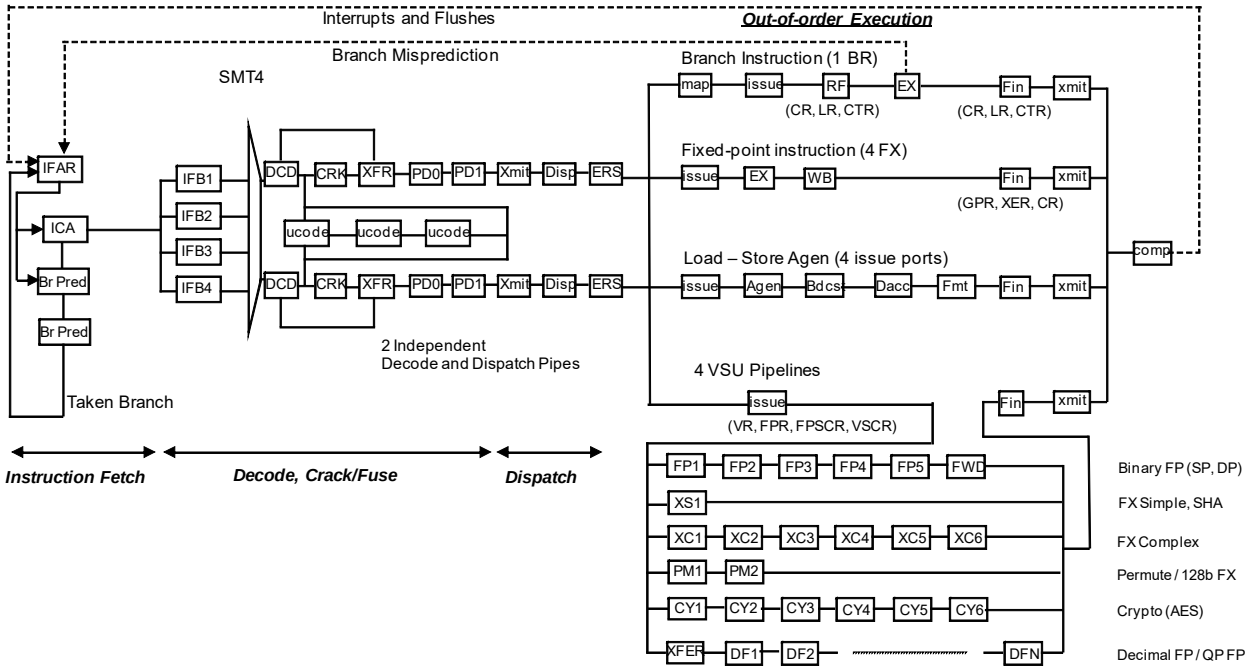


Figure 2-2 illustrates these pipelines, where each box represents a pipeline stage. Definitions for *Figure 2-2* are as follows:

CA/fmt	Data cache access and data formatting	GCT	Global completion table
comp	Group completion	ICA	Instruction cache access
CRK	Crack/instruction fuse	IFAR	Instruction Fetch Address Register
DCD	Decode	ifbx	Instruction fetch buffer latches
DISP	Dispatch	pdx	Pre-dispatch
DSP	Group dispatch	RF	Register file
EA/CA	Effective address generation and data cache decode	ucode	Microcode
ERS	Issue queue	WB	Writeback to the register file
EX	Execution	WRT	Format and write into the GCT
FP1	Floating-point alignment and multiply'	XFR	Transfer
FP2	Multiply	xmit	Finish and transmit
FP3	Add	Xmit	Instruction transfer to dispatch
FP4	Normalize result		
FP5	Round result and re-drive		

In addition, for VMX operations

VF1 - VF6	Represent the pipeline stages for the 4-way SIMD single-precision pipeline stages
XS1	Represents the simple FX operation stage
XC1 - XC6	Represent the complex FX operation stages
CY1 - CY6	Represent the six Crypto execution cycles
PM1 - PM2	Represent the permute stages along with the 128-bit fixed-point operations

The processor core is divided into following six units:

IFU	Instruction fetch and decode unit
ISU	Instruction dispatch and issue unit
LSU	Load/store unit
VSU	Vector and scalar unit (consists of fixed-point, VMX, binary floating-point, crypto, and VSX)
DFU	Decimal floating-point and quad precision floating-point unit
PC	Pervasive unit

2.3 Detailed Features of the Microprocessor Core

2.3.1 Instruction Fetching and Branch Prediction

- 32 KB, 8-way set associative I-cache
  - 128-byte lines (broken into four 32-byte sectors).
  - Dedicated 64-byte interface from the L2 cache that can supply 64-bytes in every processor clock. The I-cache takes in only 32 bytes and buffers the other half.
  - Critical-sector-first reload policy.
  - Effective-address index, real-address tags.
  - Banked I-cache, supports one read and one write per cycle when there is no bank conflict.
  - Predecode bits to aid in fast decoding and group formation.
  - Parity protected; force invalidate and reload on parity error.
- 64-entry effective-to-real address (ERAT) translation cache, fully associative
  - Each entry can translate 4 KB, 64 KB, 2 MB, and 16 MB pages. For MSR[IR] = '1' and nonhypervisor real mode accesses, 1 GB and 16 GB pages take multiple 16 MB entries.
  - In hypervisor real mode, entries are installed as 2 MB pages.
  - In SMT mode, each entry is tagged to indicate invalid or valid for thread 0 - 3.
- Fetch one quadword aligned block of eight instructions per cycle
  - In ST mode, instructions are fetched from the thread in every cycle.

  - In SMT mode, instructions are fetched from a given thread based on thread priority. If the threads are of equal priority, each thread gets approximately an equal number of fetch cycles, while optimizing the core throughput.

- Branch prediction
  - Scan all eight fetched instructions for branches in each cycle.
  - Predict up to eight branches per cycle.
  - Four table first-level prediction structure: global/local/global selector/local selector global (8K entries $\times$ 2-bit) local (8K entries $\times$ 2-bit), global selector (8K entries $\times$ 2-bit), and local selector (8K entries $\times$ 2-bit). Backed up by a TAGE predictor.
    - In SMT modes, thread priority is factored in to determine which thread to fetch from to improve overall fetch throughput.

2.3.2 Instruction Decode and Preprocessing

- One cycle pipeline to preprocess instructions
  - Up to six instructions are decoded concurrently.
  - Dedicated dataflow for cracking one instruction into two or three operations. The rest of the cracked instructions use the ucode expansion templates.
  - The expansion templates are used for longer emulation sequences of internal operations.
  - All expanded instructions delay the pipe by two cycles.
- Logically, there is one instruction fetch buffer (IFB) per thread (sizes differ based on the ST, SMT2, SMT4 modes). Each IFB entry has one instruction. There are 96 entries in an IFB per thread in ST mode, 48 in SMT2 mode, and 24 in SMT4 mode.
- Up to eight instructions can be placed in the IFB in a cycle.
- Up to six instructions can be taken out from the IFB in a cycle (ST or SMT mode).
- Instructions taken out for group formation and decode are from up to two threads; one thread in ST mode and two threads in SMT2 and SMT4 modes.
- The microcode patch facility allows most instructions to trap to software for fix-up or emulation. There are six full Instruction Mask Registers (IMRs) per core.

2.3.3 Instruction Dispatch, Sequencing, and Completion Control

- Three dispatch pipeline cycles hold up to 24 instructions when the ICT is full.
- Inter-instruction dependence generation for RAW and WAW dependences.
- 256-entry instruction completion table (ICT).
  - Each entry is assigned to a particular thread at instruction dispatch.
  - Tracks internal operations from dispatch-to-instruction completion for up to 256 operations.
  - Capable of restoring the machine state for any of the instructions in flight.
- Supports precise exceptions (including machine-check exceptions).
- Register renaming resources. The POWER9 core uses a history buffer to allow out-of-order execution. All renamed registers except the Link, TAR, and Count registers employ this mechanism.
  - GPR/FPR/VR history buffer:
    - Each of the four execution slices contain a 20-entry primary and 96-entry secondary history buffer. For each instruction that updates a GPR/FPR or VR, a copy of each architected target register is held to restore on a flush.
    - In SMT2 mode, the four slices and history buffers are shared by the two threads.
    - In SMT4 mode, two threads run on two slices of each cluster (the other two threads run on the remaining two slices). Therefore, two threads each share the 40 primary and 192 secondary available history buffer entries.
    - In transactional memory mode, the history buffer is also used to contain the checkpoint of the transaction.
  - XER, CR, FPSCR history buffer: 12-entry primary and 12-entry secondary per slice shared amongst all resources and on the same thread basis as the GPRs.
    - XER is mapped to six fields: ov, ca/oc, fxcc, tgcc, sc, dc/ds
    - FPSCR is mapped to four fields: fr/fi/c, fpcc, exceptions, control
    - CR is mapped to eight subfields
    - VSCR is mapped to two fields: sat, nj
  - Any instruction that sets more than four renamed fields must be cracked.
  - 20-entry mapper for LR/CTR/TAR (1 LR, 1 TAR, and 1 CTR per thread)
- Issue queues
  - There are 13 issue queue slots per slice and they are shared by the threads in the same manner as the history buffers. The issue queues hold all instructions except branches.
    - VMX permute operations and 128-bit store operations take two slices (an even/odd slice pair) to handle a 128-bit operation.
    - The load/store is comprised of four slices. Each slice handles a doubleword dataflow. Doublewords 0, 4, 8, 12 of a line are handled by one slice. This pattern is repeated across the remaining slices in the core. Therefore slice 1 contains doublewords 1, 5, 9, and 13.
    - The load-reorder queue and load-store agen queue are merged on a quadword basis. They handle the requests for two of the load-store unit slices.
  - One 15-entry issue queue for branch instructions.

2.3.4 Fixed-Point Execution Pipelines

- Four fixed-point execution pipelines:
  - All four are capable of basic arithmetic, logical, cr logical, and shifting operations.
  - All four are capable of multiplies, divides, and SPR operations.
- Out-of-order issue with a bias toward oldest operations first.
- Symmetric forwarding between fixed-point and load/store execution pipelines.

2.3.5 Load and Store Execution Pipelines

- Four symmetric load/store execution pipelines, with a 4-cycle, load-to-use latency (3-cycle bubble). There are four addresses generated per core per cycle that are picked up by 1 - 3 (misaligned operations) of the the four LSU slices.
- Out-of-order issue with a bias toward oldest operations first. All stores are issued twice: an address generation operation (LD/ST) and a data steering operation (FX/FP/VSX/VMX).
- 32 KB, 8-way set associative, banked D-cache.
  - Supports four reads and one write every cycle, when there is no bank conflict between a write and a read. A given bank can support either one read or one write in a given cycle.
  - Four cycle load-use penalty for loads (3-cycle bubble between a load and a dependent operation).
  - Store-through (to L2 cache) policy; no allocate on store misses.
  - 128-byte cache line with support for 64-byte sectors.
  - Pseudo-LRU replacement policy.
  - EA-based set predict is used to determine the initial hit information. RA-based directory and ERAT is used to define real hit information. A flush can occur on a set-predict hit, directory miss.
  - A dedicated 64-byte reload interface from the L2 cache can supply 64 bytes in every processor clock.
  - Effective address index, real address tags (hardware fix-up on alias cases). That is, two different EAs that map to the same RA are not allowed to co-exist in the D-cache.
  - Parity-protected via recovery
- 64-entry, fully-associative data effective-to-real address (D-ERAT) translation cache.
  - Each entry translates either 4 KB, 64 KB, 2 MB, or 16 MB pages.
    - 16 GB pages take multiple 16 MB pages.
    - 1 GB pages take multiple 16 MB pages.
    - MSR[DR] = '0' entries are also created in the D-ERAT and shared by all threads.
  - Binary LRU replacement policy.
  - In SMT mode, each entry is tagged by thread ID.
  - Entries are dynamically shared between all threads.
- 32-entry, fully-associative segment lookaside buffer (SLB) per thread for HPT translation. The SLB is not used for radix translation.
  - Each entry can support 256 MB or 1 TB segment sizes.
  - Multiple pages per segment (MPSS) feature is supported: 4 KB, 64 KB, and 16 MB pages.

- 16-entry, store re-order queue per slice (real address based; CAM structure).
  - Therefore, there are 64 total SRQ entries in the slices that can be forwarded out of.
    - SRQ is dynamically shared among the available threads.
    - SRQ entry is allocated at the time of a store agen getting access to the cache and deallocated when the store is written in the cache or sent to the L2 cache (after the completion point).
  - Store addresses and store data is supplied on different cycles.
  - Stores wait in this queue until they are completed; then they write the cache and drain to the L2 cache.
  - Supports store forwarding to include subsequent loads (even if both are speculative). Store forwarding takes two additional cycles compared to a D-cache hit for a load.
  - For each SRQ entry, there is a store data queue (SDQ) entry of 8 bytes.
- There is a 16-entry second queue that buffers completed store data that is sent to the L2 cache. Each of these queues is 16 bytes wide but do not forward data to loads in the pipe.
- 16 bytes of store data can be sent to the L2 cache (and also to the D-cache, on a hit) in every processor cycle.
- Two 10-entry load re-order queues and 28-entry load reorder finish queue (real address based; CAM structure).
  - A total of 76 outstanding loads can be issued.
  - LRQ is dynamically shared among the available threads.
  - Keeps track of out-of-order loads and watches for hazards.
- 8-entry load-miss queue per cluster (real address based).
  - Keeps track of loads that have missed in the L1 D-cache.
  - Allows multiple loads from the same cache line to merge onto a single entry (the two loads can be from different threads).
  - Dynamically shared among the threads in SMT modes.
  - Prefetches to L1 are also tracked using the LMQ.
  - LMQ can merge two load operations from the same sector.
- Two 16-byte load and two 16-byte store operations are supported for VMX and VSX operations per cycle. There is no penalty when the load/store operation is 8-byte or 16-byte aligned.
- True little-endian (LE) mode is supported.

2.3.6 Branch Execution Pipelines

- One branch execution pipeline.
  - Computes actual branch address and branch direction for comparison with prediction.
  - Redirects instruction fetching if either direction or target prediction was incorrect.
  - Assists in training and maintaining the branch history table predictors, the link stack, and the count cache.
- Out-of-order issue with a bias toward oldest operations first.

2.3.7 Unified Second-Level Memory Management (Address Translation)

- 1024-entry, 4-way set-associative TLB per cluster.
  - 4 KB, 64 KB, 2 MB, 16 MB, 1 GB, and 16 GB pages are supported in the TLB.
  - TLB also supports "Virtualized Page Class Key Protection" with 32 keys.
  - Indexed with partially hashed address to improve performance.
  - Hardware-based reload (from the L2 cache interface; no L1 D-cache corruption).
  - Hardware-based atomic and non-atomic update of the R-bit, C-bit, and TS-bit.
  - Parity protected via recovery.
  - In SMT mode, the TLB entries are shared by the four threads as long as the entry belongs to the logical partition running on the core.
  - 12-bit LPAR ID per entry.
- Hit-under-miss is allowed in the TLB.
- Support for four concurrent table walks (without any restriction on thread of D-side or I-side requests).
- 32-entry fully-associative SLB, one per thread.
  - SLB miss results in an interrupt (a software reload of the SLB).
  - SLB can also be loaded by using the software-initiated SLB instructions.
  - SLB supports 256 MB and 1 TB segment sizes.
- A segment with 4 KB base page size is allowed to have mixed pages of sizes 4 KB, 64 KB, and 16 MB pages.
- A segment with 64 KB base page size is allowed to have mixed pages of sizes 64 KB and 16 MB pages.
- Read of invalid SLB entry returns zeros for enhanced security.
- Supports 68-bit virtual address for HPT and 56-bit real address.
- Two outstanding table-walks per cluster and TLB hits-under-miss is allowed.
- There are no restrictions on the thread or D-side or I-side for the concurrency of the two table walks.
- Both software and hardware TLB management is allowed.
- True LRU replacement policy.
- Supports 52-bit guest effective address and 52-bit guest real address (host effective address) for radix. Guest real address bits 0:11 are ignored by the hardware and treated as zeros.

2.3.8 Data Prefetch

- Eight independent data streams capable of striding up or down.
- Prefetches and allocates ahead of demand into the L1 D-cache from the L3 cache.
- Prefetches and allocates ahead of demand into the L3 cache from memory.
- Support for software-initiated stream startup (special variant of the **dcbt** instruction).
- Hardware and software-initiated streams can use eight data streams with sharing in SMT modes.

2.3.9 VSU Execution Pipeline

The vector scalar unit has a merged fixed-point, floating-point, permute and vector dataflows. Up to four instructions can be issued to the execution pipeline in a given cycle. It contains:

- Four fixed-point execution pipelines with 5 stage execution:
  - All fully capable for the full set of floating-point instructions.
  - All data formats are supported.
  - Non-IEEE mode supported, which provides less precise results at lower divide/square root latencies.
- Out-of-order issue with bias toward oldest operations first.
- Symmetric forwarding between fixed-point and load/store execution pipelines.
- VSU unit contains binary floating-point execution unit, SIMD double-precision floating-point (VSX) execution unit, and the VMX execution unit.
- Up to four instructions can be issued to the VSU in a given cycle to the four pipelines.
  - The instruction in the first pipeline can be a simple fixed-point, a complex fixed-point, a 4-way SIMD single-precision FPU operation, a 2-way SIMD double-precision FPU operation (VSX), or a scalar floating-point operation.
  - The four pipes are symmetric.
  - Two pipes (slices) are used to handle 128-bit operations.
  - Out-of-order issue with bias toward oldest operations first.
  - Four load result bus to the VRF, each supports up to 8-byte loads in a cycle.
  - Store data bus from VRF to the SDQ supports four 8-byte stores or two 16-byte stores in a cycle.
- Floating-point execution:
  - Four symmetric floating-point execution pipelines with 6-stage execution:
    - All are capable of the full set of floating-point instructions.
    - All data formats supported in hardware (no floating-point assist interrupts).
    - A new test instruction facilitates execution of multiple concurrent divide or square-root operations.
    - Back-to-back six cycles issue to local and eight cycles to the remote FPU.
    - Non-IEEE mode supported, which provides less precise results at a lower latency for divide and square-root operations.
- VSX execution:
  - Four symmetric SIMD floating-point execution pipelines, with stage execution:
    - Both capable of the full set of VSX instructions (single-precision and double-precision).
    - All data formats supported in hardware (no assist interrupts).
    - A new test instruction facilitates execution of multiple concurrent divide or square-root operations.
    - Back-to-back 5-cycle issue to local and 7-cycle issue to the remote VSX pipe.
- VMX execution:
  - Four execution pipelines within VMX: simple fixed-point, complex fixed-point, permute, and 4-way SIMD single-precision floating-point unit.

- Simple fixed-point operations take two execution cycles.
- Complex fixed-point operations take five execution cycles.
- Permute operations take two execution cycles.
- Vector floating-point operations take five execution cycles.

2.3.10 Decimal Floating-Point Execution Pipeline

- DFP unit can execute 64-bit or 128-bit DFP operations.
- Allows out-of-order issue with bias toward oldest instruction.
- Pipelined execution.
- DFP unit shares VSU pipe 1 issue port with the VSU unit.
- 128-bit DFP instructions can be cracked into 2-way or microcoded internal operations.
- One quad-precision floating-point execution pipeline with 12-stage execution. Two issue pipes are used to handle quad-precision execution.

3. Packages

This chapter outlines the module packages for the POWER9 processor. An X bus is the socket-to-socket SMP interconnect between two POWER9 processors.

3.1 POWER9 Single-Chip Module for Cloud and Data Center

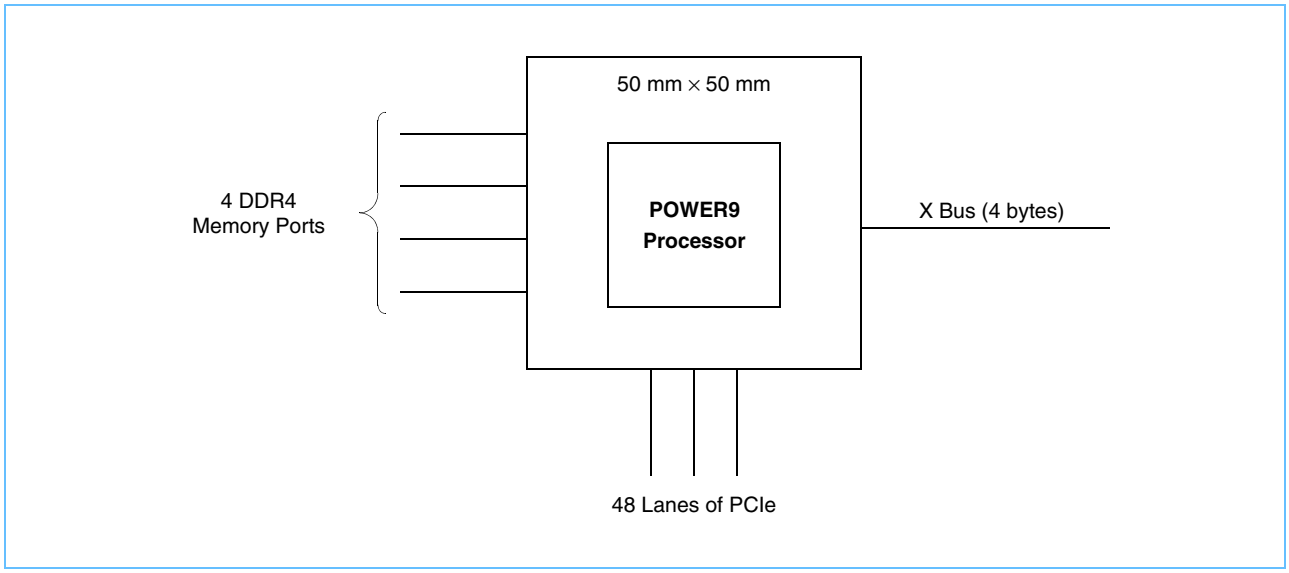
Features:

- Body size: 50 mm × 50 mm
- Interconnect technology: Hybrid LGA socket
- 1.016 mm hexagonal LGA pitch and 2601 pins
- 4-4-4 organic package construction

Buses:

- Four DDR4 ports
- One 30-bit + 2 spare (4-byte) electrical X buses at 16 Gbps
- 48 lanes PCIe Gen4 at 16 Gbps

Figure 3-1. POWER9 Single-Chip Module for Cloud and Data Center



3.2 POWER9 Single-Chip Module for High-Performance Computing and Cloud

Features:

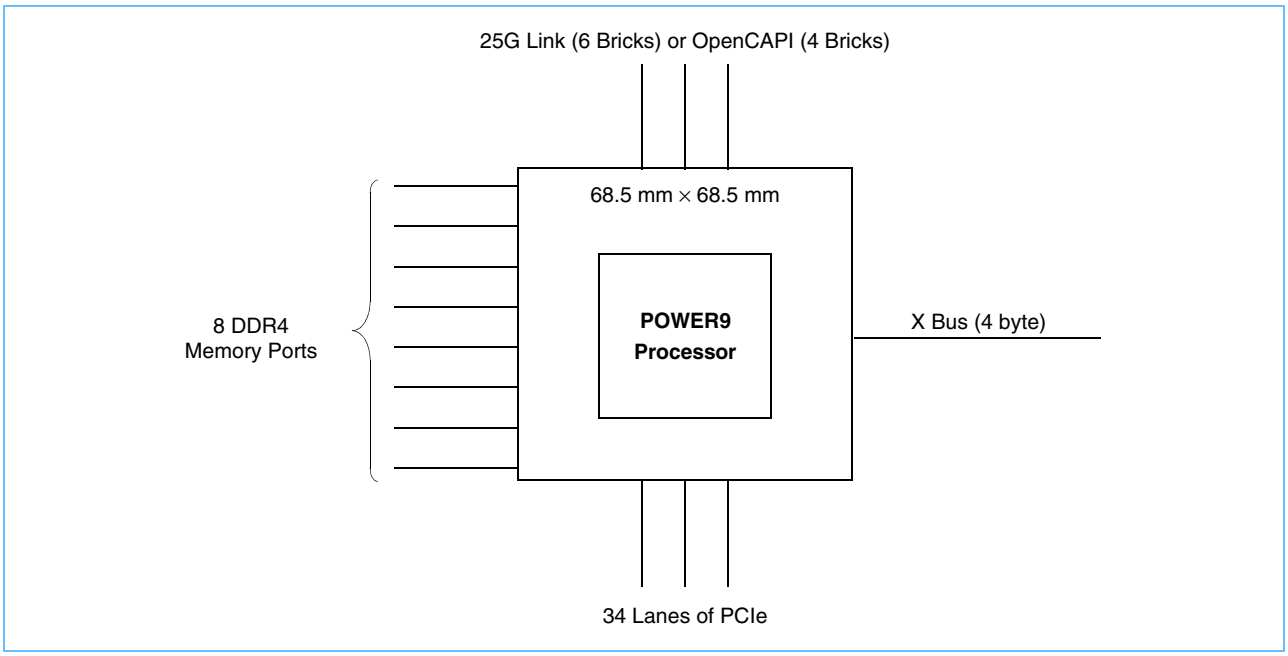
- Body size: 68.5 mm × 68.5 mm
- Interconnect technology: Hybrid LGA socket
- 1.5 mm interstitial LGA pitch with a minimum pitch of 1.06 mm and 3899 pins
- 7-2-7 organic package construction

Buses:

- Eight DDR4 ports
- 25G Link: six bricks at 25 Gbps or
  OpenCAPI: four bricks at 25 Gbps

  Note: The basic building block for a 25G Link is a high-speed, 8-lane, differential, dual simplex bidirectional link. In this document, the term "brick" is equivalent to the term "link."
- One 30-bit + two spare (4-byte) electrical X buses at 16 Gbps
- 34 lanes PCIe Gen4 at 16 Gbps

Figure 3-2. POWER9 Single-Chip Module for HPC/Cloud



3.3 POWER9 Single Chip Module for Commercial Entry

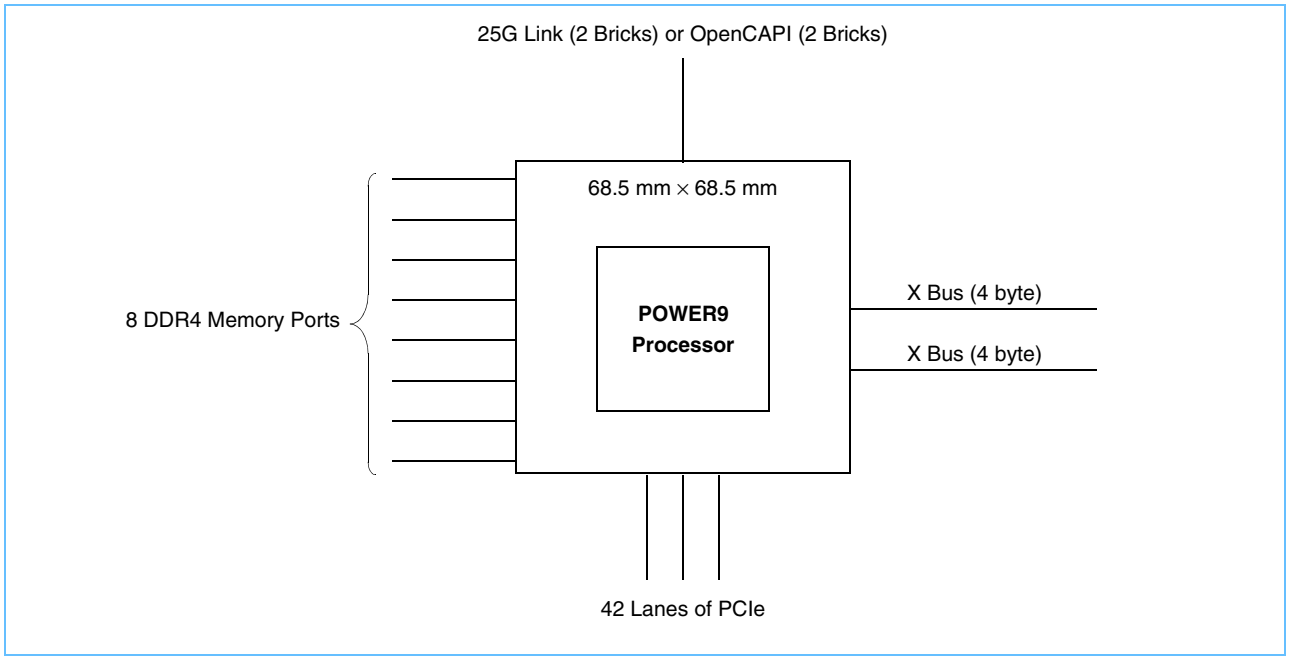
Features:

- Body size: 68.5 mm × 68.5 mm
- Interconnect technology: Hybrid LGA socket
- 1.5 mm interstitial LGA pitch with a minimum pitch of 1.06 mm and 3899 pins
- 7-2-7 organic package construction

Buses:

- Eight DDR4 ports
- 25G Link: two bricks at 25 Gbps or
  OpenCAPI: two bricks at 25 Gbps
- Two 30-bit + two spare (4-byte) electrical X buses at 16 Gbps
- 42 lanes PCIe Gen4 at 16 Gbps

Figure 3-3. POWER9 Single-Chip Module for Commercial Entry



4. Power Architecture Compliance

The following sections are intended to be read with their respective companion documents. Throughout these sections, it is assumed that the reader is familiar with the following architecture documents:

- *Power ISA User Instruction Set Architecture - Book I (version 3.0B)*
- *Power ISA Virtual Instruction Set Architecture - Book II (version 3.0B)*
- *Power ISA Operating Environment Architecture - Book III (version 3.0B)*
- *POWER9 Processor Programming Model Bulletin*

Except for *Table 4-8* on page 83, all references to the optional Secure Memory Facility (SMF) are found in *Section 24.3 Secure Memory Facility* on page 326. The rest of this chapter applies regardless of whether or not SMF is enabled by the POWER9 hardware.

4.1 Book I - User Instruction Set Architecture

This section of the document identifies version 3.0B architectural implications of the POWER9 design point as they relate to the User Instruction Set Architecture (UISA). This is accomplished by walking through each of the relevant sections of Book I and highlighting the POWER9 solution to the architectural flexibility provided by the Power ISA.

4.1.1 Instruction Classifications

The POWER9 processor core implements all Book I instructions specified in the *Power ISA (Version 3.0B)*.

4.1.1.1 Illegal Instructions

An attempt to execute an illegal instruction as defined in the *Appendix A. Illegal Instructions* of the *Power ISA (Version 3.0B)* results in a hypervisor emulation assistance interrupt.

4.1.1.2 Instructions Supported

The POWER9 core supports all of the instructions described in the *Power ISA User Instruction Set Architecture - Book I (version 3.0B)*. Furthermore, it supports the Service Processor "Attention" described in *Appendix B. Reserved Instructions* of the *Power ISA (Version 3.0B)*. This instruction is conditionally enabled by HID[3] = '1'. When enabled, this instruction is a user-level instruction.

4.1.1.3 Invalid Forms

In general, the POWER9 core handles *invalid forms* of instructions in the manner that is most convenient for the particular case (within the scope of meeting the boundedly-undefined definition described in the Power ISA). This document specifies the behavior for these cases. However, it is not recommended that software or other system facilities make use of the POWER9 behavior in these cases because such behavior might be different in another processor that implements the Power ISA.

The POWER9 core ignores the state of reserved bits in the instructions (denoted by "///" in the instruction definition) and executes the instruction normally. Software should set these bits to '0' per the Power ISA.

4.1.2 Branch Processor

4.1.2.1 Instruction Fetching

In an effort to increase performance, the POWER9 processor does instruction prefetching before it determines whether or not particular instructions will actually execute. This prefetching follows all of the architectural constraints relative to caching-inhibited and guarded regions of storage. More information on instruction fetching is available in *Section 25.1.3 Instruction Fetch* on page 336.

4.1.2.2 Branch Prediction

The POWER9 processor core uses several dynamic branch prediction mechanisms to improve performance. See *Section 25.1.3.4 Branch Prediction* on page 339. When enabled, software can override the hardware mechanisms for branch prediction by using the architected BO field "a" and "t" hint bits in the instruction itself as described in the *Power ISA (Version 3.0B)*.

In addition, for **bclr** instructions, a link stack (or call-return stack) is used to predict the target address of the branch. Similarly, for **bcctr** instructions, local and global count caches are used to predict the target address for this type of branch. To improve the efficiency of these address predictors, the POWER9 core uses the architected BH-field hints associated with several of the branch instructions. These hints are used by the hardware to improve the accuracy of the link stack and the count cache.

Although the overall performance of the machine is strongly dependent on these branch prediction mechanisms, a set of firmware-accessible mode bits is available to disable these features via scan initialization.

4.1.2.3 Instruction Cache Block Touch Hint

The POWER9 core supports the instruction cache block touch (**icbt**) instruction. However, instead of bringing the instructions into the level 1 (L1) cache as described in the Power ISA, it prefetches the instructions into the level 2 (L2) cache. Thus, **icbt** is implemented internally as a data cache block touch for store (**dcbtst**) hint instruction.

4.1.2.4 Out-of-Order Execution and Instruction Flushes

The POWER9 processor uses out-of-order instruction execution. Instructions can be speculative on a predicted branch direction, or simply speculative beyond an instruction that might cause an interrupt condition. In the event of a misprediction or an interrupt, instructions from the mispredicted path and the results produced by those instructions are discarded, presenting the effect of sequentially executed instructions down the appropriate branch paths and precise exceptions as required by the *Power ISA (Version 3.0B)*. For details and exceptions about the rules for obeying the sequential execution model, see the following sections: *Instruction Execution Order* in *Book I*, *Definitions* in *Book II*, and *Definitions and Notation* in *Book III*.

4.1.2.5 Branch Processor Instructions with Undefined Results

The results of executing an invalid form of a branch instruction or an instance of a branch instruction for which the architecture specifies that some results are undefined are described as follows. Only results that differ from those specified by the architecture are described in the following list.

- Instructions with reserved fields:
  Bits in reserved fields including the z-bits in the BO field are ignored. The results of executing an instruction in which one or more of these bits are '1' is the same as if the bits were '0'.
- **bcctr** and **bcctrl** instructions:
  If BO[2] = '0', the contents of CTR (before any update) are used as the target address and for the test of the contents of CTR to resolve the branch. The contents of the CTR are then decremented and written back to the CTR.

4.1.3 Fixed-Point Processor

4.1.3.1 Fixed-Point Exception Register

The Power ISA defines the Fixed-Point Exception Register (XER) bits: XER[0:15], XER[35:43] and XER[46:56] as reserved. An **mfxer** returns the value as shown in *Table 4-1*.

In the POWER9 core, the XER is implemented in several parts:

- XER renamed fields F0:F6 (see *Table 4-1*) are stored in an architected register file (ARF). The ARF contains latches that store the F0:F6 fields for each of four threads. The ARF contains four (one per thread) transactional memory (TM) copies of the F0:F6 fields.
- The SO bit is not renamed and is only updated at completion time. One copy for each of the four threads and four TM copies are shared by FX0, FX1, FX2, and FX3.

Table 4-1. XER Bits and Fields (Sheet 1 of 2)

XER Bits	Name	Field	Read/Write Behavior
0:15	Reserved	Unimplemented	Returns zeros on **mfxer**.
16:31	Reserved		Returns zeros on **mfxer**.
32	SO	SO	Set to '1' whenever OV = '1', except when **mtxer** sets SO = '0' and OV = '1'. This bit can be set to '0' or '1' by **mtxer**. An **mfxer** reads the bit contents. This bit is implemented inside the mapper, but it is only updated at completion time.
33	OV	F1	Set to '0' or '1' by various fixed-point instructions with OE = '1' or by **mtxer**. An **mfxer** reads the bit contents.
34	CA	F2	Set to '0' or '1' by add-carrying, subtract-from carrying, shift-right algebraic-type instructions, and by **mtxer**. An **mfxer** reads the bit contents.
35:43	Reserved	F2	Returns zeros on **mfxer**.

Table 4-1. XER Bits and Fields (Sheet 2 of 2)

XER Bits	Name	Field	Read/Write Behavior
44	OV32/Reserved	F2	There are two bits representing bit 44, an OV32 bit and a reserved bit. • The **mfxer** reads the OV32 bit if PCR[v2.07] = '0' and reads the reserved bit if PCR[v2.07] = '1'; this applies independent of privilege state and is the only PCR effect on bit 44. • The **mtxer** instruction writes both bits. • The **mcrxrx** instruction reads only the OV32 bit because it is an invalid instruction independent of the PR bit when PCR[v2.07] = '1'. • The instructions that implicitly set the OV bit also set the OV32 bit as described in the *Power ISA (Version 3.0B)*, and do not modify the reserved bit.
45	CA32/Reserved	F1	There are two bits representing bit 45, a CA32 bit and a reserved bit. • The **mfxer** reads the CA32 bit if PCR[v2.07] = '0' and reads the reserved bit if PCR[v2.07] = '1'; this applies independent of privilege state and is the only PCR effect on bit 45. • The **mtxer** writes both bits. • The **mcrxrx** reads only the CA32 bit because it is an invalid instruction independent of the PR bit when PCR[vv2.07] = '1'. • The instructions that implicitly set CA also set the CA32 bit as described in the *Power ISA (Version 3.0B)* and do not modify the reserved bit.
46:56	Reserved	F5	Written by **mtxer.** An **mfxer** reads the bit contents.
57:63	String length	F6	String length field used **lswx** and **stswx**. Written by **mtxer**. An **mfxer** reads the bit contents.

4.1.4 Storage Access Alignment Support Overview

Most storage accesses are performed without software intervention (such as, an alignment interrupt). For more information on misaligned cases that do not result in an interrupt, see *Section 25 Performance Profile* on page 333. The storage accesses that result in an interrupt condition are described in the following sections.

4.1.4.1 Alignment Interrupts

The LSU reports an alignment interrupt for the following conditions:

- Storage operand is not on a natural alignment boundary.
  - Halfword boundary:
    - **— lharx**
    - **— sthcx.**
  - Word boundary:
    - **— lwarx, lwat**
    - **— stwcx, stwat**
  - Doubleword boundary:
    - **— ldarx**, **ldat**
    - **— stdcx, stdat**
  - Quadword boundary:
    - **— lfdp**, **lfdpx**, **lq**, **stfdp**, **stfdpx**, **stq**, **stqcx**, **lqarx**
  - Cache-line (128-byte) boundary:
    - **— copy, paste.**

- Atomic memory operations take an alignment interrupt for cases identified by ALI in *Table 4-2*. Some cases only take an alignment interrupt for specific Function Code (FC) values as indicated. If no FC is indicated but ALI is indicated, the alignment interrupt occurs for any FC value.

Table 4-2. Alignment Interrupt for AMO Cases

AMO Instruction	x'00'	x'04'	x'08'	x'0C'	x'10'	x'14'	x'18'	x'1C'
lwat	FC '11100' ALI							FC '11000' ALI
								FC '11001' ALI
ldat	FC '11100' ALI	ALI		ALI		ALI	FC '11000' ALI	
							FC '11001' ALI	ALI
stwat								FC '11000' ALI
stdat		ALI		ALI		ALI	FC '11000' ALI	ALI

Note: ALI = alignment interrupt; FC = function code

- Little-endian mode
  - **lmw**, **lswi**, **lswx**
  - **stmw**, **stswi**, **stswx**
- Caching-inhibited storage
  - **lfdp**, **lfdpx**, **lmw**, **lswi**, **lswx, lxvl, lxvll**
  - **dcbz**, **stfdp**, **stfdpx**, **stmw**, **stswi**, **stswx, stxvl, stxvll**
  - Any load or store not on a natural alignment boundary:
    - Halfword boundary:
      - **lha**, **lhau**, **lhaux**, **lhax**, **lhbrx**, **lhz**, **lhzcix**, **lhzu**, **lhzux**, **lhzx, lxsihzx**
      - **sthbrx, sth**, **sthcix**, **sthu**, **sthux**, **sthx, stxsihx**
    - Word boundary:
      - **lfiwax**, **lfiwzx**, **lfs**, **lfsu**, **lfsux**, **lfsx**, **lwa**, **lwaux**, **lwax**, **lwbrx**, **lwz**, **lwzcix**, **lwzu**, **lwzux**, **lwzx**, **lxssp, lxvwsx, lxsiwax, lxsiwzx, lxsspx**
      - **stfiwx**, **stfs**, **stfsu**, **stfsux**, **stfsx**, **stwbrx**, **stw**, **stwcix**, **stwu**, **stwux**, **stwx**, **stxssp, stxsiwx, stxsspx**
    - Doubleword boundary:
      - **ld**, **ldbrx**, **ldcix**, **ldu**, **ldux**, **ldx**, **lfd**, **lfdu**, **lfdux**, **lfdx**, **lxsd, lxsdx, lxvdsx**
      - **std**, **stdbrx**, **stdcix**, **stdu**, **stdux**, **stdx**,**stfd**, **stfdu**, **stfdux**, **stfdx**, **stxsd, stxsdx**
    - Quadword boundary:
      - **lxvd2x**, **lxvw4x**, **stxvd2x**, **stxvw4x, lxv, stxv, lxvx, stxvx, lxvh8x, stxvh8x, lxvb16s, stxvb16x**

4.1.4.2 Storage Control Attribute Caused Data Storage Interrupt or Hypervisor Data Storage Interrupts

The LSU reports either a data storage interrupt (DSI) or hypervisor data storage interrupt (HDSI) for the following conditions as permitted by the *Power ISA (Version 3.0B)*:

- The effective address specified by a **lq**, **stq**, **lwat**, **ldat**, **lbarx**, **lharx**, **lwarx**, **ldarx**, **lqarx**, **stwat**, **stdat**, **stbcx.**, **sthcx.**, **stwcx.**, **stdcx.**, **stqcx., copy** or **paste** instruction refers to storage that is caching inhibited; or the effective address specified by a **lwat**, **ldat**, **stwat**, or **stdat** instruction refers to storage that is guarded.
- An attempt is made to execute one of the hypervisor accessible (Book IIIS) **lbzcix**, **lhzcix**, **lwzcix**, **ldcix**, **stbcix**, **sthcix**, **stwcix**, or **stdcix** instructions with MSR[DR] = '1' or specifying a storage location (page) that was previously accessed as non-guarded using the Hypervisor Real Mode Storage Control facility.

4.1.5 Fixed-Point Load and Store Instructions

The POWER9 core implements the fixed-point load and store instructions per *Power ISA (Version 3.0B)*. The Power ISA specifies that fixed-point loads and stores to storage, which are neither caching inhibited nor write-through and are aligned on their operand size boundary, are performed atomically. The POWER9 processor exceeds this requirement such that all fixed-point loads and stores that do not cross a doubleword boundary are performed atomically. As implied by *Section 4.1.4.1 Alignment Interrupts* on page 54, most forms of unaligned load operations are executed entirely in hardware.

There are some cases where the Power ISA states that some portion of the results of the instructions are undefined or some forms of the instructions are invalid. See *Section 4.2 Fixed-Point Invalid Forms and Undefined Conditions* on page 58 for details.

4.1.5.1 Fixed-Point Load and Store Multiple Instructions

Note: These instructions are provided for compatibility with legacy software. Software should use a sequence of load or store instructions for optimal performance.

The **lmw** and **stmw** instructions, regardless of operand alignment, are executed in hardware, even when they cross page and segment boundaries. These instructions are not considered atomic. However, the individual storage accesses associated within the instructions are atomic. If a **stmw** crosses a page boundary, and the second page translation signals an exception condition, then after the interrupt is taken, it appears as though none, some, or all of the accesses to the first page have occurred, and none of the accesses to the second page have occurred. On the other hand, for the **lmw** instruction that cross a page boundary where the second page translation signals an exception condition, some of the target registers might not be updated.

An attempt to execute a non-word-aligned **lmw** or **stmw** does not cause an alignment interrupt.

An attempt to execute an **lmw** or **stmw** to storage-marked cache inhibited causes an alignment interrupt.

See *Section 4.1.4 Storage Access Alignment Support Overview* on page 54 for details.

The architecture allows these instructions to be interrupted by certain types of asynchronous interrupts (external interrupts, decrementer interrupts, machine check interrupts, and system reset interrupts). However, the POWER9 core does not process an asynchronous interrupt in the middle of one of these instructions.

4.1.5.2 Fixed-Point Move Assist Instructions

Note: These instructions are provided for compatibility with legacy software. Software should use a sequence of load or store instructions for optimal performance.

The **lswi, lswx**, and **stswi stswx** instructions, when aligned on a word boundary, are executed in hardware, even when they cross page and segment boundaries. These instructions are not considered atomic. Furthermore, the individual storage accesses associated within the instructions are not atomic. If a store string operand crosses a page boundary, and the second page translation signals an exception condition, then after the interrupt is taken, it appears as though none, some, or all of the accesses to the first page have occurred, and none of the accesses to the second page have occurred. On the other hand, for a load string operand that crosses a page boundary where the second page translation signals an exception condition, some of the target registers might not be updated.

An attempt to execute a non-word-aligned **lswi, lswx, stswi,** or **stswx** causes an alignment interrupt.

An attempt to execute an **lswi, lswx**, **stswi**, and **stswx** to storage marked cache inhibited causes an alignment interrupt.

See *Section 4.1.4 Storage Access Alignment Support Overview* on page 54 for details.

The architecture allows these instructions to be interrupted by certain types of asynchronous interrupts (external interrupts, decrementer interrupts, machine check interrupts, and system reset interrupts). However, the POWER9 core does not process an asynchronous interrupt in the middle of one of these instructions.

4.1.5.3 Integer Select Instruction

The POWER9 core implements the integer select (**isel**) instruction as defined in the Power ISA.

4.1.5.4 Fixed-Point Logical Instructions

The architecture defines the *preferred NOP* to be '**ori** 0,0,0'. In the POWER9 processor, this NOP form is recognized by the hardware and allowed to complete without taking any execution resources. This makes the instruction valuable for padding other instructions to achieve better alignment or better separation

4.1.5.5 Access to Performance Monitor Special Purpose Registers

The POWER9 core supports the following performance monitor unit (PMU) special purpose registers (SPRs) as specified in the *Power ISA (Version 3.0B)*:

- PMC1 - Performance Monitor Counter 1
- PMC2 - Performance Monitor Counter 2
- PMC3 - Performance Monitor Counter 3
- PMC4 - Performance Monitor Counter 4
- PMC5 - Performance Monitor Counter 5
- PMC6 - Performance Monitor Counter 6
- MMCR0 - Monitor Mode Control Register 0
- MMCR1 - Monitor Mode Control Register 1
- SIAR - Sampled Instruction Address Register
- SDAR - Sampled Data Address Register

4.1.5.6 Move to/from Condition Register Fields Instructions

The architecture warns that updating a subset of the **CR** fields on an **mtcrf** instruction might have poorer performance than updating all of the fields. For best performance in the POWER9 processor, software should use the single-field variants (**mtocrf** and **mfocrf**) of these instructions as described in the Power ISA. See *Appendix A Instruction Properties* on page 375 for more details.

4.2 Fixed-Point Invalid Forms and Undefined Conditions

The results of executing an invalid form of a fixed-point instruction or an instance of a fixed-point instruction when the architecture specifies that some results are undefined are described in the following list (for the cases when executing an instruction does not cause an exception).

- Instruction with Reserved Fields
  Bits in reserved fields are ignored; the results of executing an instruction in which one or more reserved bits are '1' is the same as if the bits were '0'.
- Load with Update Instructions (RA = 0)
  EA is placed into R0.
- Load with Update Instructions (RA = RT)
  The storage operand addressed by EA is accessed. The displacement field is added to the data returned by the load and placed into RT.
- Load Quadword Instruction (RTp is odd or RTp = RA)
  The POWER9 processor always takes a hypervisor emulation assistance interrupt anytime RTp is an odd register, RTp = RA (including when RA = 0) or RTp = RB for **lq.**
- Load Quadword and Reserve Indexed Instruction (RTp is odd, RTp = RA, RTp = RB)
  The POWER9 processor always takes a hypervisor emulation assistance interrupt anytime RTp is an odd register, RTp = RA (including when RA = 0) or RTp = RB for **lqarx**.
- Load Multiple Instructions (RA in the range of registers to be loaded)
  If an exception (for example, data storage or external) causes the execution of the instruction to be interrupted, the instruction is restarted, RA has been altered by the previous partial execution of the instruction, and RA is less than or greater than '0', the new contents of RA are used to compute EA.
- Load Multiple Instructions (causing a misaligned access)
  For a Load Multiple Word instruction, if the storage operand specified by EA is not a multiple of 4, the access is performed anyway (that is, naturally). No alignment interrupt is taken.
- Load String Instructions (zero length string)
  RT is not altered.
- Load String Instructions (RA and/or RB in the range of registers to be loaded)
  If RA and/or RB is in the range of registers to be loaded, the results are as follows.
  Indexed Form: If RA = 0, let Rx be RB; otherwise let Rx be the register specified by the smaller of the two values in instruction fields RA and RB. If RT = Rx, no registers are loaded; otherwise, registers RT through RX - 1 are loaded as specified in the architecture (that is, only part of the storage operand is loaded).
  Immediate Form: If RA = 0, the instruction is executed as if it were a valid form. If RA = RT, no registers are loaded; otherwise registers RT through RA - 1 are loaded as if the instruction were a valid form but specifying a shorter operand length**.**

- Store with Update Instructions (RA = 0)
  EA is placed into R0.
- Store Quadword and Store Quadword Conditional Instruction (RSp is odd)
  For the **stq** and **stqcx.** instructions, the contents of RSp are stored into the words of storage addressed by EA and EA + 4 respectively. If RSp is odd, the low-order bit of the register number is considered to be '0' such that RSp - 1 and RSp are stored into the words in storage addressed by EA and EA + 4 respectively.
- **subfic**, **subfc**, and **subfco** Instructions and their Rc = 1 Forms
  If RA[0:15] = x'0000', XER[CA] reflects the carry-out of bit 16; otherwise, it reflects the carry-out of bit 40.
- **divw**, **divwo**, **divwu**, and **divwuo** Instructions
  RT[0:31] is set to x'00000000'.
- **mulhw** and **mulhwu** Instructions
  RT[0:31] contains the same result as RT[32:63].
- Divide Instructions (divide by zero)
  If the divisor is 0, RT is set to zero. If Rc = '1' also, CR0 is set to '0010'.
- Trap Word Immediate and Trap Word Instructions [TO = ('11110' | '11100')]
- Move To/From Special Purpose Register Instructions
  Table 4-8 on page 83 describes the read/write **mtspr** behavior for an spr value that is not defined for the implementation.
- Move From Time Base Instruction
  The **mftb** instruction is treated as an alias for the "mfspr Rx, 268" instruction. The results are the same as when executing an "mfspr Rx, 268" instruction.
- Move From Condition Register Instruction
  The entire CR is copied into RT[32:63]. RT[0:31] is set to zero.
- Move From One Condition Register Field Instruction (only 1 bit of FXM set to '1')
  Let n be the bit set to '1' in the FXM field. The CR field *n* is copied to RT[(4 × n + 32):(4 × n + 35)]. The remaining bits are set to zero.
- Move From One Condition Register Field Instruction (multiple bits of FXM set to '1')
  Let n be the first bit (from left to right) set to '1' in the FXM field. The CR field *n* is copied to RT[(4 × n + 32):(4 × n + 35)]. The remaining bits are set to zero.

4.3 Floating-Point Processor (FP, VMX, and VSX)

The POWER9 VSU contains four double-precision floating-point units. Each of these units is optimized for fully pipelined double-precision multiply-add functionality. In addition, each unit is capable of performing the floating-point divide and square root instructions. The complex integer instructions from the VMX architecture are also executed on the floating-point unit datapath.

The POWER9 VSU implements the VSX architecture, specifying 2-way double-precision or 4-way single-precision operations. Dependent floating-point instructions have a minimum issue-to-issue interval of six cycles. The vector single precision throughput has improved because it is possible to execute two 4-way single-instruction, multiple-data (SIMD), single-precision instructions per cycle.

Legacy binary floating-point and VMX architectures are also fully supported in the POWER9 VSU.

4.3.1 Vector Single-Precision Bandwidth

In the POWER9 core, the double-precision FPU supports simultaneous execution of two vector single-precision operations. This increases the single-precision bandwidth of the POWER9 core to 16 floating-point operations per second (FLOPs) per cycle.

In the POWER9 core, the convert and estimate instructions are executed in a fully pipelined manner, as well as the increased bandwidth of the multiply-add and move instructions. From the floating-point instructions, only the divide and square-root instructions cannot be started every cycle.

The compares, minimum/maximum, and test-for-software divide/square-root instructions are now executed on the vector integer (XS) pipeline to take advantage of the shorter latency. Also, the move-from-FPSCR and move-to-FPSCR instructions are separated from the floating-point datapath.

4.3.2 IEEE Compliance

The POWER9 implementation of binary floating-point (BFP), decimal floating-point (DFP), and vector-scalar floating-point (VSX) architecture complies with the IEEE P754-2008 standard.

4.3.2.1 Non-IEEE Modes

If FPSCR[NI] is set, the architecture allows a change in the behavior of the binary floating-point unit (BFU) instructions and the VSX floating-point instructions. See the sections Floating-Point Facility and Vector-Scalar Floating-Point Instructions in the *Power ISA (Version 3.0B)*. The intent is to be faster in some cases. This feature is not implemented in the POWER9 core. Setting FPSCR[NI] does not have any effect.

The architecture requires implementation of VSCR[NJ]. This alters the behavior of the VMX floating-point instructions [see the section Vector Facility in the *Power ISA (Version 3.0B)*] by replacing denormal operands and results with the value '0'. There is no performance difference. Thus, there is no requirement to use VSCR[NJ] = '1' in the POWER9 implementation.

Denormal operands are always handled at full-speed. Denormal results are also handled at full-speed. The only exception is for the double-precision divide instructions **fdiv**, **xsdivdp**, and **xvdivdp**. Their latency is five cycles longer if the exact result is smaller than 2^{-1022}.

4.3.3 Floating-Point Exceptions

Precise floating-point exceptions are provided whenever either of the floating-point enabled exception mode bits (MSR[FE0], MSR[FE1]) are set. In all cases, the floating-point instructions are executed out-of-order, and any resulting exceptions are sorted out at completion time. In some cases, the hardware flushes the pipeline and re-dispatches the instructions individually to provide the precise exception. Because this only happens if an interrupt is to be taken, it does not represent a measurable slowdown in performance.

4.3.4 Floating-Point Load and Store Instructions

Most forms of unaligned floating-point storage accesses are executed entirely in hardware.

4.3.4.1 Scalar Load and Store Atomicity

The *Power ISA (Version 3.0B)* requires binary floating-point and VSX scalar load and store accesses be treated as atomic provided they are aligned on an operand boundary and access storage that is not caching inhibited. The POWER9 core complies with the Power ISA in this regard. Furthermore, binary floating-point and VSX scalar load and store accesses, which do not cross a doubleword boundary and access storage that is not caching inhibited, are also atomic.

4.3.4.2 Vector Load and Store Atomicity

The *Power ISA (Version 3.0B)* requires each doubleword of vector (both VMX and VSX) load and store accesses be treated as atomic provided they are doubleword aligned and access storage that is not caching inhibited. The POWER9 core complies with the Power ISA in this regard.

4.3.5 Heterogeneous Precision Arithmetic

The following instructions are referred to as scalar single-precision arithmetic instructions:

- **fadds**[**.**], **xsaddsp**, **fsubs**[**.**], **xssubsp**, **fmuls**[**.**], **xsmulsp**
- **fmadds**[**.**], **xsmadd**[**am**]**sp**, **fmsubs**[**.**], **xsmsub**[**am**]**sp**
- **fnmadds**[.], **xsnmadd**[**am**]**sp**, **fnmsubs**[**.**], **xsnmsub**[**am**]**sp**
- **fsqrts**[**.**], **xssqrtsp**, **fdivs**[**.**], **xsdivsp**
- **fres**[**.**], **xsresp**, **frsqrtes**[**.**], **xsrsqrtesp**

4.3.5.1 NaN Propagation

If a single-precision arithmetic instruction propagates a not-a-number (NaN) where any of the fraction bits [24:52] is nonzero, the resulting quiet not-a-number (QNaN) has all of the fraction bits [24:52] cleared to zero.

4.3.5.2 Square Root Overflow and Underflow

Due to the compacting nature of the square-root operation, the instructions **fsqrts**, **xssqrtsp**, **frsqrtes**, and **xsrsqrtesp** cannot underflow or overflow if their operands are representable in single-precision format. However, if the operand is not representable in single-precision format, an underflow or overflow can occur. This result is recorded in the FPSCR.

4.3.5.3 Hardware Behavior on Enabled Underflow and Enabled Overflow Exception

If FPSCR[UE] is enabled and an underflow occurs, the contents of the result register and FPSCR status are not defined for scalar single-precision (SP) instructions. The hardware takes the following actions:

1. Underflow exception is set, FPSCR[UX] = '1'.
2. The exponent of the normalized intermediate result is adjusted by adding 192.
3. The double-precision bias of 1023 is added to the exponent.
4. The biased exponent is reduced to 11 bits by ANDing with x'7FF'.
5. The adjusted rounded result is placed into the target FPR.
6. FPSCR[FPRF] is set to indicate a normalized number.

If FPSCR[OE] is enabled and an overflow occurs, the contents of the result register and the FPSCR status are not defined for scalar SP instructions. The hardware takes the following actions:

1. Overflow exception is set, FPSCR[OX] = '1'.
2. The exponent of the normalized intermediate result is adjusted by subtracting 192.
3. The double-precision bias of 1023 is added to the exponent.
4. The biased exponent is reduced to 11 bits by ANDing with x'7FF'.
5. The adjusted rounded result is placed into the target FPR.
6. FPSCR[FPRF] is set to indicate normalized number.

4.3.6 Handling of Denormal Single-Precision Values in Double-Precision Format

Unlike previous generation processors, such as the POWER8 processor, the POWER9 processor is capable of handling denormal single-precision values as inputs for all subsequent instructions. Whereas, in some cases, the POWER8 processor takes a soft-patch interrupt to allow the interrupt handler to reformat the input operands to a double-precision format and then re-execute the instruction, the POWER9 processor simply executes normally regardless of how that number was produced.

4.3.7 Floating-Point Invalid Forms and Undefined Conditions

The results of executing an invalid form of a floating-point instruction or an instance of a floating-point instruction when the architecture specifies that some results are undefined are described in the following list (for the cases when executing an instruction does not cause an exception).

- Scalar single-precision instructions with operands not representable in single-precision format.
  See *Section 4.3.5 Heterogeneous Precision Arithmetic* on page 61.
- Instructions with reserved fields.
  Bits in reserved fields are ignored. The results of executing an instruction when one or more reserved bits are '1' is the same as if the bits were '0'.
- Load or store floating-point with update instructions (RA = 0).
  EA is placed into R0.
- Floating-point store single instructions (exponent < 874 and FRS[9:31] less than or greater than '0').
  The value placed in storage is a '0' with the same sign as the value in the register.
- Scalar floating-point instructions.
  VSR[64:127] is set to x'0000_0000_0000_0000'.

- Scalar convert to integer word instructions (**xscvdpuxws**, **xscvdpsxws**, **fctiwuz**, **fctiwz**, **ctiwu**, **fctiw**).
  VSR[0:31] is set to VSR[32:63].
- VSX scalar convert from double-precision to single-precision (**xscvdpsp**, **xscvdpspn**).
  VSR[32:63] is set to VSR[0:31].
- Scalar convert to integer instructions.
  FPSCR[FPRF] is set to '00000'.
- VSX vector convert from double-precision to single-precision (**xvcvdpsp**).
  VSX vector convert double-precision to integer word (**xvcvdpsxws**, **xvcvdpuxws**).
  VSX vector convert from integer doubleword to single-precision (**xvcvsxdsp**, **xvcvuxdsp**).

  VSR[32:63] is set to VSR[0:31].
  VSR[96:127] is set to VSR[64:95].
- Move from FPSCR instruction.
  FRT[0:63] is set to FPSCR[0:63] with the first 29 bits set to zero.
- Scalar reciprocal estimate instructions: **fre**, **fres**, **xsredp**, **xsresp**, **frsqrte**, **frsqrtes**, **xsrsqrtep**, **xsrsqrtesp**.
  FPSCR[FR] and FPSCR[FI] are set to '0' and FPSCR[XX] is unchanged, even if an overflow exception occurs.
- VSX vector floating-point reciprocal estimate instructions: **xvredp**, **xvresp**, **xvrsqrtedp**, **xvrsqrtesp**.
  FPSCR[XX] is unchanged, even if an overflow exception occurs.
- Disabled overflow exception (OX = '1', OE = '0').
  For divide and square root instructions, FPSCR[FR] is set to '1' if the result is rounded to $\pm\infty$, and set to '0' if the result is rounded to the largest representable number. For scalar reciprocal estimate instructions, FPSCR[FR] is set to '0'. For all other instructions, FPSCR[FR] is set to '1' if a disabled overflow exception occurs.
- Decimal floating-point quad instructions, where an odd target or source register is specified, are considered invalid forms. The POWER9 core always ignores the low-order VSR bit number and addresses the even-odd resulting register pair. This applies to both the source and target register pairs.
- For cacheable (I = 0) memory accesses, the POWER9 core implements the load vector element instructions the same as **lvx** (same as the POWER8 core). Thus, those bytes in the target VSR that are undefined in the Power ISA for the load vector element instructions contain the value that would be written there if an **lvx** instruction was executed instead. For non-cacheable memory accesses, the load vector element instructions are implemented as described in the *Power ISA (Version 3.0B)*. Note that in terms of Data Address Watchpoint Register (DAWR) match criteria, a match will only occur for the bytes specified, as written to the target VSR by the Power ISA, regardless of whether the access is to cacheable or non-cacheable storage.
- For the VSX scalar loads, the Power ISA defines the right-most elements of the target vector-scalar register as undefined. For the POWER9 core, these bits are written with zero.
- VSX load and store vector with length (**lxvl**, **stxvl**, **lxvll**, **stxvll**) specify the number of bytes to load in RB[0:7]. The architecture requires RB[8:63] to be equal to zero. For these instructions, for effective address calculation purposes, the hardware will discard the upper 8 bits of RB in computing the effective address EA; cycle time does not permit for zeroing out RB[8:63]. Therefore, the result of the address generation is: EA = RA[0:63] + RB[8:63].
- Because the binary floating-point registers (FPRs) are mapped to the vector-scalar registers 0 - 31 in the Power ISA, the rightmost doubleword is updated with zero whenever a binary or decimal floating-point

instruction writes the target FPR. This behavior applies to any binary or decimal floating-point instruction that writes an FPR, not just loads.

4.4 Optional Facilities and Instructions

There are no POWER Architecture optional facilities or instructions implemented.

4.5 Little-Endian Mode

The POWER9 core supports true little-endian mode. Byte swapping is performed before data is written to the I-cache and before data is fetched into the execution units; that is, between the D-cache and the execution units (for example, GPR, FPR/VR/VSR).

The load and store multiple instructions and the move-assist instructions are not supported in little-endian mode. Attempting to execute any of these instructions in little-endian mode causes the system alignment error handler to be invoked.

4.6 Book II - Virtual Environment Architecture

4.6.1 Cache

The POWER9 core supports a coherence block size of 128-bytes that is commonly referred to as a cache line.

The POWER9 chip contains three levels of cache hierarchy. All the caches (L1 I-cache, L1 D-cache, and the L2 and L3 caches) are dynamically shared among all the threads on a core. A cache block might be installed by one thread and used by the other threads (as long as the architecture rules pertaining to transactional accesses permit the sharing). The basic coherence block size for the POWER9 core is 128 bytes.

The POWER9 chip automatically maintains the coherency of all data cached in these caches. The L1 cache employs Harvard cache organization, with separate L1 I-cache and L1 D-cache. L2 and L3 caches are unified. Because some levels of the cache hierarchy contain both instructions and data, when an instruction cache reload request is serviced by the L2 and/or the L3 caches, it is done so in a coherent manner.

The processor keeps the instruction storage consistent with the data storage. All cache lines in the L1 I-cache and L1 D-cache are also present in the L2 cache (inclusive property maintained). Instruction fetches to lines previously written by transactional stores executing from any thread will cause the hardware to fail the transaction.

The L1 I-cache is 8-way set associative and is indexed with five effective address bits (EA[51:55]). A particular physical block of memory with a given real address can be found in one of two positions in the L1 I-cache. The tag comparison associated with lookups in this cache (as well as all other caches in the system) are done using physical addresses, so that there are no synonym or alias hazards that must be explicitly handled by the system software.

The L1 D-cache is 8-way set associative and is indexed with five effective address bits (EA[52:56]). A particular physical block of memory with a given real address can only be found at a particular location in the L1 D-cache. On each access, the tag comparison is done with the physical address. On a cache miss, the cache reload mechanism searches the other seven related sets to determine if the required real address block is located elsewhere in the cache, and if so, it appropriately eliminates these copies.

In addition to maintaining caches, the POWER9 chip also includes several types of queues that act as logical predecessors and extensions to the caches. In particular, the machine contains store queues for holding store data above the caches, cast-out queues for holding modified data that has been pushed out of the caches (by the replacement algorithm, cache control instructions, or snoop requests), and others. All of these queues are maintained coherent by the hardware. In general, their presence should not be observable by either software or system hardware.

4.6.2 Classes of Instructions

The POWER9 core implements all of the Book II instructions as described in the following subsections.

4.6.2.1 Instruction Cache Block Touch Instruction

The POWER9 core also supports the instruction cache block touch (**icbt**) instruction by mapping it to **dcbtst** to prefetch a 128-byte line into the L2 cache.

4.6.2.2 Instruction Cache Block Invalidate (icbi)

The POWER9 core implements a split instruction/data (I/D) L1 cache where both caches are kept coherent with the L2 cache. Whenever any modification is made to the cache lines contained in the L2 cache, the L2 invalidates the copies in the L1 I-caches. Because of this, after an **icbi** instruction is translated, the processor core converts it to a NOP and does not broadcast the cache line targeted by the **icbi** instruction as the architecture stipulates. As a result of this and other implementation-specific design optimizations, instead of requiring the instruction sequence specified by the Power ISA to be executed on a per cache-line basis, software must only execute a single sequence of three instructions to make any previous code modifications become visible: **sync**, **icbi** (to any address), **isync**.

4.6.2.3 Instruction Cache Synchronize (isync)

As a performance optimization, the POWER9 core internally tracks and scoreboards **icbi** instructions that are required to be synchronized by the **isync** instruction. When the **isync** instruction is executed, this scoreboard bit is checked to see whether or not the machine must *flush and refetch* the instructions following the **isync**.

4.6.2.4 Vector Category Prefetch Instructions (dss, dst, and dstst)

The vector category data stream instructions **dss**, **dst**, and **dstst** are implemented as NOPs.

4.6.2.5 Data Cache Block Touch Instructions (dcbt and dcbtst)

The data cache block size for **dcbt** and **dcbtst** on the POWER9 core is 128 bytes.

With the exception of trace interrupts, these instructions do not take interrupts. That means, if they miss either the SLB (for HPT translation) or the page table, or they encounter some other type of translation related exception condition, no interrupt will be reported. This property applies regardless of which TH value is specified. Other details that are unique to specific TH values are documented in the following sections.

In general, the **dcbt** and **dcbtst** instructions check the state of the L1 D-cache; and if the address block is not present, it initiates a reload. Note that this might also reload the L2 cache and/or the L3 cache with the addressed block if it is not already present in these caches. If the address block is already present in the L1 D-cache, the cache content is not altered. If the **dcbt** or **dcbtst** instruction does reload the address blocks, it affects the state of the cache replacement algorithm bits.

The **dcbt** and **dcbtst** instructions largely perform their intended operation independent of MSR state bits such as HV, PR or DR. In other words, they perform their intended function as described in the *Power ISA (Version 3.0B)* whether translation is enabled or disabled. Reference bit updates vary by TH value as described in the various sections that follow. Change bits are never set in PTEs by either **dcbt** or **dcbtst** instructions regardless of the TH value.

The POWER9 core implements the optional extension to the **dcbt** instruction that enables software to directly engage a data stream prefetch from a particular address (see *Section 4.6.2.11 Data Cache Block Touch - Transient (TH = '10000')* on page 67).

4.6.2.6 Data Cache Block Touch Instructions (dcbt and dcbtst) - Single Cache Line (TH = '00000')

The **dcbt** and **dcbtst** instructions operate on a single 128-byte cache (address) block specified by the effective address of the storage operand. The **dcbtst** instruction operates exactly the same way as the **dcbt** instruction.

These instructions act as a touch for the D-cache hierarchy, ERAT and the TLB independent of MSR state bits such as HV, PR, and DR. If data translation is enabled (MSR[DR] = '1') and an SLB miss results, the instruction is treated as a NOP. For HPT translation, if a TLB miss results, the instruction reloads the TLB (and sets the reference bit if it is not already set). For Radix translation, if a TLB miss results, the instruction reloads the D-cache hierarchy, ERAT and TLB if the reference bit is already set. However, if the reference bit is not already set, the instruction will be treated as a no-op and the D-cache hierarchy, ERAT and TLB will not be reloaded.

Once past translation, if the page protection attributes prohibit access, the page is marked *cache inhibited* or the page is marked *guarded*, the instruction is finished as a no-op and does not reload the cache. Otherwise, the instruction checks the state of the L1 D-cache, and if it is not present, it initiates a reload as described in *Section 4.6.2.5 Data Cache Block Touch Instructions (**dcbt and dcbtst)*.

4.6.2.7 Data Cache Block Touch - Invalid TH Forms (TH = '00001' through TH = '00111')

The POWER9 core treats **dcbt** and **dcbtst** for the invalid TH values of '00001' through '00111' the same as TH = '00000'. They do go through translation and they do set reference bits.

4.6.2.8 Data Cache Block Touch Data Stream (TH = '01000')

The POWER9 core treats **dcbt** and **dcbtst** with a TH value of '01000' as described in the *Power ISA (Version 3.0B)*. For HPT translation, reference bits will not be set. For Radix translation, atomic PTE update interrupts will not occur.

4.6.2.9 Data Cache Block Touch Data Stream Descriptor (TH = '01010')

The POWER9 core treats **dcbt** and **dcbtst** with a TH value of '01010' as described in the *Power ISA (Version 3.0B)*. For HPT translation, reference bits will be set and instruction execution proceeds as described in the *Power ISA (Version 3.0B)*. For Radix translation, atomic PTE update interrupts will not occur. If the reference bit is already set in the PTE, the instruction reloads the TLB. However, if the reference bit is not already set, the instruction will be treated as a nop and the D-cache hierarchy, ERAT and TLB will not be reloaded.

4.6.2.10 Data Cache Block Touch Data Stream Stride Descriptor (TH = '01011')

The POWER9 core treats **dcbt** and **dcbtst** with a TH value of '01011' as described in the *Power ISA (Version 3.0B)*. For HPT translation, reference bits will be set and instruction execution proceeds as described in the *Power ISA (Version 3.0B)*. For Radix translation, atomic PTE update interrupts will not occur. If the reference bit is already set in the PTE, the instruction reloads the TLB. However, if the reference bit is not already set, the instruction will be treated as a nop and the D-cache hierarchy, ERAT and TLB will not be reloaded.

4.6.2.11 Data Cache Block Touch - Transient (TH = '10000')

The POWER9 core implements the load and store version of the following transient touch instructions: **dcbtct**, **dcbtds**, **dcbtt**, **dcbtstct**, and **dcbtstt**. The transient property of a cache line is retained in the L3 cache for both the load and store version of the transient touch instructions. The transient property of a cache line is retained in the L2 cache for the load version of the transient touch instruction for the case that the line is loaded but not stored into it. In this transient state, the transient line becomes the most likely cache line in its congruence class to be replaced next, thus preserving the other cache lines in that congruence class. This behavior is useful if it is known that a set of lines will be loaded or stored with a low probability for temporal cache reuse and it is desirable that they be as minimally intrusive to the cache as possible (for example, displacing as few lines in the cache as possible). Reading or writing a large array with the help of transient touch instructions only impacts one of the eight sets in the L3 cache. Reading a large array with the help of transient touch instructions only impacts one of the eight sets of the L2 cache.

For HPT translation, reference bits will be set and instruction execution proceeds as described in the *Power ISA (Version 3.0B)*. For Radix translation, atomic PTE update interrupts will not occur. If the reference bit is already set in the PTE, the instruction reloads the TLB. However, if the reference bit is not already set, the instruction will be treated as a no-op and the D-cache hierarchy, ERAT and TLB will not be reloaded.

4.6.2.12 Data Cache Block Touch - No Access Needed Anymore (TH = '10001')

The POWER9 core supports this instruction as specified in the Power ISA. For HPT translation, reference bits will be set and instruction execution proceeds as described in the *Power ISA (Version 3.0B)*. For Radix translation, atomic PTE update interrupts will not occur. If the reference bit is already set in the PTE, the instruction reloads the TLB. However, if the reference bit is not already set, the instruction will be treated as a nop and the D-cache hierarchy, ERAT and TLB will not be reloaded.

4.6.2.13 Data Cache Block Zero (dcbz)

The data cache block size for **dcbz** on the POWER9 core is 128 bytes.

The function of **dcbz** is performed in the L2 cache. As a result, if the block addressed by the **dcbz** is present in the L1 D-cache, the block is invalidated before the operation is sent to the L2 cache logic for execution. The L2 cache gains exclusive access to the block (without actually reading the old data) and performs the zeroing function in a broadside manner.

If the cache block specified by the **dcbz** instruction contains an error (even one that is not correctable with ECC), the contents of all locations within the block are set to zeros in the L2 cache. If the specified block in the L2 cache does not contain a hard fault, a subsequent load from any location within the cache block returns zeros and does not cause a machine-check interrupt.

If the block addressed by the **dcbz** instruction is in a memory region marked *cache inhibited*, or if the L1 D-cache or L2 cache is disabled, the instruction causes an alignment interrupt to occur.

4.6.2.14 Data Cache Block Store (dcbst)

The data cache block size for ***dcbst*** on the POWER9 core is 128 bytes.

The **dcbst** instruction has no direct effect on the L1 D-cache (because it is store-through and it never contains modified data). The **dcbst** instruction also has no direct effect on the L2 cache or L3 cache (both of these are kept coherent with memory and I/O, so that nothing special must be done). As a result, the instruction simply goes through address translation, reports any errors, and is completed. The instruction is not sent to the storage subsystem, and consequently it does not broadcast any transactions onto the inter-processor SMP interconnect.

4.6.2.15 Data Cache Block Flush (dcbf, dcbfl, and dcbflp)

The data cache block size for **dcbf**, **dcbfl,** and **dcbflp** on the POWER9 core is 128 bytes.

The POWER9 core supports **dcbf** (L = 0), **dcbfl** (**dcbf** with L = 1), and **dcbflp** (**dcbf** with L = 3) as specified in the Power ISA.

4.6.2.16 Key Aspects of Storage Control Instructions

In the POWER9 core, all cache control instructions operate on aligned 128-byte sections of storage. *Table 4-3* summarizes many of the key aspects of the storage control instructions.

Table 4-3. Storage Control Instructions

Aspect	Book II Cache Instructions					
	icbi	**dcbt**	**icbt/dcbtst**	**dcbz**	**dcbst**	**dcbf/dcbfl**
Granularity	128 bytes	128 bytes	128 bytes	128 bytes	128 bytes	128 bytes
Semantic checking	load (DSI on storage exception)	load (NOP on storage exception)	load (NOP on storage exception)	store (DSI on storage exception)	load (DSI on storage exception)	load (DSI on storage exception)
"r" bit update	yes	yes	yes	yes	yes	yes
"c" bit update	no	no	no	yes	no	no
L1 I-cache effect	see *Section 4.6.2.3* on page 65	none	none	none	none	none
L1 D-cache effect	none	see *Section 4.6.4.8* on page 74	see *Section 4.6.4.8* on page 74	as define in architecture	NOP	as define in architecture
L2 cache effect	none	see *Section 4.6.4.8* on page 74	see *Section 4.6.4.8* on page 74	see *Section 4.6.2.13* on page 68	see *Section 4.6.2.14* on page 68	see *Section 4.6.2.15* on page 68
L3 cache effect	none	see *Section 4.6.4.8* on page 74	see *Section 4.6.4.8* on page 74	see *Section 4.6.2.13* on page 68	see *Section 4.6.2.14* on page 68	see *Section 4.6.2.15* on page 68
TLB effect	reload as required	reload as required	reload as required	reload as required	reload as required	reload as required
SLB effect	reload as required	None (NOP if miss)	None (NOP if miss)	reload as required	reload as required	reload as required

4.6.2.17 Copy/Paste Instructions

As stated in the *Power ISA (Version 3.0B)*, the **copy** instruction is only permitted to read from local address space. The **paste** instruction is only permitted to write to foreign address space. See *Section 4.9.2 Foreign Address Space Definition and Accessibility* on page 96 for the definition of local versus foreign address space.

4.6.2.18 Near Memory Instruction Support

The POWER9 processor supports the Atomic Memory Operation (AMO) instructions as described in the *Power ISA (Version 3.0B)*. For details regarding when AMOs take alignment interrupts, see *Section 4.1.4 Storage Access Alignment Support Overview* on page 54.

4.6.2.19 Wait Instruction

The **wait** instruction is implemented as described in the *Power ISA (Version 3.0B)*. However, in addition to waking a thread up from the wait state via an interrupt or event-based branch, the thread can also be removed from the wait state via a VAS_notify command sent on the SMP interconnect fabric generated by the Virtual Accelerator Switchboard when there is work for the thread to perform. See *Section 12.3 Core-Core Wakeup Via ASB_Notify* on page 202 for additional information.

See *Section 4.9.2 Foreign Address Space Definition and Accessibility* on page 96 for the definition of local versus foreign address space.

4.6.3 Storage Model

4.6.3.1 Storage Access Ordering

The architecture defines a weakly ordered storage model for most types of storage access scenarios. For these cases, the POWER9 processor takes advantage of this relaxed requirement to achieve better performance through out-of-order instruction execution and out-of-order bus transactions. As a result, if strongly-ordered storage accesses are required, software must use the appropriate synchronizing instruction (**sync**, **ptesync**, **eieio**, or **lwsync**) to enforce order explicitly, or perform these accesses to pages marked with address translation attributes that require the hardware to enforce strong ordering as defined in *Power ISA Operating Environment Architecture - Book III (version 3.0B)*.

In hypervisor real mode, the POWER9 core employs the page-based Real Mode Storage Control (RMSC) facility described in *Power ISA Operating Environment Architecture - Book III (version 3.0B)*.

In any addressing mode, stores to storage marked as *non-guarded*, can be performed out-of-order. Stores to storage marked as *guarded*, cannot be performed out-of-order.

4.6.3.2 Atomicity

The POWER9 core is fully compliant with the architectural requirement for single-copy atomicity on naturally aligned cacheable storage accesses. This includes the quadword data atomicity associated with the **lq, lqarx, stq**, and **stqcx.** instructions. Additional information regarding which instruction accesses are performed atomically are described in *Section 4.1.5.1 Fixed-Point Load and Store Multiple Instructions* on page 56 and *Section 4.3.4 Floating-Point Load and Store Instructions* on page 61.

Furthermore, transactional mode accesses executed by a given thread appear to execute atomically to all other threads in the system. For more information on transactional mode accesses, see *Section 4.6.4 Transactional Memory* on page 71.

4.6.3.3 Atomic Updates and Reservations

Atomic accesses can be performed using the load and reserved and store conditional family of instructions. In the *Power ISA Virtual Instruction Set Architecture - Book II (version 3.0B)*, the load and reserve instructions are: **lbarx, lharx, lwarx, ldarx** and **lqarx**. The store conditional instructions are: **stbcx., sthcx., stwcx., stdcx.**, and **stqcx.** While these instructions differ in the size of the operand read or written, in regards to the establishment or clearing of a reservation, all of the instructions operate on the same size reservation granule. The reservation granule size in the POWER9 core is 128 bytes. There is at most one reservation per thread at any point in time.

Note: The load and reserve instructions and store conditional instructions are generically referred to as **larx** and **stcx** in the remainder of this document.

4.6.4 Transactional Memory

Similar to the POWER8 chip, the POWER9 chip supports the transaction memory (TM) facility as described in the *Power ISA Virtual Instruction Set Architecture - Book II (version 3.0B)*. The transaction memory facility can increase the scalability of multi-threaded applications, reduce the latency of synchronization operations, improve programmability for the developers of multi-threaded applications, and enable speculative optimizations by programmers and compilers through support for fast checkpoint and rollback of the architectural state. Applications that have not been heavily tuned to implement fine-grain locking can better use the large number of available cores, while improved latency of synchronization operations require less fine-tuning of multi-threaded code.

The transactional memory facility is controlled by using a set of instructions in the POWER instruction set. Programmers and compilers can selectively mark a sequence of instructions that appear to execute atomically with respect to other processors and devices. These atomic sequences are called transactions, and can be used to write shared memory applications with fewer sources of lock contention, while avoiding the performance cost and software development cost burdens of the memory barriers required by the existing weak POWER memory model. A transactional sequence is initiated using a **tbegin** instruction, and committed using a **tend** instruction. A transaction can be intentionally aborted through the execution of an unconditional or conditional **tabort** instruction within a transaction. A transaction can also fail for a variety of other reasons; for example, due to conflicting access with another thread. Upon failure, all transactional updates to memory are nullified, Book I registers are reverted to their pre-transaction values, and control flow is redirected to a software-defined failure handler associated with each transaction.

The POWER9 transactional memory facility also includes support for the handling of exception conditions and debugging through the use of suspend mode. Upon the taking of any interrupt during transactional mode, transactional execution is suspended. Memory accesses performed while the thread is operating in suspend mode occur according to the conventional POWER storage model. They are not part of the transaction's atomic access, nor are they discarded should the transaction abort. Transactions can also be directly suspended and resumed using a **tsr** instruction. Through this suspension mechanism, it is possible to support a mixture of transactional and nontransactional code, allowing for a variety of uses, from simple **printf** debugging within a transaction, to transactional operating system services, to an inter-thread signaling mechanism for distributed commit of a set of transactions.

4.6.4.1 TDOOMED

The *Power ISA (Version 3.0B)* defines the TDOOMED bit to be an indication as to whether or not a given transaction has failed. The value of the TDOOMED bit is determined by executing the **tcheck** instruction. In transactional and suspended states, the **tcheck** instruction returns the value of TDOOMED as described in the *Power ISA (Version 3.0B)*. However, the *Power ISA (Version 3.0B)* states that the value of TDOOMED is undefined in the non-transactional state.

4.6.4.2 Transactional Lock Elision and Increased Scalability

Eliminating lock contention is one of the primary uses of the transactional memory facility. Through transaction lock elision (TLE), conventional locking algorithms for mutual exclusion are augmented with code that first attempts to speculatively execute the critical section as a transaction. Such speculation has a number of advantages compared to a conventional lock. If the data protected by the lock is rarely contended, it is probable that multiple threads will be able to concurrently execute the critical section. The false contention on the lock variable is then avoided. In the event that the data is contended, the transaction fails and a failure handler can acquire the lock as would happen in the absence of TLE. After collecting a history of transaction failure rates, software can adaptively disable transactions for a critical section known to be a source of excessive transactional conflicts. It is important to realize that when transactions are used as described previously, the transactional code sequence must still check for the presence of a lock in the event that another thread is performing the atomic accesses using conventional lock-based methods.

4.6.4.3 Reduced Latency of Synchronization Operations

For some applications, gains from the transactional memory facility come from increased concurrency. Others that have frequent synchronization can benefit from the reduced latency of synchronization implemented using transactional memory. Because a conventional lock acquire and release sequence requires **larx**, **stcx**, **isync**, and **lwsync** instructions, while a TLE-enabled lock requires only a **tbegin** and **tend** instruction, some applications might observe better performance due to the lower total latency required to execute these sequences.

4.6.4.4 Improved Programmability

In addition to these performance benefits, the transactional primitives also improve the programmability of POWER systems in the following ways:

- Emerging transactional programming model. While still in its infancy, this programming model is emerging, with support in compilers already widespread. Hardware acceleration of the programming model is expected to provide a competitive advantage for POWER systems to applications written to this interface.
- Shared memory programming without memory barriers. A burden to novice programmers, as well as independent software vendors (ISVs) porting existing code to the POWER platform, the POWER memory model requires significant thought in determining the correct sequence of memory barriers to be used in different situations. With the use of the transactional memory primitives, updates to shared memory can be encapsulated in a transaction, and the intricacies of the weak memory model can be largely ignored.
- Lock-free algorithms. Existing lock-free algorithms are limited by the width of memory atomically read and updated by **larx** and **stcx**. The transactional memory facility provides new flexibility with the ability to atomically modify a set of non-contiguous memory locations.

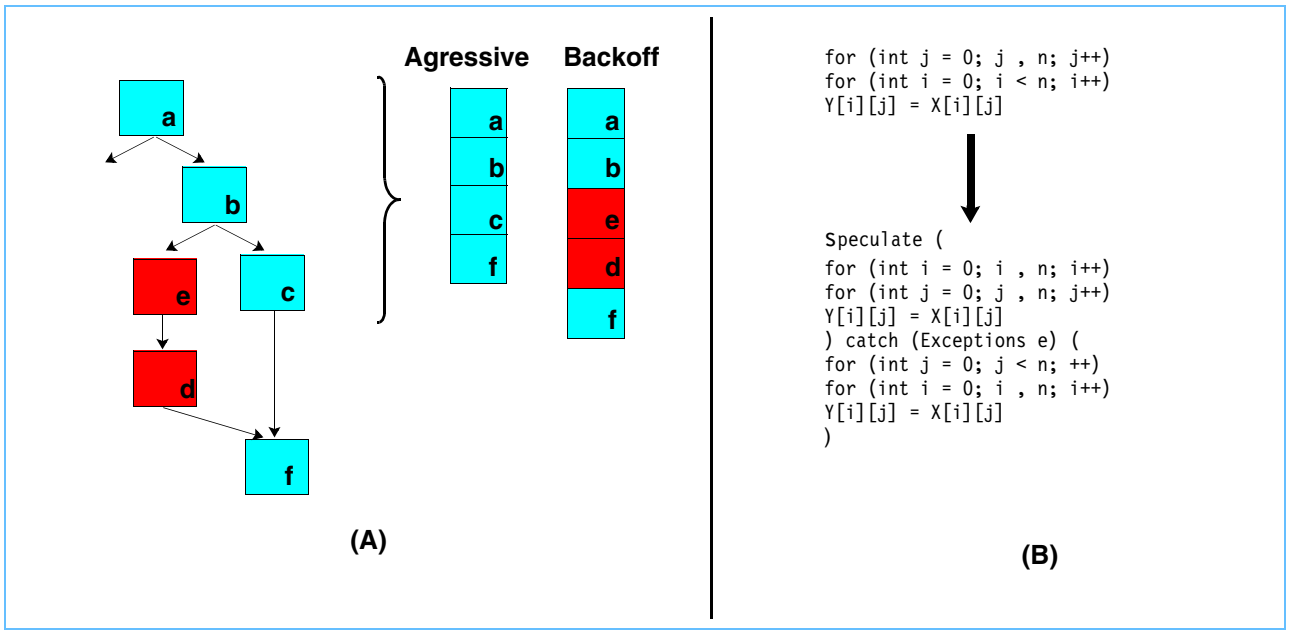
4.6.4.5 Rollback-Only Transaction Enablement of Speculative Optimizations

As described in *Power ISA Virtual Instruction Set Architecture - Book II (version 3.0B)*, the **tbegin** instruction can be used to specify that a rollback-only transaction (ROT) can be used for purposes that do not require accesses to shared storage. While all of the previously described uses of the transactional memory facility leverage its ability to atomically perform a set of shared memory operations with respect to other threads, its support of architectural state checkpoint and rollback is also useful in its own right, as an enabler of speculative optimization.

Figure 4-1 illustrates two examples of speculative optimizations where a piece of code is speculatively transformed, with support from hardware-based checkpoint and rollback. In *Figure 4-1*(A), a set of hot basic blocks (a, b, c, f) are co-located and subsequently optimized, scheduling instructions across basic block boundaries (labeled "aggressive"), while the original version of the code is retained (labeled "backoff"). In *Figure 4-1*(B), a matrix is copied in column-major form, which leads to significant cache misses given the non-trivial size of "n" because each iteration of the inner loop will be touching a different cache line. Because transforming the code to use row-major form can cause exceptions, a simple transformation is not allowed by many languages. Instead, a compiler must rely on a speculative transformation; in the absence of exceptions, the transformation is correct; otherwise, the original version of the code must be executed.

Although both of these transformations can be implemented without hardware support, the bookkeeping that is necessary to enable the successful rollback of architectural state in the case of an exceptional event would likely cause overhead that exceeds any gains from the optimization. With the support of the transactional memory facility, each speculative sequence can be wrapped in a ROT. Should an exceptional event occur, transaction abort can be initiated by a **tabort** instruction and the nonspeculative version of the code immediately called.

Figure 4-1. Speculative Optimizations



4.6.4.6 Transactional Memory Footprint Capacity

Details regarding the transactional memory footprint capacity can be found in *Section 6.6 Transactional Memory Support* on page 163.

4.6.4.7 Implementation-Specific Failure Causes

The Transaction Exception and Status Register (TEXASR) is used to record various failure conditions that are described in the Power ISA. In addition, TEXASR[15] is used to specify various implementation-specific transaction-failure causes that are not architected. The POWER9 processor sets TEXASR[15] = '1' for the following reasons (implementation-specific transactional failure causes):

- Instruction fetch to caching-inhibited page in transactional mode
- Recovery in transactional or suspend mode
- Quiesce request in transaction or suspend mode

The persistent bit is set to '0' for all of these cases.

4.6.4.8 Effects of Cache and Translation Management Instructions on Transactional Accesses

Table 4-4 lists how certain cache, SLB, and TLB management instructions affect transactions.

Table 4-4. Cache, SLB, and TLB Management Instruction Effects on Transactional Accesses (Sheet 1 of 2)

Mode			
Instruction	TM State	Fails Transaction	TEXASR Bit Set
slbmte	T	Always	8 - disallowed
slbmte	S	Never	N/A
slbia	T	Always	8 - disallowed
slbia	S	Never	N/A
slbie	T	Always	8 - disallowed
slbie	S	Never	N/A
slbieg	T	Always	8 - disallowed
slbieg	S	When the virtual address it is attempting to invalidate hits in the SLB bloom filter for the current transaction	14 - translation invalidation conflict
tlbie	T	Always	8 - disallowed
tlbie	S	When the virtual address it is attempting to invalidate hits in the TLB bloom filter for the current transaction	14 - translation invalidation conflict
tlbiel	T	Always	8 - disallowed
tlbiel	S	Never	N/A
dcbt (any TH)	T	Never (unless it causes a castout of the TM footprint)	10 - footprint overflow
dcbt (any TH)	S	Never (unless it causes a castout of the TM footprint)	10 - footprint overflow
dcbst	T	Always	8 - disallowed
dcbst	S	Never (**dcbst** is treated as a NOP in this case)	11 - self-induced conflict
dcbf (L = 0, 1)	T	Always	8 - disallowed
dcbf (L = 0, 1)	S	When the block (line) being pushed out of the cache is part of the TM footprint	11 - self-induced conflict
dcbf (L = 3)	T	Always	8 - disallowed
dcbf (L = 3)	S	Never	N/A
dcbz	T	Never (unless **dcbz** causes a castout of the TM footprint)	N/A
dcbz	S	Case 1: When the block (line) being zero'ed is part of the TM footprint Case 2: When the **dcbz** causes a castout of the TM load or store footprint	Case 1: 11 - self-induced conflict Case 2: 11 - footprint overflow

Table 4-4. Cache, SLB, and TLB Management Instruction Effects on Transactional Accesses (Sheet 2 of 2)

Mode			
Instruction	TM State	Fails Transaction	TEXASR Bit Set
dcbtst	T	Never (unless **dcbtst** causes a castout of the TM footprint)	10 - footprint overflow
dcbtst	S	Never (unless **dcbtst** causes a castout of the TM footprint)	10 - footprint overflow
icbi	T	Always	8 - disallowed
icbi	S	Never (**icbi** is treated as NOP with regards to transaction failure in suspend mode)	N/A
icbt	T	Never (unless **icbt** causes a castout of the TM footprint)	10 - footprint overflow
icbt	S	Never (unless **icbt** causes a castout of the TM footprint)	10 - footprint overflow

4.6.5 Storage Ordering/Barrier Instructions

4.6.5.1 sync Instruction

The POWER9 design achieves high performance by exploiting speculative out-of-order instruction execution. The heavyweight sync (**hwsync)** instruction, as defined in the architecture, acts as a serious barrier to this type of aggressive execution and therefore, can have a dramatic effect on performance. Although the POWER9 core has optimized the performance of **hwsync** to some degree, care should be exercised in the indiscriminate use of this instruction. As a performance consideration, software should attempt to use the lightweight version of **sync** (often referred to as **lwsync** in this document) whenever possible. Unless otherwise stated, **sync** refers to **hwsync**.

The POWER9 core implements the **ptesync** for use in synchronizing both segment and page table updates as described in the *Power ISA Operating Environment Architecture - Book III (version 3.0B)*.

See the *Power ISA Virtual Instruction Set Architecture - Book II (version 3.0B)* and *Power ISA Operating Environment Architecture - Book III (version 3.0B)* for a complete description of the different forms of the **sync** instruction.

4.6.5.2 eieio Instruction

The POWER9 core implements the **eieio** instruction as described in the Power ISA.

In the POWER9 nest logic, the store queues above the L2 cache attempt to gather sequential cacheable and cache-inhibited store operations to improve bandwidth. If this behavior is not required, software must insert either an **eieio** (preferable for performance) or a **sync** to prevent it.

4.6.5.3 miso Instruction

The POWER9 core implements the **miso** instruction as a NOP. It has no effect on the execution of stores.

4.6.5.4 Transactional Memory Instructions

In general, the POWER9 core implements the transactional memory instructions as described in the *Power ISA Virtual Instruction Set Architecture - Book II (version 3.0B)*. The POWER9 core does not however implement the **tbegin** instruction as having an implicit memory barrier as described in the Power ISA. Instead, the hardware applies the barrier effects described in the Power ISA as belonging to the **tbegin** instruction to the **tsr** (**tsuspend** and **tresume**) instructions. In addition, the POWER9 core uses hypervisor software to fully implement suspending and resuming transactions. This assistance is transparent to non-hypervisor software. The **tend** instruction and all the conditional and unconditional **tabort** instructions have the associated barrier functions as described in the *Power ISA Virtual Instruction Set Architecture - Book II (version 3.0B)*.

4.6.6 Data Prefetch

The POWER9 core provides an aggressive hardware-based data-prefetching engine that is designed to work well for stride-one technical workloads with up to eight streams. The eight streams can be dynamically shared among all of the threads. The POWER9 core implements enhanced data prefetching (**edcbt** instruction), where each thread can employ up to eight software-initiated streams in ST mode, four in SMT2 mode, and two in SMT4 mode.

The POWER9 core uses an adaptive prefetch mechanism, which employs L3 nest feedback and long-term averaging to automatically reduce prefetch aggressiveness and increase performance in areas where prefetch consumption or memory bandwidth is low.

The POWER9 core supports instruction cache block touch (**icbt**) by mapping it to **dcbtst** to prefetch into the L2 cache.

The POWER9 core also allows problem state access to DSCR[58], which turns off hardware load-stream prefetching.

4.6.7 Timer Facilities

Time Base

Time base is designed to tick at the rate of time-of-day (TOD). In other words, bit 59 of the Time-Base Register increments at the 32 MHz clock. There is one time base per processor core that is shared by all the threads, running on a core. There is one decrementer per thread.

The POWER9 core implements two time-base modes: POWER9 time-base mode and non-POWER9 time-base mode. They are selectable by setting a mode bit in the Time Facility Management Register (TFMR).

Time Facility Management Register

The Time Facility Management Register (TFMR) is an SPR that is accessible only in the hypervisor state. Executing a move to or move from TFMR in a nonhypervisor state causes a privileged interrupt. There is one TFMR per processor core that is shared among the threads. The TFMR is used as both a status and control register.

POWER9 Time-Base Mode

The time-base function uses an external time-of-day (TOD) clock, which is independent of the processor frequency. This is required to support dynamic frequency variation for power management. The external TOD oscillator can be 16 MHz or 32 MHz. The external TOD oscillator is sampled to provide a 32 MHz step signal, which is distributed to all processors in the system.

Bits 0:59 of the TB are incremented at the 32 MHz frequency as provided by the distributed step signal. Bits 60:63 of the TB are incremented at a fixed frequency of 500 MHz. If the value of bits 60:63 is '1111' (saturated), it is held until the 32 MHz step signal causes bit 59 to change. At that time, bits 60:63 are allowed to change to '0000'.

To support multi-node configurations across multiple oscillator domains, error detection and recovery, and concurrent maintenance, the POWER9 core uses the following means of synchronizing the time bases across all processors.Each multicore processor chip contains a Time-of-Day (TOD) Register. The chip TOD registers are first synchronized across all the processor chips. Then, the time-base registers in each processor core are synchronized to the chip TOD. Also encoded on the step signal is a synchronization pulse that is used for synchronization and error checking. The synchronization mechanism requires system operations to complete within a synchronization interval. The synchronization interval can be set via the TFMR bits to be, for example, 1μs, 2 μs (default, corresponds to TB bit 53), 4 μs, or 8 μs.

Error checking includes parity checks on all registers, and functional checking such as missing step signal detection and synchronization errors. The step signal rate is defined in the POWER9 mode to be 32 MHz, and the logic checks for the correct number of steps for each synchronization signal (which is selected by TFMR). After the TB is operational, the hardware also detects a missing step signal, which requires specifying in the TFMR the maximum number of processor cycles allowed without seeing a 32 MHz step signal, for the fastest allowable operating frequency. The TFMR maximum cycle step time-out should be specified as $(2 \times 31.25$ ns) / (minimum processor cycle time in ns $\times$ 4).

The initial synchronization requires some software sequencing, which is performed by writing values to the TFMR (via **mtspr**). The TFMR also indicates the status of the various time facilities. The status bits in the TFMR are read-only, not modified by **mtspr** to the TFMR. The time facility logic implements error detection for hardware and also for invalid software sequencing. Because synchronized time is critical to a system, writes to the time base or the TFMR that would break synchronization cause the logic to enter an error state and trigger a hypervisor maintenance interrupt.

To initially set the time and synchronize the time-base values, software must synchronize all processors in the system and choose one processor to perform updates to the TFMR via a read-modify-write operation to preserve the other bits. This sequence assumes the external TOD oscillator distribution is already running.

After the chip TOD is running on all chips and the TB is running on the processor that drove this sequence, software must then release the remaining processors to synchronize their TB registers to their corresponding chip TOD.

4.6.8 Hypervisor Decrementer (HDEC)

There is one hypervisor decrementer register per thread. HDEC decrements every time TB bit 63 is incremented. The *Power ISA (Version 3.0B)* defines the HDEC as a 64-bit register architecturally but states that a given implementation can implement fewer than 64 bits. The POWER9 implements 56 bits (that is, bits 8:63) of the HDEC register. The number of bits is not changeable by the LPCR[LD] value.

4.6.9 Decrementer (DEC)

There is one decrementer register per thread. DEC decrements every time TB bit 63 is incremented. The *Power ISA (Version 3.0B)* defines the DEC as a 64-bit register architecturally but states that a given implementation can implement fewer than 64 bits. The POWER9 implements 56 bits (that is, bits 8:63) of the DEC register. The number of bits used in determining when a decrementer interrupt occurs is 56 bits when the LPCR[LD] = '1' but only 32 bits when LPCR[LD] = '0'.

4.6.10 Book II Invalid Forms

The results of executing an invalid form of an instruction in Book II or an instance of such an instruction for which the architecture specifies that some results are undefined, are described here for the cases when executing an instruction does not cause an exception. Only results that differ from those specified by the architecture are described in the following list.

- Instruction with reserved fields
  Bits in reserved fields are ignored; the results of executing an instruction in which one or more reserved bits are '1' is the same as if the bits were '0'.
- Transactional memory instructions and store conditional instructions (bit 31 is ignored)
  Bit 31 of **tbegin.**, **tend.**, **tabort.**, **tabortwc.**, **tabortdc.**, **tabortwci.**, **tabortdci.**, **treclaim.**, **stbcx**., **sthcx.**, **stwcx., stdcx.** and **stqcx.** is ignored. Bit 31 = '1' or bit 31 = '0' is treated the same given that other x-form instructions implicitly set CR and have no "non-record" form variant. Ignoring bit 31 is an acceptable way to handle this invalid form.
- **mftb** instructions
  This instruction produces the same result as the **mfspr** instruction.

4.7 Book III - Operating Environment Architecture

4.7.1 Classes of Instructions

4.7.1.1 Storage Control Instructions

The POWER9 core provides support for the following instructions:

- **tlbie** - TLB invalidate entry
- **tlbiel** - Processor local form of TLB invalidate entry
- **tlbsync** - TLB synchronize
- **slbmte**- Segment lookaside buffer move to entry
- **slbmfev**- Segment lookaside buffer move from entry VSID
- **slbmfee**- Segment lookaside buffer move from entry ESID
- **slbfee.**- Segment lookaside buffer find entry ESID
- **slbie** - SLB invalidate entry
- **slbieg** - SLB invalidate global
- **slbsync** - SLB synchronize
- **slbia** - SLB invalidate all
- **mtmsr** - Move to Machine State Register (32-bit)
- **mtmsrd -** Move to Machine State Register (64-bit)
- **sc -** System call
- **scv -** System call vectored
- **rfscv -** Return from system call vectored
- **rfid -** Return from interrupt doubleword
- **hrfid -** Hypervisor return from interrupt doubleword

The POWER9 core does not provide support for the following optional or obsolete instructions (attempted use of these results in a hypervisor emulation assistance interrupt):

- **tlbia** - TLB invalidate all
- **tlbiex** - TLB invalidate entry by index (obsolete)
- **slbiex** - SLB invalidate entry by index (obsolete)
- **dcba** - Data cache block allocate (Book II; obsolete)
- **dcbi** - Data cache block invalidate (obsolete)
- **rfi** - Return from interrupt (32-bit; obsolete)

The following instruction variants are implemented:

- **ptesync** - Page table synchronize
- **hwsync** - Heavyweight synchronize
- **lwsync**- Lightweight synchronize

4.7.1.2 Reserved Instructions

The architecture breaks the reserved instruction class down into several categories as described in the *Reserved Instructions* appendix of the Power ISA. The POWER9 processor core behaves in the following manner with respect to these categories:

- The primary opcode of zero is treated as an illegal instruction.
- For the Power Architecture® instructions not in the Power ISA, the POWER9 core takes a hypervisor emulation assistance interrupt. See a complete list in the *Power ISA (Version 3.0B)* appendices.
- The service processor "Attention" instruction is treated as an illegal instruction unless HID[en_attn] = '1'.

In addition, there are several implementation-specific registers available for access through the **mtspr** and **mfspr** instructions. These are described in *Section 4.7.3.4 Move To/From Special Purpose Register Instructions* on page 82.

4.7.2 Branch Processor

4.7.2.1 SRR1 Register

In the POWER9 processor core, the SRR1 is implemented per the Power ISA.

4.7.2.2 HSRR1 Register

In the POWER9 processor core, the HSRR1 is implemented per the Power ISA.

4.7.2.3 MSR Register

In the POWER9 processor core, the MSR is implemented per the Power ISA. All reserved bits should be set to '0' by software.

4.7.2.4 System Call and System Call Vectored Instructions

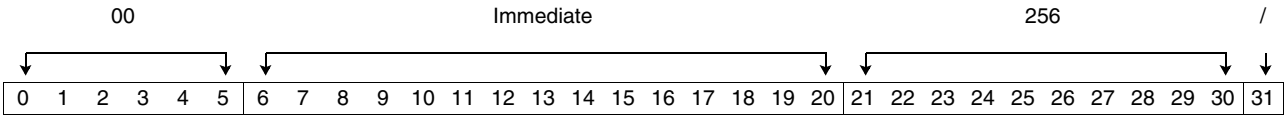
In the POWER9 core, the system call (**sc**) and system call vectored (**scv**) instructions are implemented as described in *Table 4-5* using primary opcode 17.

Table 4-5. System Call and System Call Vectored Invocation

Bits [30:31]	Description
'00'	**sc** instruction
'01'	**scv** instruction
'10'	**sc** instruction
'11'	**sc** instruction

4.7.2.5 Support Processor Attention Instruction

The POWER9 processor core supports a special, implementation-dependent instruction for signaling an attention to the support processor.



The immediate field (I) has no effect on the operation of this instruction in the POWER9 processor core.

If HID[3] = '1' (support-processor attention enable bit is set), this instruction causes all preceding instructions to run to completion, the machine to quiesce, and the assertion of the support processor attention signal. Instruction execution does not resume until the support processor signals it to do so. When setting HID[3] = '1', the I-cache must be flushed by setting HID[2] so that all Attention instructions in the I-cache see the effect of enabling the Attention Enable bit.

If HID[3] = '0' (support-processor attention enable not set), this instruction causes a hypervisor emulation assistance interrupt. Note that due to some design features unique to the POWER9 core, when the hypervisor emulation assistance interrupt occurs, the instruction opcode saved in the Hypervisor Emulation Assistance Interrupt Register (HEIR) is slightly modified and appears as x'001E0200' when read.

4.7.2.6 Current Instruction Address Breakpoint Register (CIABR)

The POWER9 processor core supports the CIAB Register as implemented per the Power ISA.

4.7.3 Fixed-Point Processor

4.7.3.1 Processor Version Register (PVR)

The Process Version Register (PVR) is a 32-bit register that contains the version and revision level information for the POWER9 core. *Table 4-6* summarizes how to interpret the PVR.

Table 4-6. PVR

Bits	Field Name	Description
32:47	Version	Processor version number. The processor version number for the POWER9 core in both Nimbus and Cumulus is x'004E'.
48:51	Chip Type	POWER9 chip scaling factor. x'0' Scale out 12 cores x'1' Scale out 24 cores x'2' Scale up 12 cores x'3' Scale up 24 cores
52:55	Revision (major)	Major revision level. The major processor revision level starts at x'1', indicating major revision '1'. Subsequent major revisions will be x'2', x'3', and so on.
56:59	Revision (reserved)	Read as zero.
60:63	Revision (minor)	Minor revision level. Minor revisions are indicated in PVR[60:63]. Each major revision will reset the minor revision field to x'00' and each minor revision will increment PVR[60:63] by x'01'.

Example: The PVR value for the 12-core POWER9 chip for design revision level 2.0 is: x'004E0200'.

4.7.3.2 Processor ID Register (PIR)

The Processor Identification Register (PIR) is a 64-bit register that holds a processor identification tag in the least-significant bits. This tag is used for tagging bus transactions and for processor differentiation in multiprocessor systems

Table 4-7 shows how to interpret the PIR values.

Table 4-7. PIR

Bits	Field Name	Description
0:48	Reserved	Read as zeros
49:52	Node ID	Node ID.
53:55	ChID	Chip ID
56	Reserved	Read as zero
57:61	CoID	Core number
62:63	TID	Thread ID

The PIR is a read-only register. During power-on reset, PIR is set to a unique value for each processor in a multiprocessor system.

4.7.3.3 Chip Information Register (CIR)

The POWER9 processor implements the CIR per the Power ISA.

4.7.3.4 Move To/From Special Purpose Register Instructions

The POWER9 core supports the SPRs listed in *Table 4-8* on page 83. Many of these SPRs are only accessible in hypervisor or privileged modes. Additionally, some SPRs are only accessible in ultravisor privileged mode when the optional Secure Memory Facility (SMF) is enabled. See *Section 24.3 Secure Memory Facility* on page 326. A handful of these registers (for example, DSCR) are also user-mode accessible through a second SPR number.

To support multithreading, some of the SPRs are replicated in the POWER9 core, while others are shared, as shown in the SMT column in *Table 4-8*. In the table column headers, Prob indicates problem state (S = x, HV = x, PR = 1), Priv indicates privileged state (S = x, HV = 0, PR = 0), Hyp indicates hypervisor state (S = 0, HV = 1, PR = 0) and UV indicates ultravisor state (S =1, HV = 1, PR = 0). In the SPR-specific rows, Priv indicates that a privileged instruction type program interrupt will occur in that state for the attempted read or write of the SPR. Illegal indicates a hypervisor emulation assistance interrupt will occur. NOP indicates the instruction will be treated as a NOP. A blank under each column indicates the access will be performed normally.

Table 4-8. SPR Table (Sheet 1 of 10)

SPR Name	Binary SPR Code	Decimal SPR Code	Thread / LPAR Replica	Length (bits)	Read (MFSPR)				Write (MTSPR)			
					Prob S = x HV = x PR = 1	Priv S = x HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0	Prob S = x HV = x PR = 1	Priv S = X HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0
FPSCR												
VSCR												
MSR												
Special SPR 0	00000 00000	0			Illegal	illegal	illegal	illegal	illegal	illegal	illegal	illegal
XER	00000 00001	1	Replicated per PT	64								
Reserved (MTMSR)	00000 00010	2										
DSCR_RU (FSCR[61]=0)	00000 00011	3	Replicated per PT	25	FAC Unavail Intr				FAC Unavail Intr			
DSCR_RU (FSCR[61]=1)	00000 00011	3	Replicated per PT	25								
Special SPR 4	00000 00100	4			Illegal	illegal	illegal	illegal	illegal	illegal	illegal	illegal
Special SPR 5	00000 00101	5			Illegal	illegal	illegal	illegal	illegal	illegal	illegal	illegal
Special SPR 6	00000 00110	6			Illegal	illegal	illegal	illegal	illegal	illegal	illegal	illegal
LR	00000 01000	8	Replicated per PT	64								
CTR	00000 01001	9	Replicated per PT	64								
UAMR	00000 01101	13	Replicated per PT	64								
DSCR (HFSCR[61] = 0)	00000 10001	17	Replicated per PT	25	Priv	FAC Unavail Intr			Priv	FAC Unavail Intr		
DSCR (HFSCR[61] = 1)	00000 10001	17	Replicated per PT	25	Priv				Priv			
DSISR	00000 10010	18	Replicated per PT	32	Priv				Priv			

Table 4-8. SPR Table (Sheet 2 of 10)

SPR Name	Binary SPR Code	Decimal SPR Code	Thread / LPAR Replica	Length (bits)	Read (MFSPR)				Write (MTSPR)			
					Prob S = x HV = x PR = 1	Priv S = x HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0	Prob S = x HV = x PR = 1	Priv S = X HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0
DAR	00000 10011	19	Replicated per PT	64	Priv				Priv			
DEC	00000 10110	22	Replicated per VT	32	Priv				Priv			
SRR0	00000 11010	26	Replicated per PT	64	Priv				Priv			
SRR1	00000 11011	27	Replicated per PT	64	Priv				Priv			
CFAR	00000 11100	28	Replicated per PT	64	Priv				Priv			
AMR	00000 11101	29	Replicated per PT	64	Priv				Priv			
PIDR	00001 10000	48	Replicated per PT	32	Priv				Priv			
IAMR	00001 11101	61	Replicated per PT	32	Priv				Priv			
Reserved (BHRB)	00010 00000 00010 11111	64 - 95										
TFHAR	00100 00000	128	Replicated per PT	64								
TFIAR	00100 00001	129	Replicated per PT	64								
TEXASR	00100 00010	130	Replicated per PT	64								
TEXASRU	00100 00011	131	Replicated per PT	32								
CTRL_RU	00100 01000	136	Shared/LPAR bit manipula-tion	32	Return Bit 63 Only				Illegal	Nop_ev	Nop_ev	Nop_ev
CTRL	00100 11000	152	Shared	32	Priv	Nop_ev	Nop_ev	Nop_ev	Priv			
FSCR	00100 11001	153	Replicated per PT	64	Priv				Priv			
UAMOR	00100 11101	157	Replicated per PT	64	Priv				Priv			

Table 4-8. SPR Table (Sheet 3 of 10)

SPR Name	Binary SPR Code	Decimal SPR Code	Thread / LPAR Replica	Length (bits)	Read (MFSPR)				Write (MTSPR)			
					Prob S = x HV = x PR = 1	Priv S = x HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0	Prob S = x HV = x PR = 1	Priv S = X HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0
GSR	00100 11110	158			Priv	Nop_ev	Nop_ev	Nop_ev	Priv			
PSPB	00100 11111	159	Replicated per PT	32	Priv				Priv			
DPDES	00101 10000	176	Per Core	8	Priv				Priv	Priv_ev		
DAWR0	00101 10100	180	Replicated per PT	64	Priv	Priv_ev	HEAI with HSSR (45) SMF CTRL (D) = 1		Priv	Priv_ev	HEAI with HSSR (45) SMF CTRL (D) = 1	
RPR	00101 11010	186	Per Core	64	Priv	Priv_ev			Priv	Priv_ev		
CIABR	00101 11011	187	Replicated per PT	64	Priv	Priv_ev	HEAI with HSSR (45) SMF CTRL (D) = 1		Priv	Priv_ev	HEAI with HSSR (45) SMF CTRL (D) = 1	
DAWRX0	00101 11100	188	Replicated per PT	32?	Priv	Priv_ev	HEAI with HSSR (45) SMF CTRL (D) = 1		Priv	Priv_ev	HEAI with HSSR (45) SMF CTRL (D) = 1	
HFSCR	00101 11110	190	Replicated per PT	64	Priv	Priv_ev			Priv	Priv_ev		
VRSAVE	01000 00000	256	Replicated per PT	32								
SPRG3_RU	01000 00011	259	Replicated per PT	64					Illegal	Nop_ ev	Nop_ ev	Nop_ ev
TB	01000 01100	268	Per LPAR VT	64					Illegal	Nop_ ev	Nop_ ev	Nop_ ev
TBU_RU	01000 01101	269	Per LPAR VT	32					Illegal	Nop_ ev	Nop_ ev	Nop_ ev
SPRG0	01000 10000	272	Replicated per PT	64	Priv				Priv			

Table 4-8. SPR Table (Sheet 4 of 10)

SPR Name	Binary SPR Code	Decimal SPR Code	Thread / LPAR Replica	Length (bits)	Read (MFSPR)				Write (MTSPR)			
					Prob S = x HV = x PR = 1	Priv S = x HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0	Prob S = x HV = x PR = 1	Priv S = X HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0
SPRG1	01000 10001	273	Replicated per PT	64	Priv				Priv			
SPRG2	01000 10010	274	Replicated per PT	64	Priv				Priv			
SPRG3	01000 10011	275	Replicated per PT	64	Priv				Priv			
SPRC	01000 10100	276	Replicated per VT	64	Priv	Priv_ ev			Priv	Priv_ ev		
SPRD	01000 10101	277	n/a (physi-cal target controlled by SPRC)	64	Priv	Priv_ ev			Priv	Priv_ ev		
CIR	01000 11011	283	Shared	32	Priv				Priv	NOP	NOP	NOP
TBL	01000 11100	284	Per LPAR VT	32	Priv	Nop_ ev	Nop_ ev	Nop_ ev	Priv	Priv_ ev		
TBU	01000 11101	285	Per LPAR VT	32	Priv	Nop_ ev	Nop_ ev	Nop_ ev	Priv	Priv_ ev		
TBU40	01000 11110	286	Per LPAR VT	64	Priv	Nop_ev	Nop_ev	Nop_ev	Priv	Priv_ev		
PVR	01000 11111	287	Shared	32	Priv				Priv	Nop_ev	Nop_ev	Nop_ev
HSPRG0	01001 10000	304	Replicated per PT	64	Priv	Priv_ev			Priv	Priv_ev		
HSPRG1	01001 10001	305	Replicated per PT	64	Priv	Priv_ev			Priv	Priv_ev		
HDSISR	01001 10010	306	Replicated per PT	32	Priv	Priv_ev			Priv	Priv_ev		
HDAR	01001 10011	307	Replicated per PT	64	Priv	Priv_ev			Priv	Priv_ev		
SPURR	01001 10100	308	Replicated per VT	64	Priv				Priv	Priv_ev		
PURR	01001 10101	309	Replicated per VT	64	Priv				Priv	Priv_ev		
HDEC	01001 10110	310	Per LPAR VT	32	Priv	Priv_ev			Priv	Priv_ev		
HRMOR	01001 11001	313	Shared	64	Priv	Priv_ev			Priv	Priv_ev		

Table 4-8. SPR Table (Sheet 5 of 10)

SPR Name	Binary SPR Code	Decimal SPR Code	Thread / LPAR Replica	Length (bits)	Read (MFSPR)				Write (MTSPR)			
					Prob S = x HV = x PR = 1	Priv S = x HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0	Prob S = x HV = x PR = 1	Priv S = X HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0
HSRR0	01001 11010	314	Replicated per PT	64	Priv	Priv_ev			Priv	Priv_ev		
HSRR1	01001 11011	315	Replicated per PT	64	Priv	Priv_ev			Priv	Priv_ev		
TFMR	01001 11101	317	Shared (partial)/ LPAR bits 26 and 45 replicated	64	Priv	Priv_ev			Priv	Priv_ev		
LPCR	01001 11110	318	Replicated per PT	64	Priv	Priv_ev			Priv	Priv_ev		
LPIDR	01001 11111	319	Replicated per PT	64	Priv	Priv_ev			Priv	Priv_ev		
HMER	01010 10000	336	Replicated per VT	64	Priv	Priv_ev			Priv	Priv_ev		
HMEER	01010 10001	337	Shared	64	Priv	Priv_ev			Priv	Priv_ev		
PCR	01010 10010	338	Per LPAR PT	64	Priv	Priv_ev			Priv	Priv_ev		
HEIR	01010 10011	339	Replicated per PT	32	Priv	Priv_ev			Priv	Priv_ev		
AMOR	01010 11101	349	Per LPAR PT	64	Priv	Priv_ev			Priv	Priv_ev		
TIR	01101 11110	446	Replicated per PT	8	Priv				Priv	Nop_ev	Nop_ev	Nop_ev
Reserved (PC internal)		447-463										
PTCR	01110 10000	464	Per Core	64	Priv	Priv_ev			Priv	Priv_ev	HEAI w HSSR1 (45) SMFC-TRL(E) =1	
Reserved (PC internal)		465-475										
Reserved (msgclr)	01110 11100	476										
Reserved (msgclrp)	01110 11101	477										

Table 4-8. SPR Table (Sheet 6 of 10)

SPR Name	Binary SPR Code	Decimal SPR Code	Thread / LPAR Replica	Length (bits)	Read (MFSPR)				Write (MTSPR)			
					Prob S = x HV = x PR = 1	Priv S = x HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0	Prob S = x HV = x PR = 1	Priv S = X HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0
Reserved (msgsndp)	01110 11110	478										
Reserved (msgclru)	01110 11111	479										
USPRG0	01111 10000	496	Replicated per PT	64	Priv	Priv	Priv		Priv	Priv	Priv	
USPRG1	01111 10001	497	Replicated per PT	64	Priv	Priv	Priv		Priv	Priv	Priv	
URMOR	01111 11001	505	Replicated per PT	64	Priv	Priv	Priv		Priv	Priv	Priv	
USRR0	01111 11010	506	Replicated per PT	64	Priv	Priv	Priv		Priv	Priv	Priv	
USRR1	01111 11011	507	Replicated per PT	64	Priv	Priv	Priv		Priv	Priv	Priv	
SMFCTRL	01111 11111	511	Replicated per PT	64	Priv	Priv	Priv		Priv	Priv	Priv	
SIER_RU	11000 00000	768	Replicated per PT	64					Illegal	Nop	Nop	Nop
MMCR2_RU	11000 00001	769	Replicated per PT	64					Illegal	Nop	Nop	Nop
MMCRA_RU	11000 00010	770	Replicated per PT	64					Illegal	Nop	Nop	Nop
PMC1_RU	11000 00011	771	Replicated per PT	32					Illegal	Nop	Nop	Nop
PMC2_RU	11000 00100	772	Replicated per PT	32					Illegal	Nop	Nop	Nop
PMC3_RU	11000 00101	773	Replicated per PT	32					Illegal	Nop	Nop	Nop
PMC4_RU	11000 00110	774	Replicated per PT	32					Illegal	Nop	Nop	Nop
PMC5_RU	11000 00111	775	Replicated per PT	32					Illegal	Nop	Nop	Nop
PMC6_RU	11000 01000	776	Replicated per PT	32					Illegal	Nop	Nop	Nop
MMCR0_RU	11000 01011	779	Replicated per PT	32					Illegal	Nop	Nop	Nop
SIAR_RU	11000 01100	780	Replicated per PT	64					Illegal	Nop	Nop	Nop

Table 4-8. SPR Table (Sheet 7 of 10)

SPR Name	Binary SPR Code	Decimal SPR Code	Thread / LPAR Replica	Length (bits)	Read (MFSPR)				Write (MTSPR)			
					Prob S = x HV = x PR = 1	Priv S = x HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0	Prob S = x HV = x PR = 1	Priv S = X HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0
SDAR_RU	11000 01101	781	Replicated per PT	64					Illegal	Nop	Nop	Nop
MMCR1_RU	11000 01110	782	Replicated per PT	32					Illegal	Nop	Nop	Nop
SIER	11000 10000	784	Replicated per PT	64	Priv				Priv			
MMCR2	11000 10001	785	Replicated per PT	64	Priv				Priv			
MMCRA	11000 10010	786	Replicated per PT	64	Priv				Priv			
PMC1	11000 10011	787	Replicated per PT	32	Priv				Priv			
PMC2	11000 10100	788	Replicated per PT	32	Priv				Priv			
PMC3	11000 10101	789	Replicated per PT	32	Priv				Priv			
PMC4	11000 10110	790	Replicated per PT	32	Priv				Priv			
PMC5	11000 10111	791	Replicated per PT	32	Priv				Priv			
PMC6	11000 11000	792	Replicated per PT	32	Priv				Priv			
MMCR0	11000 11011	795	Replicated per PT	32	Priv				Priv			
SIAR	11000 11100	796	Replicated per PT	64	Priv				Priv			
SDAR	11000 11101	797	Replicated per PT	64	Priv				Priv			
MMCR1	11000 11110	798	Replicated per PT	32	Priv				Priv			
IMC	11000 11111	799	Shared	64	Priv	Priv_ev			Priv	Priv_ev	HEAI with HSSR1 (45) SMF CTRL (E) = 1	

Table 4-8. SPR Table (Sheet 8 of 10)

SPR Name	Binary SPR Code	Decimal SPR Code	Thread / LPAR Replica	Length (bits)	Read (MFSPR)				Write (MTSPR)			
					Prob S = x HV = x PR = 1	Priv S = x HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0	Prob S = x HV = x PR = 1	Priv S = X HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0
BESCRS	11001 00000	800	Replicated per PT	64								
BESCRSU	11001 00001	801	Replicated per PT	32								
BESCRR	11001 00010	802	Replicated per PT	64								
BESCRRU	11001 00011	803	Replicated per PT	32								
EBBHR	11001 00100	804	Replicated per PT	64								
EBBRR	11001 00101	805	Replicated per PT	64								
BESCR	11001 00110	806	Replicated per PT	64								
Reserved	11001 01000	808	NA		Nop	Nop	Nop	Nop	Nop	Nop	Nop	Nop
Reserved	11001 01001	809	NA		Nop	Nop	Nop	Nop	Nop	Nop	Nop	Nop
Reserved	11001 01010	810	NA		Nop	Nop	Nop	Nop	Nop	Nop	Nop	Nop
Reserved	11001 01011	811	NA		Nop	Nop	Nop	Nop	Nop	Nop	Nop	Nop
TAR	11001 01111	815	Replicated per PT	64								
ASDR	11001 10000	816	Replicated per PT	64	Priv	Priv_ev			Priv	Priv_ev		
PSSCR_SU	11001 10111	823	Replicated per PT	64	Priv				Priv			
Reserved (MTXER)	11001 11001	825										
Reserved (MFNIA)	11001 11010	826										
IC	11010 10000	848	Replicated per PT	64	Priv				Priv	Priv_ev		
VTB	11010 10001	849	Per LPAR VT	64	Priv				Priv	Priv_ev		

Table 4-8. SPR Table (Sheet 9 of 10)

SPR Name	Binary SPR Code	Decimal SPR Code	Thread / LPAR Replica	Length (bits)	Read (MFSPR)				Write (MTSPR)			
					Prob S = x HV = x PR = 1	Priv S = x HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0	Prob S = x HV = x PR = 1	Priv S = X HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0
LDBAR	11010 10010	850	Per LPAR VT	64	Priv	Priv_ev			Priv	Priv_ev	HEAI with HSSR1 (45) SMF CTRL (E) = 1	
MMCRC	11010 10011	851	Shared	32	Priv	Priv_ev			Priv	Priv_ev		
PMSR	11010 10101	853	Shared	32	Priv	Priv_ev			Priv	Nop_ev	Nop_ev	Nop_ev
PSSCR	11010 10111	855	Replicated per VT	64	Priv	Priv_ev			Priv	Priv_ev		
L2QOSR	11010 11101	861	Per Core		Priv	Nop_ev	Nop_ev	Nop_ev	Priv	Priv_ev		
TRIG0	11011 10000	880	Replicated per PT	64	Priv	Nop_ev	Nop_ev	Nop_ev	Priv			
TRIG1	11011 10001	881	Replicated per PT	64	Priv	Nop_ev	Nop_ev	Nop_ev	Priv			
TRIG2	11011 10010	882	Replicated per PT	64	Priv	Nop_ev	Nop_ev	Nop_ev	Priv			
PMCR	11011 10100	884	Per Core	64	Priv	Priv_ev			Priv	Priv_ev		
RWMR	11011 10101	885	Shared	64	Priv	Priv_ev			Priv	Priv_ev		
WORT	11011 11111	895	Replicated per PT	18	Priv				Priv			
PPR	11100 00000	896	Replicated per PT	64								
PPR32	11100 00010	898	Replicated per PT	32								
TSCR	11100 11001	921	Shared	32	Priv	Priv_ev			Priv	Priv_ev		
TTR	11100 11010	922	Shared	64	Priv	Priv_ev			Priv	Priv_ev		
TRACE	11111 01110	1006	Shared	64	Illegal	Nop_ev	Nop_ev	Nop_ev				
HID	11111 10000	1008	Shared	64	Priv	Priv_ev			Priv	Priv_ev		

Table 4-8. SPR Table (Sheet 10 of 10)

SPR Name	Binary SPR Code	Decimal SPR Code	Thread / LPAR Replica	Length (bits)	Read (MFSPR)				Write (MTSPR)			
					Prob S = x HV = x PR = 1	Priv S = x HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0	Prob S = x HV = x PR = 1	Priv S = X HV = 0 PR = 0	Hyp S = 0 HV = 1 PR = 0	UV S = 1 HV = 1 PR = 0
PIR	11111 11111	1023	Replicated per VT	32	Priv				Priv	Nop_ev	Nop_ev	Nop_ev
Unsupported SPRs w/ SPR(0)=0	xxxxx 0xxxx				Illegal	Nop_ev	Nop_ev	Nop_ev	Illegal	Nop_ev	Nop_ev	Nop_ev
Unsupported SPRs w/ SPR(0)=1	xxxxx 1xxxx				Priv	Nop_ev	Nop_ev	Nop_ev	Priv	Nop_ev	Nop_ev	Nop_ev

4.7.3.5 SPRC/SPRD Usage

In the pervasive design, SPRC and SPRD are used as a pair to provide a programmable interface into microarchitected SPRs using indirect addressing. The address of the SPR to access is written into SPRC and then a following read or write to SPRD reads or writes the SPR addressed by SPRC. The SPRC and SPRD can both be accessed using an **mtspr**/**mfspr** instruction or SCOM. An SPRC/SPRD pair is dedicated per thread for SPR access and per core for SCOM access. Only 8 bits of the SPRC register [53:60] are implemented, bits [61:63] are reserved for future use and must be set to '000'. SPRD is a full 64-bit register whose meaning changes based on the contents of SPRC. This register is, in reality, an alias to microarchitected hypervisor resources.

For **mtsprc** access, the thread determines which thread-specific SPRC to set. For SPRD access, each thread-specific SPRC can be used to access the SPRs in normal-core mode. In normal core mode, SPRC/SPRD can only access the logical thread of the requesting thread as shown in *Table 4-9 SPRC Definition Normal Core Mode (1 LPAR per Thread)* on page 92. The thread-specific SPRC can also be set via SCOM using the SPR_MODE_REG.

For OCC SCOM access, a dedicated SPRC/SPRD only accesses the activity counters and auto-increments as shown in *Table 4-11* on page 93. SCOM (x'0A83') is used to read the data addressed by this SPRC (x'0A82').

In *Table 4-9*, only core TFMR bits [0:56] can be accessed using an SCOM access.

Table 4-9. SPRC Definition Normal Core Mode (1 LPAR per Thread) (Sheet 1 of 2)

SPRC[54:60]							SPRD Selection	Notes
0	0	0	0	0	S	S	SCRATCH0 - 3; unique per core chiplet	1, 3
0	0	0	0	1	S	S	SCRATCH4 - 7	1, 3
0	0	0	1	0	0	0	TFMR (logical thread)	1

1. Registers can be accessed by both an **mtspr**/**mfspr** instruction and an SCOM. Only core TFMR bits [0:56] can be accessed using an SCOM access.
2. Registers can only be accessed using **mtspr**/**mfspr**.
3. S = scratch register number; T = Thread; e = Empath counter number.

Table 4-9. SPRC Definition Normal Core Mode (1 LPAR per Thread) (Sheet 2 of 2)

SPRC[54:60]							SPRD Selection	Notes
0	0	1	0	0	0	0	PURR (logical thread)	2
0	0	1	1	0	0	0	SPURR (logical thread)	2
0	1	0	0	0	0	0	DEC (logical thread)	2
0	1	1	1	0	0	0	SPR_MODEREG SPR	2
0	1	1	1	0	0	1	AVP output pin	2
0	1	1	1	0	1	0	Core checkstop	2
0	1	1	1	0	1	1	SPATTN	2
0	1	1	1	1	0	0	Core Thread State	1
1	0	e	e	e	e	e	Empath counters; always do autoincremen	2, 3

1. Registers can be accessed by both an **mtspr**/**mfspr** instruction and an SCOM. Only core TFMR bits [0:56] can be accessed using an SCOM access.
2. Registers can only be accessed using **mtspr**/**mfspr**.
3. S = scratch register number; T = Thread; e = Empath counter number.

In *Table 4-10*, only core TFMR bits [0:56] can be accessed via an SCOM access.

Table 4-10. SPRC Definition Normal Core Mode (1 LPAR per Core)

SPRC[54:60]							SPRD Selection	Notes
0	0	0	0	0	S	S	SCRATCH0 - 3; unique per core chiplet	1, 3
0	0	0	0	1	S	S	SCRATCH4 - 7	1, 3
0	0	0	1	0	T	T	TFMR (logical thread)	1, 3
0	0	1	0	0	T	T	PURR (logical thread)	2, 3
0	0	1	1	0	T	T	SPURR (logical thread)	2, 3
0	1	0	0	0	T	T	DEC (logical thread)	2, 3
0	1	1	1	0	0	0	SPR_MODEREG SPR	2
0	1	1	1	0	0	1	AVP output pin	2
0	1	1	1	0	1	0	Core checkstop	2
0	1	1	1	0	1	1	SPATTN	2
0	1	1	1	1	0	0	Core Thread state	1
1	0	e	e	e	e	e	Empath counters; always do auto-increment	2, 3

1. Registers can be accessed by both an **mtspr**/**mfspr** instruction and an SCOM. Only core TFMR bits [0:56] can be accessed using an SCOM access.
2. Registers can only be accessed using **mtspr**/**mfspr**.
3. S = scratch register number; T = Thread; e = Empath counter number.

Table 4-11. OCC SPRC Definition

SPRC[54:60]							SPRD Selection
1	c	e	e	e	e	e	Empath counters (e); always do auto-increment

Note: c = select core chiplets; e = empath counter.

4.8 HID Register

The POWER9 processor core includes several implementation-dependent mode bits that allow various features of the chip to be enabled and disabled. These bits are included in the Hardware Implementation Dependent Register (HID). In general, the HID Register controls high-level functions of the POWER9 core and is only accessible in hypervisor mode. Reserved bits in the HID Register should not be set by software and can return either a zero or one value depending on the bit if set. Attempts to set some of these bits might enable functions that are no longer supported and thus could cause unpredictable behavior. Two values of the contents of the register are shown in the descriptions.

Initial state:
This is the state of the register after a normal scan-based power-on-reset (POR). The actual and full POR sequence can set bits beyond the scan-based POR.

Preferred state:
This is the preferred state of the register for optimal performance and function.

1. The following sequence must be used when modifying the HID Register:
   sync
   mtspr HID,Rx
   isync

4.8.1 HID Register Description

Initial state: x'0400_0000_0000_0000'
Preferred state: x'0000_0000_0000_0000'

Table 4-12 describes the HID Register.

Table 4-12. HID Register (Sheet 1 of 2)

Bits	Field Name	Description
0	one_ppc	One (Power ISA) instruction is sent out of the ibuffers and decoded at a time. The IFU waits for ict empty to let the next instruction go. Multi-thread mode uses a round-robin method through the enabled threads.
1	en_instruc_trace	Enable the enhanced trace facility, which requires special hardware initializations.
2	flush_ic	Flush the instruction cache and the instruction EADIR on a transition from '0' to '1'.
3	en_attn	Enable the support-processor attention instruction. This bit is used to enable the **attn** instruction to quiesce the thread. **Note:** The instruction cache must be flushed after changing the value of this bit.
4	hile	The contents of this bit are copied into the MSR.
5	dis_recovery	Disable the processor recovery mechanism.
6	megamouth	Order stores for the Megamouth adapter when IG = '10'.
7	prefetch_reset	Clear all streams from the prefetch unit and restart from an idle state.
8	tlb_config_radix	TLB configuration mode. 0 Supports HPT translation only. PWC is disabled. 1 Supports either HPT or Radix translation. PWC is enabled.
9	dcache_partitioned	The D-cache is partitioned by the thread in a balanced method, such that each active thread can use an equal portion of the cache.

Table 4-12. HID Register (Sheet 2 of 2)

Bits	Field Name	Description
10	icache_partitioned	The I-cache is partitioned by the thread in a balanced method, such that each active thread can use an equal portion of the cache.
11	en_spec_execution	Enable speculative execution mode.
12	spare	Spare.
13:63	reserved	Reserved.

4.8.2 Core-to-Core Trace SPR

The Trace SPR is used to access enhanced instruction trace information from the processor core trace logic. This 64-bit register is read only and has a privileged read access. There is a protocol associated with the use of this register to coordinate gathering instruction trace images from the other processor core.

4.8.3 Trigger Registers

Writes to the trigger registers (TRIG0, TRIG1, and TRIG2), can be inserted in the instruction stream to cause triggers to the on-chip trace array debug logic. These registers are used for lab debug and bringup only and architecturally behave as a NOP.

4.8.4 IMC Array Access Register

The Instruction Match CAM (IMC) array facility is used for performance monitoring instrumentation and for the soft patch of instructions. (This latter use is restricted for the support processor and is not available through the SPR access to this register array.) The array has privileged write access and user-level read access via this SPR. Writes to the register array are used to configure the IMC, and reads return information about the availability of registers within the facility.

4.8.5 Performance Monitor Registers

The performance monitor counter registers (PMC1 - PMC6), the performance monitor control registers (MMCR0, MMCR1, MMCRA), and the sampled address registers (SIAR, SDAR) are supported in the POWER9 processor core. The performance monitor counter registers PMC7 and PMC8 are not implemented in the POWER9 processor core (an operation for these two performance counter registers is treated as a NOP).

4.8.6 Other Fixed-Point Instructions

The POWER9 processor core supports both the 32-bit **mtmsr** instruction and the 64-bit **mtmsrd** instruction.

The POWER9 processor core optimizes the **mtmsr** and **mtmsrd** instructions by helping to speed up the cases where little or no synchronization is required (such as, updates to the EE and RI bits). To exploit this capability, software should set the L-bit of the desired instruction to '1' as described in the *Power ISA Operating Environment Architecture - Book III (version 3.0B)*.

Software must avoid placing **mtmsr** and **mtmsrd** instructions that change the SF bit at address x'00000000FFFFFFFC' or x'FFFFFFFFFFFFFFFC'.

4.9 Storage Control

4.9.1 Effective, Virtual, and Physical Address Ranges Supported

The POWER9 processor core supports a 64-bit effective address (EA), 68-bit virtual address (VA), and a 56-bit host real (physical) address (RA). See *Section 4.10.31 Processor Compatibility Mode* on page 138 for details specific to various translation modes. Host real addresses in this 56-bit range are hereafter referred to as local address space accesses and are considered to be system-wide coherent.

4.9.2 Foreign Address Space Definition and Accessibility

The POWER9 processor core considers host real addresses with a nonzero value in RA(8:12) as foreign address space, accessible only by the **paste** instruction. By specifying a host real address in this range for the **paste** instruction, a **copy** and **paste** instruction pair can be used to invoke the Nest accelerator via the Virtual Accelerator Switchboard (VAS). For more details on invoking the Nest accelerators, see *Section 11.1 Features* on page 196. A host real address containing a zero value in RA(8:12) is referred to as local address space.

Attempts to access host real pages in this addressing range by an instruction fetch or any other data access (that is, other than a **paste** instruction) results in a machine check interrupt per the Power ISA.

As described in the *Power ISA (Version 3.0B)*, the **copy** instruction can copy a 128-byte cache line (block) from a local address to a per thread noncoherent buffer. Similarly, a **paste** instruction reads the noncoherent buffer and writes the contents of the buffer to a foreign address.

For more details on the **copy** and **paste** instructions, see the *Power ISA (Version 3.0B)*.

4.9.3 Hypervisor Real Mode Addressing Using HRMOR

The POWER9 processor supports the Hypervisor Real Mode Offset Register (HRMOR) as described in the *Power ISA (Version 3.0B)* for the purpose of accessing real memory when the processor thread is operating in Hypervisor Real Mode. As described in the *Power ISA Operating Environment Architecture - Book III (version 3.0B)*, when EA(0) = '1', the HRMOR is bypassed. When EA(0) = '0', the HRMOR value is logically OR'ed with the EA to produce the real address (RA). The ISA defines the number of implemented bits in the HRMOR as implementation dependent. The POWER9 processor implements HRMOR[13:42]. All other HRMOR bits are reserved and return zero when read.

4.9.4 Partition Table Control Register

The POWER9 processor supports the Partition Table Control Register (PTCR) mostly as described in the *Power ISA (Version 3.0B)* for the purpose of accessing virtual memory either in virtual real mode (HPT) and guest real mode (nested Radix) or when translation is enabled for either HPT or Radix. The PTCR implements bits 13:51 for the Partition Table Base (PATB) field and bits 59:63 for the Partition Table Size (PATS) field. All other PTCR bits are reserved and return zero when read. The POWER9 processor core, however, ignores the value in the PATS field and only supports a 64 KB partition table size.

4.9.5 Access Segment Descriptor Register

The POWER9 processor supports the Access Segment Descriptor Register (ASDR) as described in the *Power ISA (Version 3.0B)*.

4.9.6 Real Mode Addressing for Operating Systems

The POWER9 processor does not implement the real mode offset (RMOR/RMLS) mechanism described in the *Power ISA (version 2.07)*. Instead, it implements the virtual real mode addressing mechanism for HPT translation and guest real mode for Radix translation as described in the *Power ISA (Version 3.0B)*.

4.9.7 HRMOR Update Sequence

Table 4-13 describes a sequence that the hypervisor privileged software might use to update the HRMOR.

Table 4-13. HRMOR Update Sequence

Master	Slave
Thread sync up point 1	Thread sync up point 1
EA[0] = 1	EA[0] = 1
Thread sync up point 2	Thread sync up point 2
Change HRMOR	
Thread sync up point 3	Thread sync up point 3
isync	isync
slbia IH = x'7'	slbia IH = x'7'
isync	isync
Thread sync up point 4	Thread sync up point 4

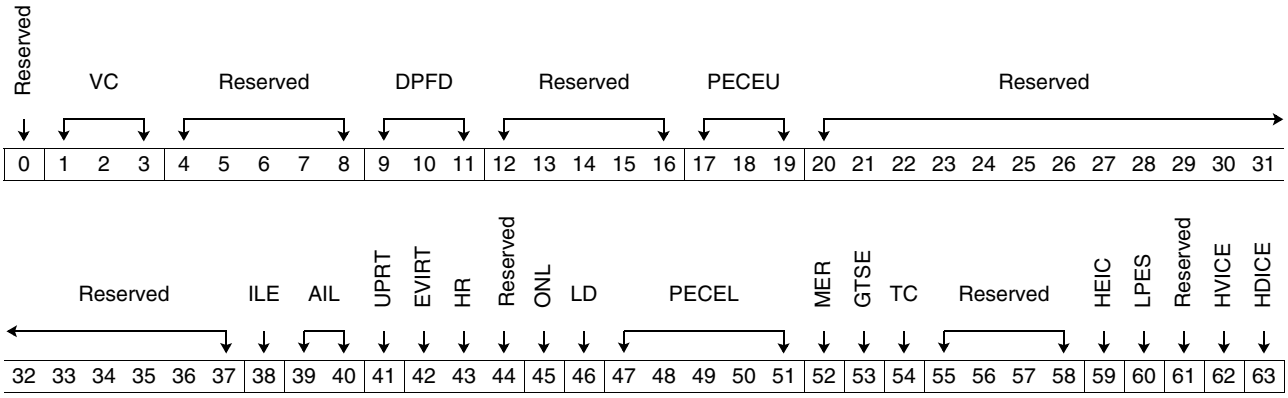
4.10 Translation Architecture

The POWER9 processor supports the two translation mechanisms described in the *Power ISA (Version 3.0B)*, specifically:

- The POWER9 processor uses the Partition Table Control Register (PTCR) to find the partition table entry.
- The LSU allows writes to LPCR[12:16], LPCR[41], and LPCR[53]. Bits 12:16 are considered reserved in the new architecture and always return '00000' when read using the **mflpcr** instruction. The page size information for virtual real mode is taken from the partition table entry as described in the *Power ISA (Version 3.0B)*. The hardware ignores the value of bits 12:16. Bit 41 is the UPRT value and bit 53 is the GTSE value. The **mflpcr** instruction returns the values in bits 41 and 53.
- When LPCR[GTSE] = '0', **slbiag**, **slbieg, slbsync**, and **tlbsync** are hypervisor privileged only and take a privileged instruction interrupt when HV = '0' (independent of PR); or when HV = '1' and PR = '1'.
- When LPCR[GTSE] = '1', **slbiag**, **slbieg, slbsync**, and **tlbsync** are legal instructions when PR = '0' and take a privileged instruction interrupt when PR = '1'.
- The **tlbie** instruction is privileged except when LPCR[GTSE] = '0' or when PRS = '0' and R = '1', making it hypervisor privileged. Note that the POWER9 processor uses the "R" bit in the instruction instead of the "HR" bit in the partition-table entry, as described in the *Power ISA (Version 3.0B)* for determining the instruction's privilege level.
- The **tlbiel** instruction is privileged except when PRS = '0' and R = '1', making it hypervisor privileged. Note that the POWER9 processor uses the "R" bit in the instruction instead of the "HR" bit in the partition-table entry as described in the *Power ISA (Version 3.0B)* for determining the instruction's privilege level.

4.10.1 Logical Partitioning Control Register (LPCR)

The POWER9 LPCR Register is illustrated as follows:



Bits	Field Name	Description
0	Reserved	Reserved.
1:3	VC	Virtualization control.
4:8	Reserved	Reserved.
9:11	DPFD	Default prefetch depth.
12:16	Reserved	Reserved.
17:19	PECEU	Power-saving mode exit causes enable upper section.
20:37	Reserved	Reserved.
38	ILE	Interrupt little-endian mode.
39:40	AIL	Alternate interrupt location.
41	UPRT	Use process table Set this bit to '0' when the thread is performing HPT translation and set to '1' when the thread is performing Radix translation.
42	EVIRT	Enhanced virtualization enable.
43	HR	Host Radix.
44	Reserved	Reserved.
45	ONL	Online (PURR/SPURR incrementing control).
46	LD	Large decrementer.
47:51	PECEL	Power-saving mode exit causes enable lower section.
52	MER	Mediated external exception request.
53	GTSE	Guest translation shoot-down enable.
54	TC	Translation control secondary PTEG is not searched if TC = '1'.
55:58	Reserved	Reserved.
59	HEIC	Hypervisor external interrupt control.
60	LPES	Logical partitioning environment selector.
61	Reserved	Reserved.
62	HVICE	Hypervisor virtualization interrupt conditionally enable.

Bits	Field Name	Description
63	HDICE	Hypervisor decrementer interrupt conditionally enable.

Note: All fields except the AIL, EVIRT, ONL, HDICE, MER, PECE, HEIC, and HVICE fields must be set to the same value by all sub-processors with the same LPIDR value.

4.10.2 Translation Modes

Within the translation architecture described in the *Power ISA (Version 3.0B)*, there are two types of address translation:

- Radix: A Radix guest (operating system) running on top of a Radix host (hypervisor) is also commonly referred to as nested Radix. When there is no guest operating system, this is referred to as single-level Radix.
- Hashed Page Table (HPT): Also, known as paravirtualized translation. For HPT translation, the processor core supports only the behavior specified when the Logical Partition Control Register (LPCR) Use Process Table (UPRT) bit is set to '0'.

In these modes, the Partition Table Control Register (PTCR) contains the host real address of the partition-table base and the size of the table itself. In general, the partition table is indexed by the logical partition ID (LPID) value specified in the Logical Partition ID Register (LPIDR).

When the partition-table entry is read, the host Radix (HR) bit determines which translation type is used by the hardware to convert an effective address to a host real address. When either single-level or nested Radix is used for translation, HR = '1'. When HPT translation is used, HR ='0'. For either of the these translation types, there exists a partition-scoped page table that translates a host virtual address (hVA) (referred to as a guest real address or gRA for Radix) to a host real address (hRA).

For either of the translation types, the in-memory translation related tables managed by the operating system (guest) are translated as though they reside in "normal" memory (such as, ATT = '00' or WIMG = '0010'), regardless of the storage attributes specified in the partition scoped page-table entries used to translate those guest tables. These guest-managed translation tables include the process table for either Radix or HPT, the guest Radix tree for Radix, and the segment table for HPT (for the in-memory segment table supported by the Nest).

4.10.3 tlbie and tlbiel Instruction Format and Operands

When software changes a translation path that involves either a Radix or HPT page table, either the **tlbie** or **tlbiel** instruction must be used in a manner as specified in the *Power ISA (Version 3.0B)*. The ISA provides the format for both the **tlbie** and **tlbiel** instructions, but the AP and L/LP encodings are implementation specific. The format and operands are shown in *Figure 4-2* on page 100 for the **tlbie** instruction and *Figure 4-4* on page 101 for the **tlbiel** instruction.

*Figure 4-2. **tlbie** Instruction Format for the POWER9 Core*

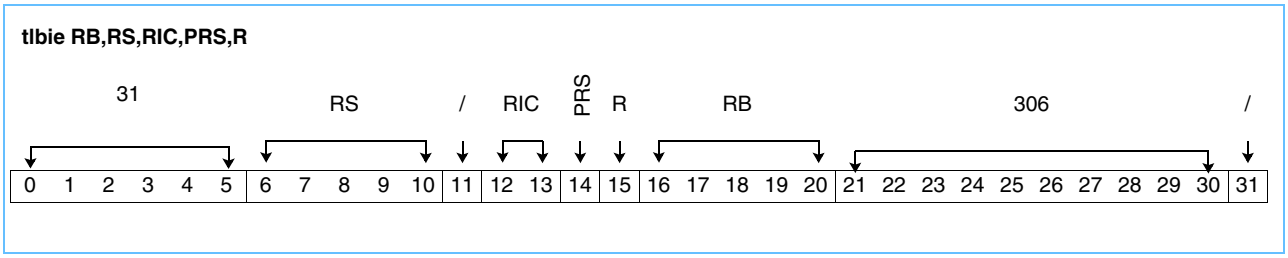


*Table 4-14. Description of **tlbie** Instruction Format for the POWER9 Core*

Bits	Description
RIC	Radix invalidation control. 0 Only invalidate the TLB and ERATs. 1 Invalidate only the page-walk cache (PWC). 2 Invalidate TLB/ERATs, PWC, and any caching of partition and process table entries. 3 Invalidate a series of consecutive translations (only in TLB/ERATs, cluster bomb).
PRS	Process scoped. 0 Invalidate partition-scoped translations. 1 Invalidate process-scoped translations.
R	Radix. 0 Invalidate HPT translations. 1 Invalidate Radix tree translations.
IS	Invalidation selector (specified in the RB Register). 0 Invalidate only the target VA for matching PID and LPID. 1 Invalidate matching PID (and matching LPID). 2 Invalidate matching LPID. 3 If MSR[HV] = '1', invalidate all entries; otherwise, invalidate matching LPID.

The format and operands for the **tlbie** instructions are indicated in *Figure 4-3*. The R and RIC values determine the format of the RB operand.

*Figure 4-3. **tlbie** Operands for the POWER9 Core*

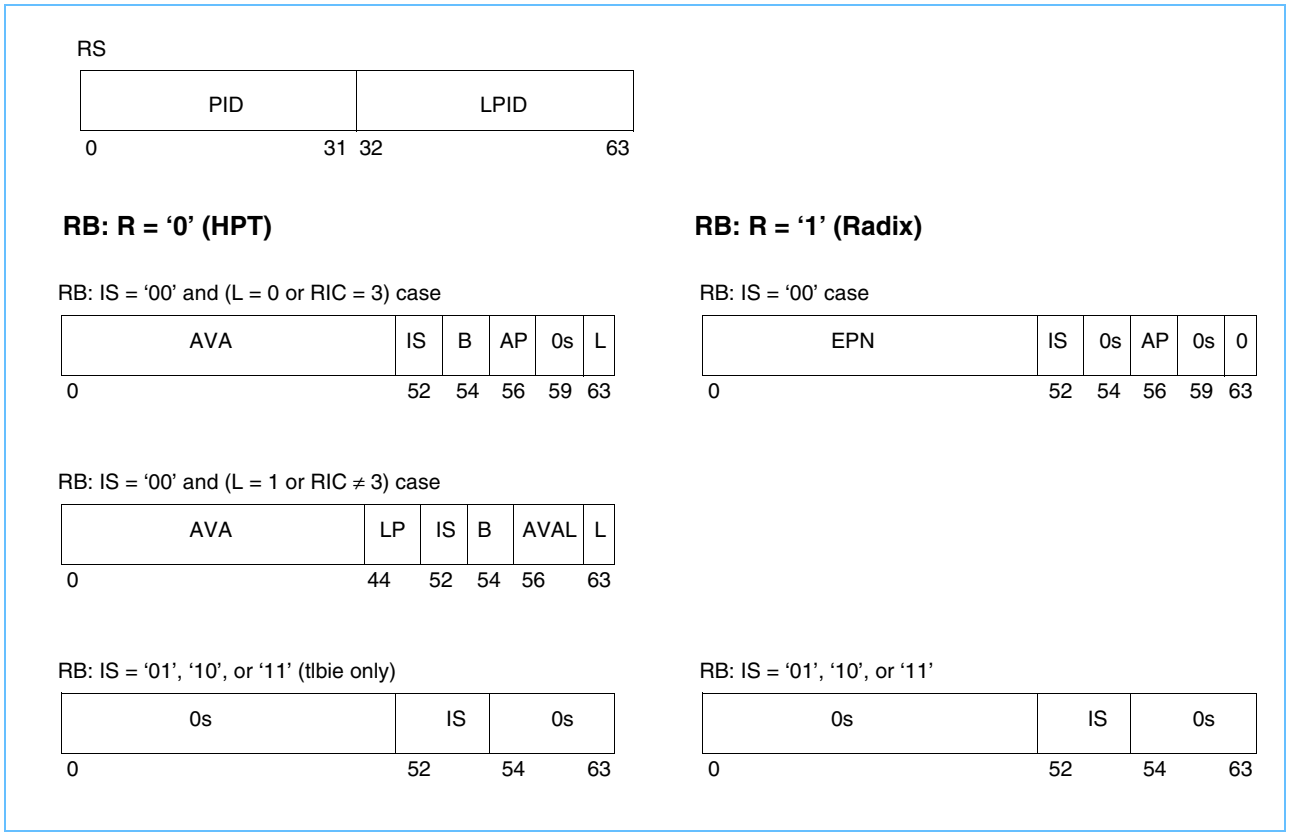


*Figure 4-4. **tlbiel** Instruction Format for the POWER9 Core*

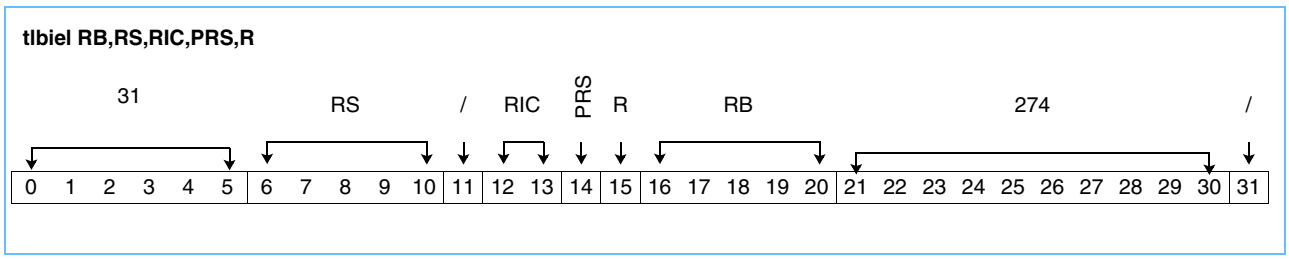
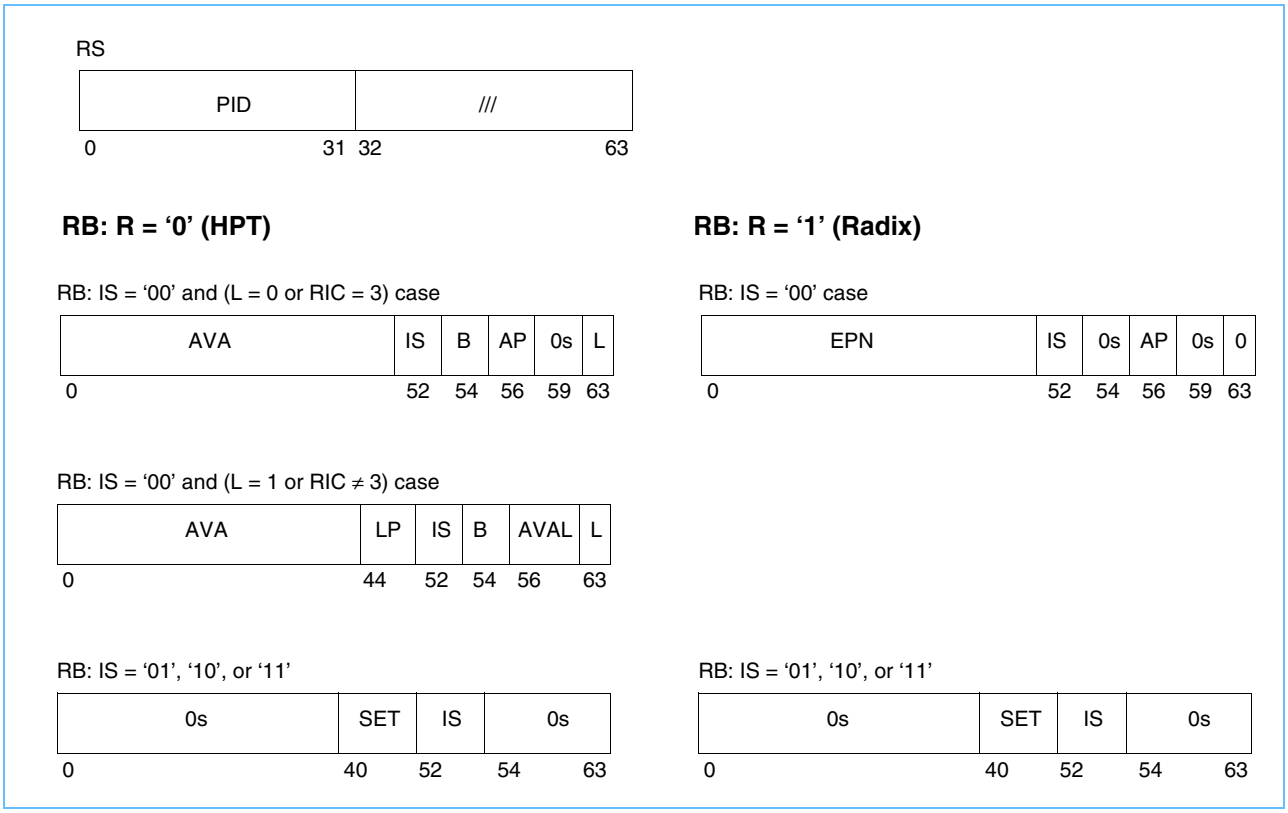


*Table 4-15. Description of **tlbiel** Instruction Format for the POWER9 Core*

Bits	Description
RIC	Radix invalidation control. 0 Only invalidate TLB and ERATs. 1 Invalidate only page-walk cache (PWC). 2 Invalidate TLB/ERATs, PWC, and any caching of partition and process table entries. 3 Not supported.
PRS	Process scoped. 0 Invalidate partition-scoped translations. 1 Invalidate process-scoped translations.
R	Radix. 0 Invalidate HPT translations. 1 Invalidate Radix tree translations.
IS	Invalidation selector (specified in the RB Register). 0 Invalidate only the target VA for matching PID and LPID. 1 Invalidate all entries in specified TLB congruence class (SET) with a matching PID (and matching LPID). 2 Invalidate all entries in specified TLB congruence class (SET) with a matching LPID. 3 If MSR[HV] = '1', invalidate all entries all entries in specified TLB congruence class (SET); otherwise, invalidate all entries in specified TLB congruence class with a matching LPID.

The format and operands for the **tlbiel** instructions are indicated in *Figure 4-5*. The R and RIC values determine the format of the RB operand.

*Figure 4-5. **tlbiel** Operands for the POWER9 Core*



4.10.4 Radix Translation

When the partition-table entry has HR = '1', the translation mechanism is referred to as Radix translation. Radix translation can be either single-level or nested.

For nested Radix, there exists two sets of Radix trees:

- One set of Radix trees that is managed by the guest operating system, is determined by indexing a process table using the process table ID (PID) value.
- A second set of Radix trees is managed by the host (hypervisor) software.

For a Radix host, the LPIDR value can be overridden using EA(0:1) to indicate which EA quadrant the thread is accessing as described in the ISA. The term effective LPID (effLPID) is more commonly used to discuss how the partition table is indexed. Analogous to how the effLPID is used to index the partition table for host translation, an effective PID (effPID) is the commonly used term for indexing the process table. See the *Power ISA (Version 3.0B)* for more details on EA quadrants, effLPID, and effPID. When the effLPID = '0', there exists only a single level of Radix translation. This translation is considered to be process-scoped (that is, indexed by the effPID) and used to translate the EA directly to an hRA.

For Radix translation, the EA range is 64-bits, but as stated previously, EA(0:1) indicate which EA quadrant the thread is in. When EA(2:11) are nonzero, a segment interrupt results. Thus, the usable EA space (when the thread is not in hypervisor real mode) is limited to 52 bits. In addition, the gRA is limited to 52 bits, otherwise a segment interrupt occurs. *Table 4-16* shows the details of the address bit range checking by hardware.

Table 4-16. Address Bit Range Checking by Hardware (Sheet 1 of 2)

Mode	Instruction or Data EA(0:1)	EA(2:11) (Nonzero)	Instruction or Data gRA(0:11) (Nonzero)	Guest PDE RPN(4:11) (Nonzero)	Guest PTE RPN(7:11)	Host PDE RPN(4:7) (Nonzero)	RPN(8:12) Nonzero in Host PDE or Host PTE[1]	Host PTE RPN(7) (Nonzero)	Host PTE for Instruction/ Data Access[2]
Guest Real Mode (IR/DR off)	Ignore EA(0:1); always treat as '00'.	Segment interrupt (gEA = gRA)	Segment interrupt	N/A	N/A	Ignore	Machine check	Ignore	**Paste**: ignore; otherwise, machine check

1. RPN(8:12) nonzero in any host PDE (that maps a guest table or instruction/data access) or in any host PTE that maps a guest table.
2. Host PTE for instruction/data acess where RPN(8:12) is nonzero.
3. Hardware can generate a segment interrupt due to quadrants and HV/PR values. This assumes that IR = 1 for instruction access and DR = 1 for data access.
4. Future processors might generate an instruction or data segment interrupt.

Table 4-16. Address Bit Range Checking by Hardware (Sheet 2 of 2)

Mode	Instruction or Data EA(0:1)	EA(2:11) (Nonzero)	Instruction or Data gRA(0:11) (Nonzero)	Guest PDE RPN(4:11) (Nonzero)	Guest PTE RPN(7:11)	Host PDE RPN(4:7) (Nonzero)	RPN(8:12) Nonzero in Host PDE or Host PTE[1]	Host PTE RPN(7) (Nonzero)	Host PTE for Instruction/ Data Access[2]
Nested Radix with IR/DR on	Quadrant bits	Segment interrupt	Ignore[3]	Ignore[4]	Ignore	Ignore	Machine check	Ignore	**Paste**: ignore; otherwise, machine check
Process scoped single-level Radix (effLPID = 0)	Quadrant bits	Segment interrupt	N/A[3]	N/A	N/A	Ignore	Machine check	Ignore	**Paste**: ignore; otherwise, machine check
HPT	Only part of EA	Only part of EA	N/A	N/A	N/A	Ignore	Machine check	Ignore	**Paste**: ignore; otherwise, machine check

1. RPN(8:12) nonzero in any host PDE (that maps a guest table or instruction/data access) or in any host PTE that maps a guest table.
2. Host PTE for instruction/data acess where RPN(8:12) is nonzero.
3. Hardware can generate a segment interrupt due to quadrants and HV/PR values. This assumes that IR = 1 for instruction access and DR = 1 for data access.
4. Future processors might generate an instruction or data segment interrupt.

4.10.4.1 Supported Radix Tree Configurations and Resulting Page Sizes

The Power ISA provides the general architecture to support implementations that might implement a variety of page sizes. The POWER9 processor supports only the page sizes specified in *Table 4-17* on page 104 when performing Radix translation. The values in the Level columns indicate the supported size of each level of the Radix tree for each resulting page size. All other Radix tree configurations are unsupported and result in the hardware generating an unsupported Radix tree type of DSI, HDSI, ISI, or HISI.

Table 4-17. Supported Radix Tree Configurations and Resulting Page Sizes

Page Size	Level 1 Size	Level 2 Size	Level 3 Size	Level 4 Size
4 KB	64 KB	4 KB	4 KB	4 KB
64 KB	64 KB	4 KB	4 KB	256 B
2 MB	64 KB	4 KB	4 KB	
1 GB	64 KB	4 KB		

4.10.4.2 TLB and PWC Hash Functions for Radix

The TLB is used to cache page table entries (PTEs). When the POWER9 core is running in Radix mode, the TLB is reduced to 512 entries (128 entries × 4-way set-associative). The hash algorithm for the TLB operating in this mode is specified in *Table 4-18*.

Table 4-18. TLB Hash for Radix Mode

Page Size	TLB Hash Function
4 KB	[LPIDR(29:31) XOR EA(45:47)] || [PIDR(28:31) XOR EA(48:51)]
64 KB	[LPIDR(29:31) XOR EA(41:43)] || [PIDR(28:31) XOR EA(44:47)]
2 MB	[LPIDR(29:31) XOR EA(36:38)] || [PIDR(28:31) XOR EA(39:42)]
1 GB	[LPIDR(29:31) XOR EA(27:29] || [PIDR(28:31) XOR EA(30:33)]

1. The effPID and effLPID values are used to determine the hash.
2. Partition scoped translations force PID = 0 for the hash.
3. EA in the TLB hash function description means "guest EA" for process scoped translations when HV = 0; "host EA" when HV = 1 and LPID = 0; and "host EA" (which equals the "guest RA") for partition scoped translations.

The other 512 entries (128 entries × 4-way set-associative) of the TLB array are used as a page-walk cache. The PWC is used to cache page directory entries (PDEs). Each level of the Radix tree has a unique cache to store entries (level 1 entries are not mixed with level 2 or level 3 entries in the cache). Entries are tagged with the LPID and PIDR values used when performing the translation. The full EA must get to a particular level in the tree and is also stored as part of the tag in an entry: level 1 EA(12:24), level 2 EA(12:33), level 3 EA(12:42). An entry must match the LPID, PID, and EA to hit within the PWC. The PWC is accessed after the TLB lookup, and its results are only used in the event of a TLB miss. In a TLB miss/PWC hit translation, the Radix walk begins using the PDE data retrieved from the PWC entry. The hash algorithm for the PWC is specified in *Table 4-19*.

Table 4-19. PWC Hash for Radix Mode

PDE Level	PWC Hash Function
1	[LPIDR(29:31) XOR EA(18:20)] || [PIDR(28:31) XOR EA(21:24)]
2	[LPIDR(29:31) XOR EA(27:29)] || [PIDR(28:31) XOR EA(30:33)]
3	[LPIDR(29:31) XOR EA(36:38)] || [PIDR(28:31) XOR EA(39:42)]

1. Partition scoped translations force PID = 0 for the hash.
2. EA in the PWC hash function description means "guest EA" for process scoped translations when HV = 0, "host EA" when HV = 1 and LPID = 0, and "host EA" (which equals the "guest RA") for partition scoped translations.

4.10.4.3 tlbie and tlbiel Encodings for Radix Translations

When software must invalidate the translation caches for one of these page sizes, it should execute the appropriate **tlbie** or **tlbiel** instruction sequence as specified in the *Power ISA (Version 3.0B)* with R = '1' and AP set to the corresponding value as indicated in *Table 4-20*, and a RIC value of either '0' or '2' for the required outcome.

Table 4-20. ***tlbie(l)*** *Page Encodings for POWER9 Radix (R = '1')* Only RIC≠ 3 is supported

RB(32:51)	RIC	RB(56:58) AP	Actual Page Size to be Invalidated
vvvv vvvvvvvv vvvvvvvv	0, 2	000	4 KB
vvvv vvvvvvvv vvvvxxxx	0, 2	101	64 KB
vvvv vvvvvvvv xxxxxxxx	0, 2	001	2 MB
vxxx xxxxxxxx xxxxxxxx	0, 2	010	1 GB

1. All other values of AP should not be used when R = 1 and results in a machine check interrupt.
2. 2 MB and 1 GB page sizes are only supported for Radix.

Alternatively, software can use the **tlbiel** instruction with IS ≠ 0 per the *Power ISA (Version 3.0B)* to perform a congruence class invalidation of the TLB on the processor that executes the **tlbiel** instruction. When this option is chosen, RB(45:51), select the congruence class of the TLB and/or PWC to be invalidated.

See *Appendix B* ***tlbie and tlbiel Encodings for Radix Translations*** on page 469 for details.

4.10.5 Changing the Process ID Register

The POWER9 processor implements 20-bits of process ID in the Process ID Register (PIDR).

When using Radix translation and software wants to change the PIDR value, it should do so in either guest real mode (or hypervisor mode) or it should use the following sequence if translation is enabled:

1. branch to quadrant 3
2. **mtpidr**
3. **isync**
4. branch back to quadrant 0

4.10.6 Switching between Radix and HPT Partitions

To switch between running a Radix partition and an HPT partition, the following sequence must be observed:

1. Start in an HPT partition with translation on.

   Note: HID[8] is likely set to '0' at this point for full TLB mode.
2. Switch all active threads on the core to hypervisor real mode.
3. Invalidate all of the TLB, ERATs, and translation caching by executing one tlbie with RIC = '2', IS = '3', R = '0', PRS = '0' and one tlbie with RIC = '2', IS = '3', R = '0', and PRS = '1'.

   Note: The PRS = '1' form is only required for the nest MMU because the core does not support UPRT = '1' for HPT partitions.
4. Change LPIDR to point to a Radix partition.

5. Execute a **mthid** instruction to set bit 8 to configure the TLB for half-TLB mode (required for Radix translation).
6. Change LPCR so that the following conditions are true: UPRT = '1', HR = '1', VC = '000'.
7. Execute an **rfid** to turn translation back on in Radix partition.

To switch from a Radix partition to an HPT partition, the following sequence must be observed:

1. Start out in a Radix partition with translation on.

   Note: HID[8] is set to '1' at this point for "half-TLB mode" (required for radix translation).
2. Switch all active threads on the core to hypervisor real mode.
3. Invalidate all of the TLB, ERATs, and translation caching by executing one tlbie with RIC = '2', IS='3', R = '1', PRS = '0' and one tlbie with RIC = '2', IS = '3', R = '1' and PRS = '1'.
4. Change LPIDR to point to an HPT partition.
5. Optional: Execute a **mthid** instruction to clear bit 8 to configure the TLB for "full-TLB mode". This allows the TLB to be fully utilized for HPT partitions, which should improve performance.
6. Change LPCR so that the following conditions are true: UPRT = '0', HR = '0', and set VC accordingly for the target partition.
7. Execute an **rfid** to turn translation back on in an HPT partition.

4.10.7 Hashed Page Table Translation

When the partition table entry has HR = '0', the translation mechanism is referred to as either paravirtualized or HPT. In HPT mode, there is no concept of an effPID or an effLPID, only PIDs and LPIDs. In other words, only the values found in the PIDR and LPIDR, respectively, are used to index the appropriate translation table. This translation mode is most similar to the legacy translation architecture supported on past processors such as the POWER8 processor. As state earlier, the POWER9 core only supports the LPCR[UPRT] = '0' submode of the HPT translation architecture. In this mode, the POWER9 core supports 32 software-managed SLB entries (the same as POWER8). The PIDR is not used in this submode in the processor core, but is used by the NMMU.

In HPT mode, the effective address space is 64 bits (0:63), the virtual address space is implemented as 68 bits (10:77) of the 78-bit architected maximum virtual address space, and the real address space is 51 bits (13:63).

4.10.7.1 In-Memory Segment Table and Bolted SLB Entries

The segment table is managed by the operating system running in a logical partition (LPAR) and resides in virtual memory. At a high level, when operating in this mode, the hardware searches the in-memory segment table for a matching effective segment ID (ESID) and caches matching segment table entries (STE) in the SLB. Upon subsequent accesses to that same ESID, the hardware will hit on that entry in the SLB and only walk the segment table if no matching entry is found. In addition to the segment table entries that are cached in the SLB, the hardware provides the ability for software to manage four bolted entries in the SLB. Unlike the cached segment table entries, these bolted entries are not eligible to be cast out of the SLB using the hardware's LRU policy. It is the responsibility of the operating system to manage these four bolted entries explicitly as described in the *Power ISA (Version 3.0B)*. If a matching ESID is found in the SLB, that entry is used regardless of whether another segment table entry resides in the in-memory segment table with a different ESID to Virtual Segment ID (VSID) mapping. Software must prevent more than one matching ESID in the in-

memory segment table. Furthermore, an ESID that exists in the segment table must not overlap with one of the bolted SLB entries. Violating any of these rules can result in an SLB multi-hit caused machine check interrupt.

4.10.7.2 SLB Management Instructions

The POWER9 core implements the SLB management instructions as defined in the *Power ISA (Version 3.0B)*. Specifically, the following instruction details are noteworthy:

- The **slbmfee** and **slbmfev** instructions can read any SLB entry when UPRT = '1', if the L-bit in the instruction image is set to a '1'. This is an implementation-specific feature that will only be used in the future if and when the POWER9 processor core supports UPRT = '1' for HPT translation.
- The **slbfee.** instruction writes '0' to CR field 0 whenever UPRT = '1'.

  Note: UPRT should always be set to '1' (per the ISA) whenever Radix translation is being performed (that is, LPCR[HR] = '1').
- The **slbia** instruction with IH = '101' (that is, a reserved value) is treated the same as IH = '111'.

4.10.7.3 Supported Segment and Page Sizes for HPT Translations

The POWER9 core supports two segment sizes for HPT translation: 256 MB and 1 TB. The POWER9 core also provides support for 4 KB, 64 KB, 16 MB, and 16 GB page sizes for HPT translation. Translation information for all these page sizes is kept in the TLB. Irrespective of the page size, a given page takes up only one entry in the TLB.

If a virtual address is mapped into a small (large) page and then later mapped into a large (small) page without invalidating TLB entries between changing page size, a machine check interrupt can result with an indication that a parity error occurred when the TLB was accessed to translate an effective address. The error condition can be corrected by invalidating the entire TLB and SLB.

The POWER9 core also supports multiple page sizes per segment (MPSS) as described in the Power ISA. Specifically, the POWER9 core supports mixing page size in a single segment with the following combinations only:

- 4 KB base / 64 KB actual
- 4 KB base / 16 MB actual
- 64 KB base / 16 MB actual

Table 4-21 on page 109 shows the correspondence between PTE[L, LP] values and STE[L, LP]/SLBE[L, LP] values. The supported segment table and SLB entry sizes and page sizes are also shown in *Table 4-21*. These same page sizes and their associated encodings are also used in the Partition Table Entry "PS" field.

Table 4-21. PTE and STE/SLBE Correspondence for HPT Translation

Entry Number	PTE			STE/SLBE		Base Page Size	Actual Virtual Page Size	Notes
	L	LP		L	LP			
1	0	rrrr	rrrr	0	00	4 KB	4 KB	1
2	1	0000	0000	1	00	16 MB	16 MB	
3	1	rrrr	0001	1	01	64 KB	64 KB	2
4	1	0000	0011	1	10	16 GB	16 GB	
5	1	rrrr	0111	0	00	4 KB	64 KB	1
6	1	0000	1000	1	01	64 KB	16 MB	2
7	1	0011	1000	0	00	4 KB	16 MB	1

1. Entries 1, 5, and 7 all use STE/SLBE[L, LP] = '000' encoding for base page size 4 KB but have unique PTE[L, LP] encodings for actual page size.
2. Entries 3 and 6 both use STE/SLBE[L, LP] = '101' encoding for base page size of 64 KB but have unique PTE[L, LP] encodings for actual page size.
3. Unimplemented STE/SLBE page size encodings are treated the same as the '000' case.
4. If the STE/SLBE page size is '110' (16 GB) and the segment size is small (256 MB), hardware treats the STE/SLBE page size the same as the '000' case.
5. The 'r' bits are part of the real page number. They can be any value.

4.10.7.4 TLB Hash Function for HPT

In HPT mode, the entire 1024 entries in the TLB are used to cache HPT page table entries (PTEs). The TLB is organized as 256 entries × 4-way set associative. The TLB hash function for 256 MB segments is shown in *Table 4-22* and for 1 TB segments is shown in *Table 4-23*. The corresponding hash function is used for TLB reads, writes, and snooped tlbie operations.

Table 4-22. 256 MB Segments

Page Size	Index
4 KB	[VSID(46:49) XOR EA(44:47)] || EA(48:51)
64 KB	[VSID(46:49) XOR EA(40:43)] || EA(44:47)
16 MB	VSID(46:49) || EA(36:39)
16 GB	Does not exist

Table 4-23. 1 TB Segments

Page Size	Index
4 KB	[VSID(34:37) XOR EA(44:47)] || EA(48:51)
64 KB	[VSID(34:37) XOR EA(40:43)] || EA(44:47)
16 MB	[VSID(34:37) XOR EA(32:35)] || EA(36:39)
16 GB	[VSID(34:37) XOR EA(25:28)] || EA(26:29)

4.10.7.5 tlbie and tlbiel Usage for HPT Translations

For HPT translation, the following tables show the **tlbie** and **tlbiel** specifications for various page sizes supported by the POWER9 core.

Table 4-24 shows the legal segment size and page size specifications for **tlbie** and **tlbiel** for the POWER9 HPT PTEs (R = '0') when L = '0', and RIC ≠ '3'.

*Table 4-24. Segment Size and Page Size Specifications for HPT **tlbie** and **tlbiel*** (R = '0', L = '0', and RIC ≠ '3')

RB[54:55] Segment Size	RB[63] L	RIC	RB[56:58] AP (Same as STE/SLBE[L|LP] Encoding)	Actual Page Size to be Invalidated
00	0	0,2	000	4 KB
00	0	0,2	101	64 KB
00	0	0,2	100	16 MB
01	0	0,2	000	4 KB
01	0	0,2	101	64 KB
01	0	0,2	100	16 MB

1. All other AP values must not be used when L = '0' (and R = '0') and results in a machine check interrupt.
2. RB[54:55] = '00' corresponds to a 256 MB segment size and RB[54:55] = '01' corresponds to 1 TB segment size.
3. 16 GB page with a small segment (RB(54:55] = '00') is *not* a permitted combination.
4. PRS = '1' and R = '0' is an unsupported combination (invalid form)

Table 4-25 shows the legal segment size and page size specifications for **tlbie** and **tlbiel** for the POWER9 HPT PTEs (R = '0') when L = '1' and RIC ≠ '3'.

*Table 4-25. Segment Size and Page Size Specifications for HPT **tlbie** and **tlbiel*** (R = '0', L = '1', and RIC ≠ '3')

RB[54:55] Segment Size	RB[63] L	RIC	RB[44:51] LP (same as STE/SLBE[L|LP] encoding)	Base Page Size	Actual Page Size to be Invalidated
00	1	0,2	0000 0000	16 MB	16 MB
00	1	0,2	VVVV 0001	64 KB	64 KB
00	1	0,2	0000 1000	64 KB	16 MB
01	1	0,2	0000 0000	16 MB	16 MB
01	1	0,2	VVVV 0001	64 KB	64 KB
01	1	0,2	0000 0011	16 GB	16 GB
01	1	0,2	0000 1000	64 KB	16 MB

1. All other LP values used when R = '0', L = '1', and RIC≠ '3' result in a machine check interrupt.
2. 'v' corresponds to AVA (AVPN) bits.
3. RB[54:55] = '00' corresponds to 256 MB segment size and RB[54:55] = '01' corresponds to 1 TB segment size.
4. 16 GB page with a small segment (RB[54:55] = '00') is not a permitted combination.

*Table 4-26. Segment Size and Page Size Specifications for HPT **tlbie** Cluster Bombs (R = '0', L = '0', and RIC = '3')* **Note: tlbiel** with RIC = 3 is an invalid instruction form and is treated as a NOP.

RB[54:55] Segment Size	RB[63] L	RIC	RB[56:58] AP (same as STE/SLBE[L|LP] encoding)	Actual Page Size to be Invalidated
00	0	3	110	Eight consecutive 4 KB pages aligned on 32 KB boundary
00	0	3	111	Eight consecutive 64 KB pages aligned on 512 KB boundary
01	0	3	110	Eight consecutive 4 KB pages aligned on 32 KB boundary
01	0	3	111	Eight consecutive 64 KB pages aligned on 512 KB boundary

1. All other L and AP values and combinations used when R = '0', L = '0' and RIC = '3' result in a machine check interrupt.
2. RB[54:55] = '00' corresponds to 256 MB segment size and RB[54:55] = '01' corresponds to 1 TB segment size.
3. PRS = '1' and R = '0' is an unsupported combination (invalid form)
4. The POWER9 core has dropped support of range bombs.

See *Appendix C **tlbie and tlbiel Encodings for HPT Translations*** on page 485 for details.

4.10.8 Instruction Effective-to-Real Address Translation Cache

The POWER9 processor core includes a 64-entry, fully-associative instruction effective-to-real address translation (I-ERAT) for fast translation of instruction effective addresses into physical (real) addresses on a per LSU slice basis. The ERAT is dynamically shared between all four threads.

For hypervisor real mode accesses, the I-ERAT entries are shared by all threads on a given core. The ERAT is implemented as a CAM that supports page sizes of 4 KB, 64 KB, 2 MB (Radix translation only), and 16 MB. Instruction accesses to 1 GB or 16 GB pages (HPT translations only) are installed in the I-ERAT as multiple 16 MB page entries as required.

Because addresses associated with nonhypervisor real mode accesses are translated differently than those associated with virtual-mode accesses, the I-ERAT must keep the MSR[IR] and MSR[HV] bits (along with various bits of translation information) in each entry. This allows the I-ERAT to distinguish between translations that are valid for the various modes of operation.

For HPT translation, the contents of each I-ERAT entry is the result of a page table search based on the contents of an SLB or segment-table entry. To maintain consistency with the SLB, the following instructions cause entries in the I-ERAT that belong to the thread executing the instruction to be invalidated:

- **slbia** - Use the appropriate IH field value as described in the *Power ISA Operating Environment Architecture - Book III (version 3.0B)*, some or all entries are invalidated for that thread.
- **mtiamr** - All entries are invalidated for that thread.

The **slbie** instruction causes invalidation of an I-ERAT entry belonging to the thread (no impact to the other thread) only if there is a perfect address match (that is, for invalidation effective address bits, EA[0:35] are matched for an **slbie** small 256 MB segment, EA[0:23] are matched for an **slbie** 1 TB segment) and the Class bit specified by the **slbie** instruction matches the Class bit of the SLB/ERAT entry being invalidated.

Because the POWER9 processor core does not support UPRT = '1' mode for HPT translations, the core does not invalidate any SLB or ERAT entries as the result of an **slbieg** instruction. The **slbieg** instruction is used soley for invalidating the segment table entries cached by the nest MMU hardware.

The **tlbie** instruction (or the detection of snooped-tlbie operations) invalidates all I-ERAT entries (irrespective of the thread) in the I-ERAT that have a perfect match. In other words, the entry is invalidated only if the LPIDR value matches the LPID value from the snooped-tlbie and:

- EA[36:51] are matched for HPT tlbie 4 KB page
- EA[14:51] are matched for Radix tlbie 4 KB page
- EA[36:48] are matched for HPT tlbie 8 × 4 KB cluster bomb
- EA[36:47] are matched for HPT tlbie 64 KB page
- EA[14:47] are matched for Radix tlbie 64 KB page
- EA[36:44] are matched for HPT tlbie 8 × 64 KB cluster bomb
- EA[14:42] are matched for Radix tlbie 2 MB page
- EA[36:39] are matched for HPT tlbie 16 MB page
- EA[14:33] are matched for Radix tlbie 1 GB page
- EA[24:29] are matched for HPT tlbie 16 GB page

Note: EA refers to the effective address for process-scoped Radix translations, the guest real address for partition-scoped Radix translations, and the virtual address for HPT translations.

For Radix translation, the I-ERAT caches the flattened guest effective address to host real address translation that results from searching both the guest and host page tables. The Class bit in the I-ERAT is set to '1' for quadrants 0, 1,and 2 and is set to '0' for quadrant 3 accesses (regardless of HV or PR). The preferred method for invalidating the entire I-ERAT when using Radix translation is to execute an **slbia** with IH = x'7' or an **mtiamr** instruction. For more precise ERAT invalidation where software must retain quadrant 0 accesses (regardless of HV or PR), **slbia** with IH = x'1' or IH = x'3' can be used. Additionally, **mtpidr** and **mtlpidr** instructions perform an implicit **slbia** with IH = x'3'.

Upon power-on, each I-ERAT entry is set to the invalid state.

Table 4-27 on page 113 describes how the entries in the I-ERAT are created. For HPT translation (HR = '0'), the resulting I and G values in the table are solely determined by the I and G values from the PTE in memory. For Radix translation (HR = '1'), the resulting I and G values in the table reflect the net or effective I and G values as derived from *Figure 4-6*.

Figure 4-6. Net or Effective I and G Values (I-ERAT)

	Host ATT	'00'	'01'	'10'	'11'
Guest ATT	(SAO, I, G)	'000'	'100'	'011'	'010'
'00'	'000'	Normal	Normal	WIMG Miscompare	WIMG Miscompare
'01'	'100'	Normal	Normal	WIMG Miscompare	WIMG Miscompare
'10'	'011'	No Execute	No Execute	No Execute	No Execute
'11'	'010'	Normal	Normal	Cache-Inhibited Fetch	Cache-Inhibited Fetch

Notes:
- Dark green = no exception reported.
- Light green = instruction fetch from a caching inhibited page occurs.
- Light red = no execute (for example, SRR1[35] = '1'). ISI occurs per the Power ISA.
- Dark red = mismatched ATT. ISI occurs per bit SRR1[34] = '1' in the Power ISA.

The SAO bit is not applicable to instruction accesses and thus is not shown in *Table 4-27*. The exception cases listed previously are either a mismatched ATT or "no-execute" type of exception as dictated by the *Power ISA (Version 3.0B)*.

Table 4-27. I-ERAT I and G Bit Setting

Condition				I and G Determined By:		Resulting Action
MSR[IR]	HR	MSR[HV]	First Access I = 1 Fetch			
1	X	X		PTE	PTE	If G = '0', the page is written into the I-ERAT using the I-bit value and page size determined from the PTE as described previously. If G = '1', an ISI is taken.
0	0	0		PTE	PTE	Virtual real mode. If G = '0', the page is written into the I-ERAT using the I-bit value and page size is determined from the PTE as described previously. If G = '1', an ISI is taken.
0	1	0		PTE	PTE	Guest real mode. An entry is created with the I and G values set from the host PTE. If G = '1', an ISI is taken.
0	X	1	Yes	1	N/A	Page-based RMSC mode. A 2 MB page is installed in the I-ERAT with I = '1' and G = '0'.
0	X	1	No	0	N/A	Page-based RMSC mode. A 2 MB page is installed in the I-ERAT with I = '0' and G = '0'.

4.10.9 Data Effective-to-Real-Address Translation

The POWER9 processor core includes a 64-entry, fully-associative data effective-to-real-address translation (D-ERAT) for fast translation of data effective addresses into physical (real) addresses on a per LSU slice basis. The ERAT is dynamically shared between all four threads.

For hypervisor real- mode accesses, the D-ERAT entries are shared by all threads on a given core. All other accesses are not shared across threads, even when the LPIDR value is set to the same value on two or more threads. The ERAT is implemented as a CAM that supports page sizes of 4 KB, 64 KB, 2 MB (Radix translation only), and 16 MB. Data accesses to 1 GB or 16 GB pages (HPT translations only) are installed in the D-ERAT as multiple 16 MB page entries as required.

Because addresses associated with nonhypervisor real-mode accesses are translated differently than those associated with virtual-mode accesses, the D-ERAT must keep the MSR[DR] and MSR[HV] bits (along with various bits of translation information) in each entry. This allows the D-ERAT to distinguish between translations that are valid for the various modes of operation.

For HPT translation, because the contents of each D-ERAT entry is the result of a page-table search based on the contents of an SLB or segment table entry, to maintain consistency with the SLB, the following instruction causes entries in the D-ERAT that belong to the thread executing the instruction to be invalidated:

- **slbia** using the appropriate IH field value as described in the *Power ISA Operating Environment Architecture - Book III (version 3.0B)*, some or all entries are invalidated for that thread

The **slbie** instruction causes invalidation of a D-ERAT entry belonging to the thread (no impact to the other thread) only if there is a perfect address match (that is, for invalidation effective address bits, EA[0:35] are matched for an **slbie** small 256 MB segment, EA[0:23] are matched for an **slbie** 1 TB segment). Unlike the

I-ERAT, no Class bit match is required for **slbie** invalidation of the D-ERAT (nor the SLB). Note that the preceding sentence is a design-implementation feature, not an architecture requirement. Similarly, because the POWER9 processor core does not support UPRT = '1' mode for HPT translations, the core does not invalidate any SLB or ERAT entries as the result of an **slbieg** instruction. The **slbieg** instruction is used solely for invalidating the segment table entries cached by the nest MMU hardware.

The **tlbie** instruction (or the detection of snooped-tlbie operations) invalidates all D-ERAT entries (irrespective of the thread) in the D-ERAT that have a perfect match. In other words, the entry is invalidated only if the LPIDR value matches the LPID value from the snooped-tlbie and:

- EA[36:51] are matched for HPT **tlbie** 4 KB page
- EA[14:51] are matched for Radix **tlbie** 4 KB page
- EA[36:48] are matched for HPT **tlbie** 8 $\times$ 4 KB cluster bomb
- EA[36:47] are matched for HPT **tlbie** 64 KB page
- EA[14:47] are matched for Radix **tlbie** 64 KB page
- EA[36:44] are matched for HPT **tlbie** 8 $\times$ 64 KB cluster bomb
- EA[14:42] are matched for Radix **tlbie** 2 MB page
- EA[36:39] are matched for HPT **tlbie** 16 MB page
- EA[14:33] are matched for Radix **tlbie** 1 GB page
- EA[24:29] are matched for HPT **tlbie** 16 GB page

Note: EA refers to the effective address for process-scoped Radix translations, the guest real address for partition-scoped Radix translations, and the virtual address for HPT translations.

For Radix translation, the D-ERAT caches the flattened guest effective address to host real address translation resulting from searching both the guest and host page tables. The Class bit in the D-ERAT is set to '1' for quadrants 0, 1, and 2, and set to '0' for quadrant 3 accesses (regardless of HV or PR). The preferred method for invalidating the entire D-ERAT when using Radix translation is to execute an **slbia** with IH = x'7'. For more precise ERAT invalidation where software must retain quadrant 0 accesses (regardless of HV or PR), **slbia** with IH = x'1' or IH = x'3' can be used. Additionally, **mtpidr** and **mtlpidr** instructions perform an implicit **slbia** with IH = x'3'.

Upon power-on, each D-ERAT entry is set to the invalid state.

According to the Power ISA, aliasing the I-bit storage attribute is prohibited. In the POWER9 core, due to the caching of pages in the ERATs, software should avoid accessing the same real page with different values for the I-bit storage attribute. Failure to follow this restriction can result in a cache paradox or other boundedly undefined behavior.

4.10.9.1 D-ERAT I and G Bit Setting

Entries in the D-ERAT are created as described in *Table 4-28*. For HPT translation (HR = '0'), the resulting I and G values in *Table 4-28* are solely determined by the I and G values from the PTE in memory. For Radix translation (HR = '1'), the resulting I and G values in *Table 4-28* reflect the "net" or "effective" I and G values as derived from *Figure 4-7*.

Figure 4-7. Net or Effective I and G Values (D-ERAT)

	Host ATT	'00'	'01'	'10'	'11'
Guest ATT	(SAO, I, G)	'000'	'100'	'011'	'010'
'00'	'000'	'000'	'100'	Exception	Exception
'01'	'100'	'100'	'100'	Exception	Exception
'10'	'011'	'001'	'001'	'011'	'011'
'11'	'010'	'000'	'000'	'010'	'010'

Note:
- Light Green means no exception reported.
- Red means "mismatched ATT" type of DSI occurs (for example,DSISR(34) = '1' on DSI as per ISA).

The exception cases listed above are a mismatched ATT type of exception as dictated by the *Power ISA (Version 3.0B)*.

Table 4-28. D-ERAT I and G Bit Setting

Condition				SAO	I and G Determined By:		Resulting Action
MSR [DR]	HR	MSR [HV]	First Access HV CI Instruction				
1	X	X		0	PTE	PTE	When the WIMG (HPT) or net ATT (Radix) indicate the page is not SAO, an entry is created with SAO = '0' and the I and G values are set from the PTE.
1	X	X		1	0	0	When the WIMG (HPT) or net ATT (Radix) indicate the page is SAO, an entry is created with SAO = '1' , I = '0', and G = '0'
0	0	0		PTE	PTE	PTE	Virtual real mode. An entry is created with the I and G values set from the host PTE. The SAO bit is set if the PTE (WIMG) bits specify SAO.
0	1	0		PTE	PTE	PTE	Guest real mode. An entry is created with the I and G values set from the host PTE. The SAO bit is set if the PTE (ATT) bits specify SAO.
0	X	1	Yes		1	1	Page-based RMSC mode. If the first access is caused by a hypervisor CI load or store (for example, **ldcix**, **stdcix**, and so on), an entry is established as I = '1' and G = '1'.
0	X	1	No		0	0	Page-based RMSC mode. If the first access is caused by any instruction other than a hypervisor CI load or store, storage is G = '0' and an entry is established as I = '0' and G = '0'.

Caching-Inhibited Paradox Cases

If a caching-inhibited load instruction hits in the L1 data cache, the load data is serviced from the L1 data cache and no request is sent to the NCU.

If a caching-inhibited store instruction hits in the L1 data cache, the store data is written to the L1 data cache and sent to the NCU. Note that the L1 data cache and L2 cache are no longer coherent.

These scenarios are true both for storage accesess marked as caching-inhibited by the PTE I-bit and for the hypervisor caching-inhibited load and store instructions.

The POWER9 core supports the page-based real-mode storage control (RMSC) mechanism that allows speculative access to DR = '0' space, if there is real memory there.

4.10.10 Translation Lookaside Buffer and PWC

The POWER9 core contains a unified (combined for both instruction and data), 1024-entry, 4-way set-associative TLB (LRU-based replacement algorithm) for HPT mode. When the core is operating in Radix mode, the TLB is logically cut in half with the other half being dedicated as a page walk cache (PWC). Regardless of what translation mode the core is operating in, the TLB is 4-way set associative. The TLB is used to cache PTEs. The PWC is used to cache page directory entries (PDEs) for Radix page tables. In addition, the POWER9 core contains one 64-entry, fully-associative I-ERAT and one 64-entry, fully-associative D-ERAT. The TLB is a cache of recently-used page table entries. The PWC is a cache of recently used page directory entries (for Radix). The ERATs are caches that contain flattened translations derived from information in the various page tables, segment table entries, or bolted SLB entries. The TLB, PWC, and ERATs are loaded and managed by hardware.

In the POWER9 core, the TLB entry stores the LPID in each TLB entry to indicate which partition loaded that TLB entry. Because the virtual and real address space are the same for all software threads within a logical partition, the TLB, which keeps the mapping from virtual-to-real address space, are completely shared by the threads within a partition and there is no thread-ID bit required in the TLB to identify which entry belongs to which thread. Different partitions have different mappings from virtual-to-real address space; therefore, TLB entries cannot be shared between partitions. A given entry in the TLB can be used by all the threads within a partition at the same time. Threads in different partitions are not able to access TLB entries from another partition.

For more details on the organization and size of the TLB for each type of translation mode, see *Table 4-18 TLB Hash for Radix Mode* on page 105, *Table 4-19 PWC Hash for Radix Mode* on page 105, *Table 4-22 256 MB Segments* on page 109, and *Table 4-23 1 TB Segments* on page 109.

The POWER9 core supports a hardware update of the storage access recording bits (reference and change) into the memory-based page table.

The POWER9 core supports a TLB hit under miss and two table concurrent tablewalks. The POWER9 core also supports two outstanding I-ERAT misses (from the four threads) and four outstanding D-ERAT misses at the same time.

The POWER9 core supports lockless TLBIE operations. The architectural requirement that only one thread at a time can execute **tlbie**/**tlbsync** instructions during a page table modification need not be followed (see the Page Table Updates section of the *Power ISA Operating Environment Architecture - Book III (version 3.0B)*). This was traditionally implemented with a single global lock for the entire page table modification sequences. The term lock-less TLBIEs refers to the POWER9 core's ability to manage concurrent **tlbie**/**tlbsync** sequences from multiple threads without this global lock.

However, software must still ensure that concurrent, conflicting, racing PTE updates from more than one thread do not occur (the hardware performs the updates in some fashion, but the end result is undefined due to the racing of the updates to the same PTE entry) and therefore, software locks or some other synchronization discipline are still required to prevent these collisions as necessary.

The execution of **tlbie** instructions or the detection of **snooped-tlbie** operations off the bus cause an index-based invalidate to occur in the TLB, if there is a match. In other words, an entry is invalidated only if there is a perfect match of the effective address supplied by the tlbie operation and the content of the TLB entry.

The POWER9 core does not support the **tlbia** instruction.

Upon power-on, the POWER9 core initializes each TLB entry to the invalid state.

4.10.11 Segment Lookaside Buffer

For HPT translation, the POWER9 core contains a unified (combined for both instruction and data), 32-entry, fully-associative SLB per thread. Although the *Power ISA (Version 3.0B)* supports both a hardware managed SLB which caches the in-memory segment table (that is, UPRT = '1') and a strictly software-managed SLB (UPRT = '0'), the POWER9 core only supports UPRT = '0' when performing HPT translation. Therefore, software must set the LPCR[UPRT] = '0' when using HPT translation or the results are undefined.

While the **slbieg** instruction does not invalidate SLB entries in the processor core when UPRT = '0', it is used to managed STEs cached by the NMMU. Outstanding **slbieg** instructions are ordered by the **slbsync** instruction per the *Power ISA (Version 3.0B)*. When the LPCR[UPRT] = '0', the segment table is not searched and all 32 entries for each thread can only be updated by software by using the **slbmte** instruction.

Information derived from the SLB can also be cached in the I-ERAT or the D-ERAT along with information from the TLB. As a result, many of the SLB management instructions have effects on the ERATs, as well as on the SLB itself.

The POWER9 core supports both 256 MB and1 TB segment sizes. Bit 0 of the SLB[B] field is ignored by the POWER9 core and should always be set to '0' per the Power ISA for unimplemented segment size encodings.

Because the SLB is managed by software (the operating system) either via the segment table or bolted entries, it is possible that multiple entries can be incorrectly set up to provide translations for the same effective address. If an effective address is translated by more than one SLB entry (that is, the ESID fields of the entries are identical or overlap), a machine check interrupt results with an indication that a parity error occurred when the SLB was accessed. When this happens the hardware logically ORs the data in the conflicting entries. The machine check handler can look at the SLB contents to try to determine if conflicting entries have been provided. When a parity error occurs not due to multiple entries, the entire SLB must be reloaded because the DAR does not contain an address indicating which entry caused the parity error. If the source of the error was due to multiple entries, the conflicting entries must be corrected for the translation to proceed, which might also be accomplished by reloading the entire SLB with good entries.

4.10.12 Discontinued Translation Support Items

4.10.12.1 Address Space Register

Not to be confused with the Access Segment Descriptor Register (ASDR), the Address Space Register (ASR) has been removed from the Power ISA and thus is not supported by the POWER9 processor. The ASDR is supported as described in the *Power ISA (Version 3.0B)*.

4.10.13 Block Address Translation

Although this facility existed in earlier versions of the architecture, it is no longer part of the Power ISA. As a result, the POWER9 core does not support block address translation.

4.10.13.1 Support for 32-Bit Operating Systems

The POWER9 processor does not support the optional bridge facility and instructions for 64-bit implementations described in the Bridge-to-SLB Architecture section of the *Power ISA Operating Environment Architecture - Book III (version 3.0B)*.

As a result, the following instructions are not supported in the POWER9 processor:

- **mtsr** - Move to segment register
- **mtsrin** - Move to segment register indirect
- **mfsr** - Move from segment register
- **mfsrin** - Move from segment register indirect

4.10.13.2 Real Mode

The POWER9 core does not support real mode accesses that used the Real Mode Offset Register (RMOR) and Real Mode Limit Selector (RMLS) on previous generation processors. As such, per the *Power ISA (Version 3.0B)*, LPCR[0] is considered reserved and, when using HPT translation, the POWER9 core behaves like previous generation processors (such as the POWER8 core) did when LPCR[0] = '1'. In other words, nonhypervisor real mode accesses for HPT translation are always treated as virtual real-mode accesses as per the *Power ISA (Version 3.0B)*. When Radix translation in guest real mode (that is, IR = '0' or DR = '0') is being used, the guest EA (gEA) equals the guest RA (gRA), which is then translated by the partition-scoped Radix trees.

4.10.14 Reference and Change Bits

When performing Radix translation, the POWER9 hardware triggers the appropriate interrupt (DSI, HDSI, ISI, or HISI) as defined in the *Power ISA (Version 3.0B)* for the mode and type of access whenever Reference (R) and Change (C) bits require setting in either the guest or host page-table entry (PTE). When performing HPT translation, the hardware performs the R and C bit updates nonatomically (same behavior as the POWER8 processor).

For HPT PTEs, the W and M bits in the PTE are assumed to be '01' respectively. If the change bit is updated, the W and M bit in the PTE are set to '01' respectively by the hardware.

The POWER9 core can speculatively set R bits in the PTE. In some rare circumstances, the POWER9 core can speculatively set the page table entry C bit.

4.10.15 Storage Protection

For HPT translation, the *Power ISA (Version 3.0B)* states that whether an instruction fetch is permitted from a page marked "no access" is implementation dependent. In the POWER9 core, these instruction fetches are permitted to continue without signaling an exception. The POWER9 core supports storage protection modes in the Power ISA, with 32 virtual-page class keys for HPT translation. IAMR, AMOR, and UAMOR are implemented as 32-bit SPRs.

4.10.16 Hypervisor Real Mode Storage Control

The POWER9 core supports the ability to control cacheability of data and instruction accesses while in hypervisor real mode based on the history block (also known as, page-based) mechanism described in the Power ISA section on the Hypervisor Real Mode Storage Control (RMSC). Accesses performed in hypervisor real mode are cached in the I-ERAT and D-ERAT for instruction and data accesses respectively as 2 MB pages.

The POWER9 core supports RMSC for data storage and instruction storage. The real memory in a system is often noncontiguous and the hypervisor data and instruction storage accesses can be scattered across the address space. The page-based RMSC architecture and implementation allows speculative access safely in system memory. The first time the HV = '1' access is made in DR = '0' and IR = '0' mode, it is done nonspeculatively. After the first access to a given real page, a D-ERAT entry or I-ERAT entry is established. If the first access to the page was cacheable, the page is installed in the D-ERAT with IG = '00' and thus, all subsequent accesses to said page can be performed speculatively while still ensuring that the access is made to system memory. However, if the first access to the page was a noncacheable access, the page is installed in the D-ERAT with IG = '11' and thus, all subsequent accesses to that page are considered noncacheable and are performed nonspeculatively as well.

If a page is already installed in the D-ERAT as IG = '00' and a subsequent caching inhibited load or store instruction (for example, **lbzcix**, **stbcix**) accesses that same page, a DSI is taken with DSISR[62] set to '1'.

4.10.17 Storage Access Modes - WIMG and ATT Bits

Because the POWER9 processor supports both HPT and Radix translation, two methods of specifying storage attributes on a per page basis exist in the *Power ISA (Version 3.0B)*. For HPT, the WIMG bits determine this. For Radix, the ATT bits determine this and the *Power ISA (Version 3.0B)* shows how ATT values correspond to their HPT WIMG equivalent values. The remainder of this section discusses storage attributes in HPT terms, but the Radix ATT equivalent values also apply.

The POWER9 core always assumes W = '0' and M = '1' independent of the value of these bits in the page table entry. For HPT PTEs, when the hardware is performing a change bit update, it writes the W and M bits as W = '0' and M = '1'. Per the Power ISA, accessing a page as both I = '0' and I = '1' is boundedly undefined. Software should avoid aliasing the I-bit on a page basis. Failing to do so can result in cache paradox situations which can lead to memory corruption.

Table 4-29 summarizes the treatment of the WIMG bits in the POWER9 core.

Table 4-29. WIMG Bits

ATT	WIMG	Description
00	x0x0	Treated as WIMG = '0010'
01	1110	Treated as WIMG = '0010' but accesses are strongly ordered
10	x1x1	Treated as WIMG = '0111'
11	01x0	Treated as WIMG = '0110'

For the noncacheable unit (NCU), the IG combination has the following meaning in the POWER9 core to control the store ordering and store gathering (see *Table 4-30*).

Table 4-30. IG Bits

ATT	IG	Description
0x	10	Gather, reorder in NCU is allowed
1x	11	No gather, no reorder in NCU is allowed

In IG = '11' mode, cache-inhibited loads cannot be reordered relative to loads, and cache-inhibited stores cannot be reordered relative to cache-inhibited stores. Cache-inhibited loads can be reordered relative to cache-inhibited stores and vice-versa (if it is necessary to maintain ordering between loads and stores, barrier instructions must be used). There is no defined ordering between cache-inhibited load or store operations from different threads.

In IG = '11' mode, gathering is not permitted for either load or store operations within or between threads.

In IG = '10' mode, cache-inhibited loads or stores from a given thread can be gathered and can be reordered. This mode allows for higher performance with a certain loss of control of the order in which the operations are completed or whether operations are gathered (barriers can be used where necessary to re-establish order). There is no defined ordering between cache-inhibited load or store operations from different threads.

4.10.18 Speculative Storage Accesses

The POWER9 core can execute load instructions to nonguarded storage speculatively. This can occur when a load instruction is encountered on a predicted branch path or when a logically preceding instruction causes an interrupt. As a result, it is possible for a speculative load that misses in the on-chip cache hierarchy to initiate an external storage request even if that load instruction is not actually executed as part of the true instruction stream.

4.10.19 TLB Invalidate Entry (tlbie and tlbiel) Instruction

See *Section 4.10 Translation Architecture* on page 97 for details regarding whether **tlbie** and **tlbiel** are considered privileged or hypervisor privileged instructions.

See *Section 4.10.3* ***tlbie and tlbiel Instruction Format and Operands*** on page 99, *Section 4.10.4 Radix Translation* on page 103, and *Section 4.10.7 Hashed Page Table Translation* on page 107 for details on **tlbie** and **tlbiel** instruction usage.

The *Power ISA (Version 3.0B)* describes a number of cases for **tlbie** and **tlbiel** as invalid forms.

The POWER9 core truncates RS[0:31] and RS[32:63] to the supported size of PID and LPID respectively. No interrupt is generated for values that exceed the implement PID and LPID sizes.

The POWER9 core ignores RB[54] when R = 0.

The POWER9 core reports a machine check interrupt for unsupported AP and LP values specified in the RB register.

See *Section 4.10.26.7 Machine Check Interrupt* on page 130 for a list of invalid instructions forms for **tlbie**/**tlbiel** that result in a machine check interrupt.

4.10.20 TLB Invalidate All (tlbia) Instruction

The **tlbia** instruction is not implemented in the POWER9 core and if detected causes a hypervisor emulation assistance interrupt. The effects of the instruction can easily be emulated by executing a series of **tlbiel** instructions to each congruence class in the TLB by incrementing the effective address bits [44:51] through their full range, and by setting the IS field of the **tlbiel** instruction to the appropriate values as described in the Power ISA. To invalidate all entries irrespective of the LPAR ID, MSR[HV] must equal '1'.

4.10.21 TLB Synchronize (tlbsync) Instruction

On a given thread, the **tlbsync** instruction is used to synchronize the completion of the **tlbie** instruction. Only one **tlbsync** instruction is required to synchronize the completion of a group of **tlbie** instructions. See *Section 4.10 Translation Architecture* on page 97 for details regarding when **tlbsync** is considered a privileged or hypervisor privileged instruction. The instruction is otherwise implemented as described in the Power ISA.

4.10.22 SLB Synchronize (slbsync) Instruction

On a given thread, the **slbsync** instruction is used to synchronize the completion of both the **slbieg** or the **slbiag** instruction. Only one **slbsync** instruction is required to synchronize the completion of a group of **slbieg** instructions. See *Section 4.10 Translation Architecture* on page 97 for details regarding when **slbsync** is considered a privileged or hypervisor privileged instruction. The instruction is otherwise implemented as described in the Power ISA.

4.10.23 Support for Store Gathering

The POWER9 core performs gathering of cacheable stores to reduce the store traffic into the L2 cache. For cacheable stores, the gathering occurs in L2 store queues that sit above the L2 cache. The store queue is shared by the threads. The store queue is comprised of two banks of sixteen 64-byte wide, fully-associative entries or gather stations. Stores can be gathered while architecturally permitted (that is, there is no intervening barrier operation) and the matching address is valid in the store queue. The conditions for pushing the store queue data into the L2 cache are not visible to the programmer.

Gathering of cache-inhibited stores is also supported and can be disabled with a mode bit in the noncacheable unit (NCU) configuration register. There are sixteen 64-byte gather stations in the NCU.

4.10.24 Cache Coherency Paradoxes

Accesses to a given cache line as both cacheable and caching inhibited are not supported in either the Power ISA or the POWER9 chip. Because the value of the I-bit is cached by the ERATs inside the processor core, cacheable accesses can be performed speculatively and thus, software should avoid aliasing the I-bit (that is, caching-inhibited bit) on a per page basis. Failure for software to adhere to this restriction can lead to cache corruption.

4.10.25 Handling Parity Error, Multi-Hit, and Uncorrectable Errors

4.10.25.1 Parity Error

If there is a parity error in the D-cache, I-cache, D-ERAT, I-ERAT, TLB or several other register files, SRAM dataflow or control structures (but not the SLB), the POWER9 core sets the relevant FIR bit and initiates the instruction retry and recovery (IRR) process to "clean up" all the architected states and flush the caches, ERATs, and TLB, but keep the SLB as is. Software restores the SLB. After the recovery process, a hypervisor maintenance interrupt (HMI) is generated. On a successful recovery, the HMER indicates a successful recovery.

If the same parity error occurs several times and reaches a threshold, the hypervisor can decide that the core is nonfunctional. The threshold counter is maintained by the hypervisor in software.

HID[5] must be set to '0', otherwise processor recovery does not work.

Note: The IRR process is engaged for detection of any recoverable parity error in the core or due to the firing of a control checker.

There is a separate FIR bit and FIR extension bits for a parity error in the I-cache, D-cache, SLB, D-ERAT, I-ERAT, TLB, and a few other structures. For all the other register files, there is one shared FIR bit to indicate parity error.

4.10.25.2 Multi-Hit

If there is a multi-hit in the D-ERAT, TLB, or SLB, the core finishes the operation with a machine check interrupt and sets the proper DSISR bit to indicate where the multi-hit was detected.

A multi-hit in the D-ERAT and SLB can occur due to a hardware failure. Multi-hit means more than one entry matched the EA in the D-ERAT (ESID in the case of an SLB). Due to their CAM structure, the result is a "bitters logical or" of the RA of the multiple entries (VSID in case of SLB). Because of this "bit-wise logical or", multi-hit is very likely to generate a parity error as well.

Because the SLB is managed by software with the Power ISA, a software bug can result in a multi-hit in SLB structures. There is no known case of multi-hit in I-ERAT that can produce a wrong result.

There are separate FIR bits for a multi-hit in the D-ERAT, TLB, and SLB.

4.10.25.3 Both Multi-Hit and Parity Error

If both multi-hit and parity errors happen in the D-ERAT or TLB, the processor core initiates an IRR process. No machine check is presented. However, after the recovery operation, the processor core provides an HMI interrupt.

For an SLB, any error causes the processor to take a machine check interrupt. The FIR bit setting indicates both multi-hit and parity error.

4.10.25.4 Uncorrectable Error Handling

If there is an uncorrectable error (UE) for a translate or a load operation, the instruction finishes with a machine check indication to the ISU. The instruction is flushed and re-executed without generating any machine check, and a counter is maintained to see how many UEs occurred. If the UE occurs more than a threshold, a machine-check interrupt is taken. For caching-inhibited load operation, a machine-check interrupt is taken on the first occurrence of the UE.

For the instruction side (I-side), if an instruction is executed and in the nonspeculative path, only then is it treated as a UE. Otherwise, the I-side UE handling mechanism is similar to the D-side.

The core provides the EA of the LSU operation that caused the UE in the DAR register. For a UE detected by the IFU for instruction fetches, SRR0 is set to the EA.

Table 4-31 summarizes how the POWER9 processor handles parity, multi-hit, and unrecoverable errors.

Table 4-31. Summary of POWER9 Behavior on Parity Error, Multi-Hit, and Uncorrectable Error (Sheet 1 of 2)

Structure	Parity Error	Multi-Hit	Both Parity Error and Multi-Hit	Uncorrectable Error (UE)
SLB: I-side translation	MC, SRR1, SRR0	MC, SRR1, SRR0	MC, SRR1, SRR0	N/A
SLB: D-side translation, SLBFEE, MFSLB	MC, DSISR, DAR	MC, DSISR, DAR	MC, DSISR, DAR	N/A
TLB: I-side translation	IRR, HMI	MC, SRR1, SRR0	IRR, HMI	N/A
TLB: D-side translation, MFTLB	IRR, HMI	MC, DSISR, DAR	IRR, HMI	N/A

Notes:
- SRR0, SRR1, DSISR, DAR are various SPRs set on a machine-check interrupt.
- In the TLB, a multi-hit cannot generate a parity error, but a parity error can generate a multi-hit. In the SLB and D-ERAT, a multi-hit probably generates a parity error.

Table 4-31. Summary of POWER9 Behavior on Parity Error, Multi-Hit, and Uncorrectable Error (Sheet 2 of 2)

Structure	Parity Error	Multi-Hit	Both Parity Error and Multi-Hit	Uncorrectable Error (UE)
D-ERAT	IRR, HMI	MC, DSISR, DAR	IRR, HMI	N/A
Tablewalk: I-side initiated	IRR, HMI	N/A	N/A	MC, SRR1, SRR0
Tablewalk: D-side initiated	IRR, HMI	N/A	N/A	MC, DSISR, DAR
Load	IRR, HMI	N/A	N/A	MC, DSISR
CI Load	MC, DSISR	N/A	N/A	MC, DSISR
Store	IRR, HMI	N/A	N/A	MC, DSISR
Instruction fetch	IRR, HMI	N/A	N/A	MC, SRR1, SRR0
Any other structure (I-ERAT, other Register file, I-cache, D-cache and other SRAMs, data-flow hardware control checker)	IRR, HMI	N/A	N/A	N/A

Notes:
- SRR0, SRR1, DSISR, DAR are various SPRs set on a machine-check interrupt.
- In the TLB, a multi-hit cannot generate a parity error, but a parity error can generate a multi-hit. In the SLB and D-ERAT, a multi-hit probably generates a parity error.

4.10.25.5 TLB Parity Error and Multi-Hit Action

Parity = 0 and Multi-hit = 0: No action.

Parity = 1 and Multi-hit = 0: Parity error detected, IRR followed by HMI (no machine check).

Parity = 0 and Multi-hit = 1: This case is probably caused by software setting up two TLB entries pointing to the same VSID.

Parity = 1 and Multi-hit = 1: Probably multiple bits flipped due to a soft-error that caused the parity error but also made two VSIDs look the same. The POWER9 core does IRR and then HMI.

4.10.26 Interrupts

4.10.26.1 Interrupt Vectors

Exceptions implemented in the POWER9 core are listed in *Table 4-32*.

Table 4-32. Interrupt Vectors

Exception Type	Exception Value
System Reset	x'00100'
Machine Check	x'00200'
Data Storage Interrupt (DSI)	x'00300'
Data Segment Interrupt	x'00380'
Instruction Storage Interrupt (ISI)	x'00400'
Instruction Segment Interrupt	x'00480'
External Interrupt	x'00500'
Alignment Interrupt	x'00600'
Program Interrupt	x'00700'
Floating-Point Unavailable	x'00800'
Decrementer Interrupt	x'00900'
Hypervisor Decrementer Interrupt	x'00980'
Directed Privileged Doorbell Interrupt	x'00A00'
Reserved	x'00B00'
System Call	x'00C00'
Trace Interrupt	x'00D00'
Hypervisor Data Storage Interrupt (HDSI)	x'00E00'
Hypervisor Instruction Storage Interrupt (HISI)	x'00E20'
Hypervisor Emulation Assistance Interrupt	x'00E40'
Hypervisor Maintenance Interrupt	x'00E60'
Directed Hypervisor Doorbell Interrupt	x'00E80'
Hypervisor Virtualization Interrupt	x'00EA0'
Performance Interrupt	x'00F00'
VMX Unavailable Interrupt	x'00F20'
VSX Unavailable Interrupt	x'00F40'
Facility Unavailable Interrupt	x'00F60'
Hypervisor Facility Unavailable Interrupt	x'00F80'
Soft Patch Interrupt	x'01500'
Debug Interrupt	x'01600'

4.10.26.2 Alternate Interrupt Location

Table 4-33 summarizes the alternate interrupt location (AIL) effects on interrupt processing when IR and DR have the same value before the interrupt occurs. If IR and DR have different values before the interrupt occurs, the interrupts are taken as if AIL = '00' per the *Power ISA (Version 3.0B)*.

Table 4-33. AIL Effects on Interrupt Processing (IR = DR)

Interrupt Name	Initial HV	Sets HV = 1?	IR = DR before Interrupt	AIL Value	HPT Behavior	Radix Behavior
Machine Check, SRI, HMI	X	Yes	X	0, 2, 3	IR/DR = 0 No offset added	IR/DR = 0 No offset added
System call vectored	X	No	0	0, 2, 3	IR/DR = 0 Normal effective address (00 - 0001_7xxx)	IR/DR = 0 Normal effective address
	X	No	1	0	IR/DR = 0 Normal effective address (00 - 0001_7xxx)	IR/DR = 0 Normal effective address
	X	No	1	2	IR/DR = 1 Normal effective address (00 - 0001 7xxx)	IR/DR = 1 Normal effective address
	X	No	1	3	IR/DR = 1 Alternate effective address	IR/DR = 1 Alternate effective address
Interrupts that (can) set HV = 1: sc(LEV = 1), ext(LPES = 0), program (priv, evirt=1), hypervisor emulation assistance interrupt (HEAI), hypervisor doorbell, hypervisor decrementer, hypervisor data storgage interrupt (HDSI), hypervisor instruction storage interrupt (HISI), hypervisor virtualization, hypervisor facility unavailable	X	Yes	X	0	IR/DR = 0 No offset added	IR/DR = 0 No offset added
	X	Yes	0	2, 3	IR/DR = 0 No offset added	IR/DR = 0 No offset added
	0	Yes	1	2, 3	IR/DR = 0 No offset added	IR/DR = 1 Offset
	1	Yes	1	2, 3	IR/DR = 1 Offset	IR/DR = 1 Offset
Interrupts that preserve HV: DSI, ISI, data/instruction segment, alignment, FP/vector/VSX/facility unavailable, sc (LEV = 0), trace, performance monitor, ext (LPES = 1), program (other), decrementer, privilege doorbell	X	No	X	0	IR/DR = 0 No offset added	IR/DR = 0 No offset added
	X	No	0	2, 3	IR/DR = 0 No offset added	IR/DR = 0 No offset added
	X	No	1	2, 3	IR/DR = 1 Offset	IR/DR = 1 Offset

4.10.26.3 Interrupt Definitions

Table 4-34 lists the implemented MSR and SRR1/HSRR1 bits.

Table 4-34. Implementation MSR and SRR1/HSRR1 Bits (Sheet 1 of 2)

Bits	MSR	SRR1/HSRR1
0	SF	SF
1	Reserved	Reserved
2	Not Implemented	Not Implemented
3	HV	HV
4	Not Implemented	Not Implemented
5	Reserved	Reserved
6:28	Not Implemented	Not Implemented
29:30	TS (Transactional State)	TS (Transactional State)
31	TM (Transactional Memory Available)	TM (Transactional Memory)
32	Not Implemented	Not Implemented
33	Not Implemented	Specific Interrupt Information
34	Not Implemented	Not Implemented
35:36	Not Implemented	Specific Interrupt Information
37	Not Implemented	Not Implemented
38	VMX	VMX
39	Not Implemented	Not Implemented
40	VSX	VSX
41	Not Implemented	Not Implemented
42:47	Not Implemented	Specific Interrupt Information
48	EE	EE
49	PR	PR
50	FP	FP
51	ME	ME
52	FE0	FE0
53:54	TE	TE
55	FE1	FE1
56	US	US
57	Not Implemented	Not Implemented
58	IR	IR
59	DR	DR
60	Not Implemented	Not Implemented
61	PMM	PMM
62	RI	RI

Table 4-34. Implementation MSR and SRR1/HSRR1 Bits (Sheet 2 of 2)

Bits	MSR	SRR1/HSRR1
63	LE	LE

4.10.26.4 Synchronous Interrupts

In addition to the synchronous interrupts described in the *Power ISA (Version 3.0B)*, the POWER9 core implements a soft-patch interrupt that can be used by hypervisor-level software to "trap" on particular instructions. When the conditions for trapping on the instruction are met (typically based on the instruction opcode), the soft-patch facility in the hardware generates a soft-patch interrupt and the hardware fetches the interrupt vector located at offset x'01500'. In addition, the hardware updates the Hypervisor Emulation Assist Interrupt Register (HEIR) with the 32-bit Power ISA instruction. In some cases, the hardware modifies bits of the instruction image. A partial list of notable instructions that exhibit this behavior are listed in *Table 4-35*.

Table 4-35. HEIR Instruction Formatting for Branch-Like Instructions

Instruction	Primary Opcode	Secondary Opcode	Additional Fields
sc or **scv**	17		
sp_attn	00	256	
rfscv	19	82	
rfid	19	82	
hrfid	19	274	
stop	19	370	
rfebb	19	146	
mtmsr	31	146	L = 0
mtmsrd	31	178	L = 0
ISTAT errors	N/A	N/A	

Note: For the instructions listed in *Table 4-35*, HEIR[11:14] is set to all '1's. The remaining bits are set as described in the *Power ISA (Version 3.0B)*.

4.10.26.5 Asynchronous Interrupt Priorities

The POWER9 core processes asynchronous interrupts and event-based branches (EBBs) in the following order:

1. System Reset Interrupt
2. Machine Check
3. Imprecise Floating Point Exceptions
4. Hypervisor Maintenance Interrupt
5. Hypervisor Virtualization External Interrupt
6. Mediated External Interrupt
7. Direct External Interrupt
8. Performance Monitor Interrupt

9. Hypervisor Decrementer Interrupt
10. Decrementer Interrupt
11. Hypervisor Doorbell Interrupt
12. Privileged Doorbell Interrupt
13. Event Based Branch

4.10.26.6 System Reset Interrupt

The system reset interrupt is a nonmaskable, asynchronous interrupt that is caused by an SCOM command for a soft reset.

Note: There is no explicit SRESET pin; SRESET must be invoked from the service processor.

Table 4-36. System Reset Interrupt

Register	Bits	Description
SRR0	0:63	Set to the effective address of the instruction that the processor would have.
SRR1	0:31	Implemented bits loaded from the MSR.
	32	Set to '0'.
	33	LPAR mode switch occurred while the thread was in power savings mode.
	35:36	Set to '0'.
	42:45	Interrupt caused by IFU detection of a hardware uncorrectable error (UE) 0000 Reserved by pervasive function. 0010 Interrupt caused by SCOM when not in power-saving mode or caused by back-to-back SRESET. 0011 Interrupt caused by hypervisor door bell. 0101 Interrupt caused by privileged door bell. 0100 Interrupt caused by SCOM when in power-saving mode. 0110 Interrupt caused by decrementer wake-up when in power-saving mode. 1000 Interrupt caused by external interrupt wake-up when in power-saving mode. 1001 Interrupt caused by hypervisor virtualization wake-up when in power-saving mode. 1010 Interrupt caused by HMI wake-up when in power saving mode. 1100 Interrupt caused by implementation-specific wake-up when in power-saving mode.
	46:47	Indicates if the interrupt occurs when the processor is in power-saving mode. 00 Interrupt did not occur while the processor was in power-saving mode. 01 Interrupt occurred while the processor was in power-saving mode. The state of all resources was maintained as if the processor was not in power-saving mode 10 Interrupt occurred while the processor was in power-saving mode. The state of some resources was not maintained but the state of all hypervisor resources, including TB, PURR, and SPURR, was maintained as if the processor was not in power-saving mode and the state of all other resources is such that the hypervisor can resume execution. 11 Interrupt occurred while the processor was in power-saving mode. The state of some resources was not maintained, and the state of some hypervisor resources was not maintained or the state of some resources is such that the hypervisor cannot resume execution.
	62	Loaded from MSR[62] if recoverable. Otherwise set to zero
	Others	Implemented bits loaded from MSR.

The POWER9 core implements a 1-deep queue to remember the reason of a subsequent system reset interrupt while a system reset interrupt is pending. The reason of the most important subsequent system reset interrupt is remembered per the following priority:

1. Hypervisor doorbell-initiated system reset
2. Privileged doorbell-initiated system reset
3. SCOM-initiated system reset
4. HMI-initiated system reset
5. External-initiated system reset
6. Decrementer-initiated system reset
7. Implementation-specific initiated system reset

4.10.26.7 Machine Check Interrupt

The are several possible causes of machine check interrupts in the POWER9 chip, some of which are generally recoverable and some of which are non-recoverable.

The following causes of machine check interrupts are precise and synchronous with the instruction that caused the operation which encountered the error (that is, SRR0 contains the address of the instruction that caused the operation).

1. The detection of either a parity error, or a multi-hit error, or both in the SLB during the execution of a load, store **slbfee**, or **mfslb** instruction. If the interrupt is caused by a soft error, executing the appropriate sequence of instructions in the machine-check handler program clears the error condition without causing any loss of state, permitting the interrupted program to be resumed if MSR[RI] was a '1' when the instruction that encountered the error was executed.
2. If there is a multi-hit in the D-ERAT or TLB, the core finishes the operation with a machine-check interrupt and sets the proper DSISR bit to indicate where the multi-hit occurred.
3. If there is an uncorrectable ECC error when a load instruction is executed or when the page table is being searched in the process of translating an address, the instruction finishes with a machine-check indication to the instruction-sequencing unit. The instruction is flushed and re-executed without generating any machine check. A counter is maintained to see how many UEs occurred. If the UE occurs more than a pre-established threshold, a machine-check interrupt is taken.
4. For a caching-inhibited load operation, the machine-check interrupt is taken on the first occurrence of the UE.
5. For the I-side, if an instruction is executed and the instruction is in the nonspeculative path, only then will it be treated as a UE. Otherwise, the I-side UE handling mechanism is similar to the D-side.
6. For the I side, when an instruction fetch causes an out-of-range real address (L2 address error) or a foreign link times out, a machine check interrupt is taken.
7. For the D side, when a load causes an out-of-range real address (L2 address error) or a foreign link times out, a machine check interrupt is taken.
8. For the I and D side, if a tablewalk fetch causes an out-of-range address (L2 address error) or foreign link times out, a machine check interrupt is taken.
9. Anytime that a **tlbie** or **tlbiel** instruction has either an instruction encoding or an unsupported page-size encoding that is not supported by the hardware. The **tlbie**(**l**) instruction encoding cases are outlined *Appendix B* on page 469 and *Appendix C* on page 485. The unsupported page-size cases correspond to any page-size encodings not specified in *Table 4-20* on page 106,*Table 4-24* on page 110, *Table 4-25* on page 110, or *Table 4-26* on page 111.
10. During translation, if a host real address is in the foreign address range (bits 8:12 are not zeros).

11. If an instruction fetch is in the foreign address range (bits 8:12 are not zeros).

In the cases described in items (2), (3), (4), and (5), no state is lost in the processor, but recovery of the correct data might not be possible.

For more traumatic errors or hard errors, these characteristics cannot be reliably provided on a machine check because it is likely that the failure will prevent reliable execution. Additionally, a machine-check interrupt that occurs when MSR[ME] = '0' results in a checkstop.

The following are invalid forms for **tlbie** that result in a machine-check interrupt:

- PRS = 1 and R = 0 and RIC ≠ 2
- RIC = 1 and R = 0
- RIC = 3 and R = 1
- RIC = 1 and IS = 0
- RIC = 2 and IS = 0
- RIC = 3 and IS ≠ 0
- PRS = 0 and IS = 1
- R = 0 and IS = 1 and RIC ≠ 2
- L = 1 and IS ≠ 0
- L = 1 and R = 1

The following are invalid forms for **tlbiel** that result in a machine check interrupt:

- PRS = 1 and R = 0 and RIC ≠ 2
- RIC = 1 and R = 0
- RIC = 3
- RIC = 1 and IS = 0
- RIC = 2 and IS = 0
- PRS = 0 and IS = 1
- R = 0 and IS = 1 and RIC ≠ 2
- L = 1 and IS ≠ 0
- L = 1 and R = 1

In the POWER9 core, there are two asynchronous machine-check interrupts. One is taken when a store instruction has an out-of-range real address associated with it. This is in general a programming error. The core takes a machine check to help in debugging bad code. The second asynchronous case is when a store is being performed and a foreign link times out. Again this presents an asynchronous machine check, both machine checks set bits in the SRR1 to identify the cause. A machine-check interrupt is taken when the machine-check input pin is asserted. The FIR, debug logic, and hang recovery logic can also be programmed to induce machine check interrupts for various error conditions. In general, the POWER9 core works hard to make these interrupts recoverable, but there are some scenarios where it cannot achieve this. Software can use the MSR[RI] bit to help identify the cases where the machine-check interrupt is recoverable.

Information about the suspected source of the error condition is logged into either the SRR1 Register, the DSISR Register, or both as defined in *Table 4-37* on page 132 for synchronous and asynchronous machine checks.

Table 4-37. Synchronous Machine Checks (Sheet 1 of 2)

Register	Bits	Description
SRR0	0:63	Effective address of the next instruction that would have executed if the machine-check interrupt was not taken. For cases where this is a recoverable machine check due to a load that has surfaced an error, this will be the address of the load instruction itself. (The POWER9 core allows the instruction to execute to surface the error, but inhibits the commitment of the results.) For cases where this is a recoverable machine check due to an instruction fetch surfacing an error, this will be the address of an instruction that initiated the memory/cache access. for asynchronous machine checks this address is meaningless
SRR1	42	Interrupt caused by load/store detection of error (see DSISR).
	36, 43:45	Interrupt caused by an instruction fetch, indicated by the following encoding: 0000 Reserved. 0001 Interrupt caused by a hardware uncorrectable error detected while doing an instruction fetch (but not translation related). 0010 Interrupt caused by an SLB parity error while translating an instruction fetch address. 0011 Interrupt caused by an SLB multiple hit, while translating an instruction fetch address. **Note**: This condition occurs if the ESID fields of two or more SLB entries contain the same value. 0100 Interrupt caused by an I-ERAT multi-hit error. 0101 Interrupt caused by a TLB multi-hit error detected while translating an instruction fetch address. Note: This condition occurs if an address is mapped to both a small and large page in the SLB. This condition can also occur due to a software bug, when a software-managed TLB mechanism is used. 0110 Interrupt caused by a hardware UE detected while doing a TLB reload for the I-side. 0111 Instruction fetch to foreign address space 1000 Interrupt caused by an L2 abort on an instruction fetch due to foreign link time out. 1001 Interrupt caused by an L2 abort on an instruction tablewalk due to foreign link time out. 1010 Reserved. 1011 Real address (CResp) error for an instruction fetch 1100 Real address (CResp) error for an instruction fetch tablewalk 1101 Asynchronous machine check due to a real address (CResp) error from a store 1110 Asynchronous machine check due to a foreign link time out (nest abort) due to a store instruction. 1111 I-side tablewalk used a host real address in the foreign address range
	62	Loaded from MSR[62] if recoverable. Otherwise, set to zero.
	others	Implemented bits loaded from MSR.

Table 4-37. Synchronous Machine Checks (Sheet 2 of 2)

Register	Bits	Description
DSISR	32:47	All zeros.
	48	Interrupt caused by a UE deferred error, but not for a tablewalk (D-side only).
	49	Interrupt caused by a UE deferred error during a tablewalk (D-side).
	50	Foreign Link Time out (Nest abort) for a load.
	51	Foreign Link Timeout (Nest abort) for tablewalk.
	52	Interrupt caused by a D-ERAT multi-hit.
	53	Interrupt caused by a TLB multi-hit due to translation (D-side only) or MFTLB operation.
	54	Tlbie or tlbiel programming error.
	55	Interrupt caused by an SLB parity error (translate lookup or **mfslbfee**) due to a translation (D-side only), **slbfee**, or **mfslb** instruction.
	56	Interrupt caused by an SLB multi-hit (might not be recoverable) for translation (D-side only), **slbfee**, or **mfslb** instruction.
	57	Bad real address (CResp) for a load.
	58	Bad address (CResp) for a load or store tablewalk address.
	59	Host real address to foreign space during translation
	60	Host real address to foreign space on a load or store access
	61:63	Set to zeros.
DAR	0:63	Effective address computed by a load or store instruction that caused the operation that encountered a parity error, or multi-hit, or both in the SLB, or which encountered a multi-hit in the TLB, or encountered a multi-hit in the D-ERAT, or encountered a UE while attempting to reload a TLB entry. For all other types of machine check interrupts, the DAR is undefined (including the case where the operand of the load instruction contains a UE). 1. SLB parity error, multi-hit, or both: DAR is loaded with the EA of the target of the load or store instruction that caused the error. 2. TLB multi-hit: DAR is loaded with the EA of the target of the load or store instruction that caused the error. 3. D-ERAT multi-hit: DAR is loaded with the EA of the target of the load or store instruction that caused the error. 4. UE on D-side tablewalk: DAR is loaded with the EA of the target of the load or store instruction. 5. UE on instruction fetch: DAR is undefined. 6. UE on I-side tablewalk: DAR is undefined. 7. UE on load or store instruction: DAR is undefined (EA is not available in LMQ for loads; therefore, DAR cannot be loaded). 8. CResp for load: DAR is set to the EA of the load that caused the error 9. CResp for a dside table-walk: DAR is undefined 10. Host real address to foreign space: DAR is undefined 11. Tlbie or tlbiell programming error: DAR is Undefined 12. Asynch Machine cheks: DAR is not modified

DSISR Implementation Note: All the bits have been implemented in hardware.

Machine-Check Interrupt Handler Notes:

As mentioned previously, the machine check interrupt handler is expected to help hardware recover from certain types of D-ERAT, TLB, and SLB errors detected by the hardware. In general terms, the interrupt handler must check whether or not the machine check interrupt is recoverable (by looking at the state of the RI bit in SRR1). It must determine the type of error that caused the machine check (by looking at the state of the SRR1 and DSISR Registers). It must flush the contents of the array that reported the detected error (this process is slightly different for each of the possible arrays). Finally, it must return to the interrupted process.

4.10.26.8 Hypervisor Maintenance Interrupt

The POWER9 hypervisor maintenance interrupt is implemented to replace the malfunction alert and thermal interrupt; and to provide support for recovery function. The HMER Register contains the sources of the interrupt, which can be masked by setting the HMEER enable bits to zero. For successful recovery, HMER setting indicates successful recovery.

4.10.26.9 External Interrupt

An external interrupt is classified as being either a direct external interrupt or a mediated external interrupt. Both cause an interrupt to x'500'.

Direct External Interrupt

The direct external interrupt is signaled by the assertion of the external interrupt input signal. The external interrupt signal must remain asserted until the processor has actually taken the interrupt. Failure to meet this requirement can lead the processor to not recognize the interrupt request.

When LPES = '0', the following registers are set.

Table 4-38. Direct External Interrupt (LPES = '0')

Register	Bits	Description
HSRR0	0:63	Set to the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present.
HSRR1	33:36	Set to '0'.
	42:47	Set to '0'.
	Others	Loaded from the MSR.
MSR	–	See the Power ISA.

When LPES = '1', the following registers are set.

Table 4-39. Direct External Interrupt (LPES = '1') (Sheet 1 of 2)

Register	Bits	Description
SRR0	0:63	Set to the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present.

Table 4-39. Direct External Interrupt (LPES = '1') (Sheet 2 of 2)

Register	Bits	Description
SRR1	33:36	Set to '0'.
	42:47	Set to '0'.
	Others	Loaded from the MSR.
MSR	–	See the Power ISA.

Mediated Exernal Interrupt

Mediated external interrupts are caused by the LPCR[MER] = '1', when the thread is in privileged (supervisor) or problem state mode.

When LPES = '0', the following registers are set.

Table 4-40. Mediated External Interrupt (LPES = '0')

Register	Bits	Description
HSRR0	0:63	Set to the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present.
HSRR1	33:36	Set to '0'.
	42	Set to '1'.
	43:47	Set to '0'.
	Others	Loaded from the MSR.
MSR	–	See the Power ISA.

When LPES = '1', the following registers are set.

Table 4-41. Mediated External Interrupt (LPES = '1')

Register	Bits	Description
SRR0	0:63	Set to the effective address of the instruction that the thread would have attempted to execute next if no interrupt conditions were present.
SRR1	33:36	Set to '0'.
	42:47	Set to '0'.
	Others	Loaded from the MSR.
MSR	–	See the Power ISA.

4.10.26.10 Alignment Interrupt

See *Section 4.1.4.1 Alignment Interrupts* on page 54 for details on when the POWER9 core takes alignment interrupts. The DAR is updated on an alignment interrupt as described in the *Power ISA (Version 3.0B)*. The DSISR register is not updated on an alignment interrupts per the *Power ISA (Version 3.0B)*.

4.10.26.11 Trace Interrupt

In general, a trace interrupt is taken after every instruction when the MSR[TE] = '10' and after every branch instruction when MSR[TE] = '01'. In particular, for the case where MSR[TE] = '10' before a **mtmsr**[**d**] instruction is executed that alters the MSR[TE] bits, a trace interrupt also occurs. There are several instructions and conditions for which the preceding statements are not followed. See the *Power ISA (Version 3.0B)* for details. Additionally, a trace interrupt is taken when a CIABR match occurs. After a trace interrupt is taken, SRR0, SRR1, SIAR, and SDAR are set as shown in *Table 4-42*.

Table 4-42. Trace Interrupt

Register	Bits	Description
SRR0	0:63	Set as specified in the architecture.
SRR1	0:32	Implemented bits loaded from the MSR.
	33:34	'10'
	35	Set for a load instruction; otherwise, cleared.
	36	Set for a store instruction; otherwise, cleared.
	37:41	Loaded from the MSR.
	42	Loaded from the MSR, which is an unimplemented bit (therefore, always set to '0').
	43	Set to a '1' if a CIABR trace.
	44:47	Set to '0'.
	48:63	Implemented bits loaded from the MSR.
	Note: Bit 35 and 36 are not set if an X-form load string or store string instruction specifies an operand length of 0.	
SIAR	0:63	Set to the effective address of the traced instruction; undefined if a CIABR trace.
SDAR	0:63	If the instruction that took the trace interrupt was a storage access instruction, the SDAR is set to the effective address of the storage access. SDAR is not set if an X-form load string or store string instruction specifies an operand length of 0; undefined if a CIABR trace.

The contents of SIAR and SDAR are undefined until a trace interrupt occurs.

4.10.26.12 Performance Monitor Interrupt

The performance monitor interrupt is signaled when the MSR[EE] bit is set, the MMCR0[PMAE] bit is set, and any of the performance monitor counters overflow (this includes the eight performance counters defined in the SPR space, as well as the counters defined in MMIO space for the nest).

After such an event is detected, the POWER9 core waits for previously dispatched instructions to complete, and then takes the interrupt.

4.10.26.13 Facility Unavailable Interrupt

The POWER9 core implements the facility unavailable interrupt as defined in the Power ISA.

4.10.26.14 Hypervisor Emulation Assistance Interrupt

The POWER9 core implements the hypervisor emulation assistance interrupt as defined in the Power ISA. However, the contents of the HEIR for the following instructions differ from the 32-bit Power ISA instruction described as follows:

mfbhrbe - Power ISA instruction bits 11:20 are recoded to indicate the internal SPR number assigned to each BHRB entry. SPR d'80' is assigned if the specified entry is outside of the available range. This instruction is sensitive to PCR and therefore can be made illegal.

clrbhrb - Power ISA instruction bits 11:20 are recoded to indicate the internal SPR address d'80'. This instruction is sensitive to PCR and therefore can be made illegal.

bctar, **bctarl -** This is what a recoded **bctar** looks like in the HEIR after an illegal instr interrupt, where '-' can be either 0 or 1.

```
0         1         2         3

01234567890123456789012345678901

0-100------11-0----01--10110000-
```

This should only match the following ops, none of which go into HEIR. Therefore, assume **bctar** if the above compare matches:

```
ori_nop
ori.0
ori.1
oris.0
oris.1
subfic.0
subfic.1
```

The following instruction fields can be found in the indicated HEIR bits. Note that bo(4) is lost.

```
lk      = HEIR(1)
bo(0:3) = HEIR(5:8)
bo(4)   = '0'
bi(0)   = HEIR(10)
bi(1:4) = HEIR(15:18)
bh(0:1) = HEIR(21:22)
```

Stop fetch instructions - The following instructions have Power ISA instruction bits 11:14 recoded to all '1's to indicate the "stop fetch' function:

```
scv
rfscv
sp_atten
rfebb
```

4.10.27 Logical Partitioning (LPAR) Support

Each thread on the POWER9 core is assigned to its own logical partition (LPAR). The associated architected SPRs are replicated on a per thread basis. See *Table 4-8* on page 83 for details on which SPRs are replicated per thread per LPAR.

4.10.28 Strong Access Ordering Mode (SAO)

The POWER9 core supports the SAO mode defined in Power ISA.

4.10.29 Graphics Data Stream Support

For cache-inhibited stores, the POWER9 core provides store gathering with an intentional stall to maximize the amount of gathering that can occur.

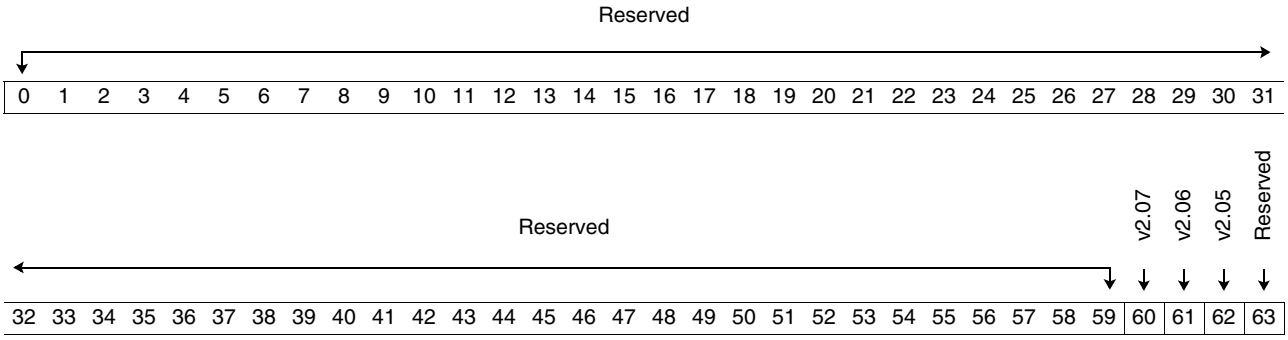
4.10.30 Performance Monitoring, Sampling, and Trace

Performance monitoring facilities have been incorporated into the POWER9 processor to enable the collection of performance related data and instruction traces. In general, the POWER9 core supports the recommended architecture for performance monitoring as described in the Power ISA.

4.10.31 Processor Compatibility Mode

The POWER9 core implements the Processor Compatibility Register (PCR) as described in the Power ISA to facilitate partition migration. Setting PCR[60] = '1' disables all problem state instructions and facilities that were added in *Power ISA (Version 3.0B)*. Thus, setting this bit effectively makes a POWER9 core architecturally appear to problem state software as a Power ISA version 2.07 core (that is, a POWER8 core). Setting PCR[61] = '1' disables all problem state instructions and facilities that were added in Version 2.07 of the Power ISA. Thus, setting both bits 60 and 61 effectively makes a POWER9 core architecturally appear to problem state software as a *Power ISA* version 2.06 core (that is, a POWER7 core). Likewise, setting PCR[62] = '1' disables all problem state instructions and facilities that were added in Version 2.06 of the Power ISA. Therefore, to migrate a partition from a version 2.05 system to a POWER9 (Version 3.0) system, PCR[60:62] must be set to '111'.

Unlike the POWER8 processor, there is no requirement to flush the I-cache using HID[2] after changing the state of the PCR with an **mtspr_PCR**.



5. Simultaneous Multithreading

5.1 Overview

The POWER9 processor core supports ST, SMT2, and SMT4 modes. Any thread number can be run in any SMT mode, on any thread set. *Table 5-1* shows the SMT mode definitions.

Table 5-1. SMT Modes

Description	Number of Threads Enabled	Switch to this SMT mode when ...
ST	0 - 1	POR state
SMT2	1 - 2	2 threads
SMT4	1 - 4	3 - 4 threads

5.2 Partitioning of Resources in Different SMT Modes

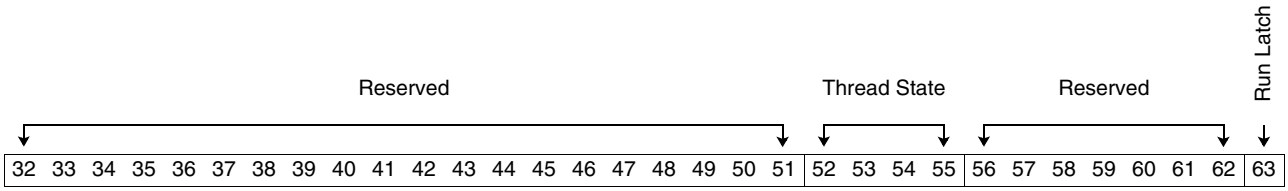
Table 5-2 lists the resources that are partitioned in certain SMT modes.

Table 5-2. Front-End Execution Core Resource

Resource	ST	SMT2	SMT4
EAT Entries	40	20 per thread	10 per thread
Instruction Buffer Entries	96	48 per thread	24 per thread
Link Stack	32	32 per thread	16 per thread
D-ERAT Entries	64	64 shared	64 shared
Dispatch Groups	6	6 with cycle toggle between threads	3 - 3 wide dispatch that each thread set has a cycle toggle
History Buffers VR/GPR/FPR renames	$4 \times (20 + 96)$	Both threads share the total	$2 \times 2(20 + 96)$ threads within the thread set share
Unified Issue Queue Entries	52	52 shared	26 per thread set
FXU, VSU, agens	Four combined FXU/FSU Four additional agens	Threads share all units	Thread sets share and each get two combined and two agen units
Completion Rates	Up to 64 instructions per cycle	Up to 64 instructions per cycle for one thread	Up to 64 instructions per cycle for one thread

5.3 Control Register

The Control Register (CTRL) is an architected 32-bit register. The bit assignment for the thread control bits in the CTRL supports up to four threads. The POWER9 core supports threads 0 - 3. A bit in the CTRL Register represents the architected state for a particular thread.



Bits	Field Name	Description
32:51	Reserved	Reserved.
52:55	Thread State	Thread State bits corresponding to threads 0 - 3 (indirectly written by supervisor and hypervisor software) as described in the *Power ISA (Version 3.0B)*.
56:62	Reserved	Reserved.
63	Run Latch	Run Latch for thread doing CTRL read/write (read only/rerouted supervisor or hypervisor write).

The CTRL Register can be read with the **mfspr** instruction using SPR 136 in user, supervisor, or hypervisor state.

The CTRL Register can be selectively written with the **mtspr** instruction using SPR 152 in the supervisor or hypervisor state.

The CTRL Register is initialized to x'0000_0000_0000_0000' at power-on.

Even though a single CTRL Register is shared by the four threads, there is no need to obtain a lock before updating the CTRL Register. There is only one bus that goes to the core pervasive unit, and the instruction issue logic serializes all **mtctrl** instructions. Updating the Run Latch bit must be done in hypervisor mode. When updating the Run Latch bit (in hypervisor mode), the software is recommended to set the Thread State bits to '0000'. Setting the Thread State bits to '0000' is not allowed. Therefore, this updates the run latch, but there is no effect to the Thread State bits and no thread will be killed or woken up.

CTRL[52:55] contain the Run Latches for threads 0 - 3. A **mtspr CTRL** instruction does not modify CTRL[52:55] based on GPR bits [52:55]. Instead, these bits are indirectly loaded by writing a value to CTRL[63]. The value written to CTRL[63] is loaded into CTRL[52] if thread 0 issued the move to CTRL and CTRL[53] if thread 1 issued the move to CTRL, and so on. A thread cannot update the thread Run Latch bit of another thread.

The run latch bit is only used by software for status and is sent to the performance monitor for performance analysis. For this purpose, the POWER9 processor core supports one run latch per thread. To use this function, if a thread is executing a dispatchable task, software must set the CTRL Run Latch bit for that thread to '1' by writing a '1' to CTRL[63]. If a thread is in a wait state, waiting for a dispatchable task, software must set the CTRL Run Latch for that thread to '0'.

Software can load the CTRL Register with a Run Latch value for its thread by writing the Run Latch value to CTRL[63]. Hardware routes data directed to CTRL[63] into the corresponding CTRL[52] to CTRL[55] bit, depending on which thread is doing the write (see previous definition of CTRL[52:55]). When the CTRL Register is read, data driven on CTRL[63] comes from CTRL[52] to CTRL[55], depending on which thread is doing the read. CTRL[63] does not physically exist in hardware.

The data read on a **mfspr**(CTRL) is formatted differently based on the MSR[PR] and MSR[HV] bits. Bit 63 is always the Run Latch of the thread executing the **mfspr**. Bits [52:55] are formatted as shown in *Table 5-3*, where R0 equals run latch for thread 0 and RT equals run latch of thread executing **mfspr**.

*Table 5-3. **mfspr** CTRL Data Formatting*

MSR[HV], MSR[PR]		Bits [52:55]
'00'	Privileged	R0, R1, R2, R3
'*1'	Problem	RT, 0, 0, 0
'10'	Hypervisor	R0, R1, R2, R3

5.4 Thread Priority, Status, and Control Requirements

Thread priority, control, and status registers enable software to do the following:

- Give a large percentage of execution resources to critical tasks.
- Reduce the amount of resources and power used by low-priority work.
- Read foreground and background thread priority and status.
- Save and restore priority during interrupts.
- Provide a controlled way to allow supervisor/user code to change priority.
- Provide a means to kill or revive a thread.
- Avoid fine-grain livelock or deadlock situations between threads.

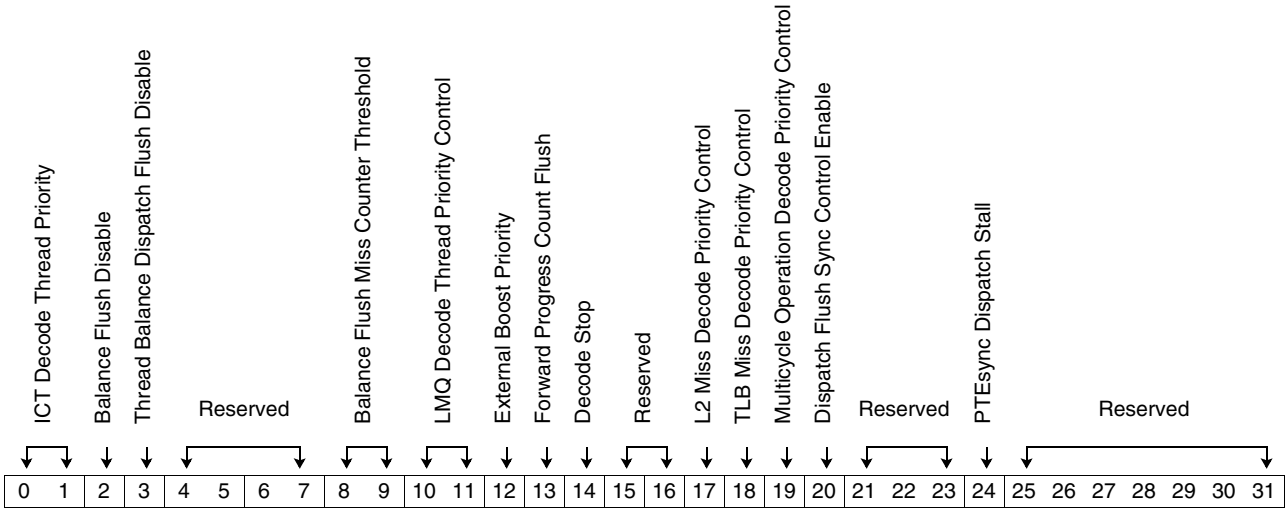
5.5 Thread Balance Control Requirements

The following mechanisms can be used to balance work between threads:

- Reduce ifetch priority of a thread that uses too many resources.
- Reduce decode priority of a thread that uses too many resources.
- Hold decode of a thread with long latency events.
- Dispatch flush decode pipe to clean congested operations.
- Balance flush from next-to-complete plus one group and hold at IBUF until a miss resolves.

5.6 Thread Switch Control Register (Hypervisor Access Only)

Thread priority controls are programmable. All bits are read/write. There is one Thread Switch Control Register (TSCR) per core. TSCR is initialized to x'0000_0000' at power-on.



Bits	Field Name	Description
0:1	ICT Decode Thread Priority	ICT Decode Thread Priority. If all threads are at the same software-thread priority, decrease the priority when a thread uses more than the following number of ICT entries: 00 Function disabled. 01 SMT2: 80; SMT4: 40. 10 SMT2: 96; SMT4: 48. 11 SMT2: 112; SMT4: 56.
2	Balance Flush Disable	Balance Flush Disable. 0 Enable NTCP1 balance flushes. 1 Disable NTCP1 balance flushes.
3	Thread Balance Dispatch Flush Disable	Thread Balance Dispatch Flush Disable. 0 Enable dispatch flush for the thread that was chosen for a balance flush if that thread is stalled at dispatch. A dispatch flush is a lower-latency flush than a balance flush. 1 Disable. **Note:** Conditions for dispatch flush are the same as a balance flush.
4:7	Reserved	Reserved.
8:9	Balance Flush Miss Counter Threshold	Balance Flush Miss Counter Threshold. If a balanced flush occurs, apply a CLB hold until the counter threshold is released or the miss is resolved. 00 Function is disabled, CLB hold is applied until the miss is resolved. 01 Scan-only latch, value is programmable, 10-bit LFSR. POR default is: 256 cycles (x'3C1' LFSR). 10 Scan-only latch, value is programmable, 10-bit LFSR. POR default is: 384 cycles (x'0E4' LFSR). 11 Scan-only latch value programmable, 10-bit LFSR. POR default is: 512 cycles (x'20F' LFSR).

Bits	Field Name	Description
10:11	LMQ Decode Thread Priority Control	LMQ Decode Priority Control. When the threads have the same software-set decode priority, decrease the thread's priority if a thread has more misses outstanding than described as follows: 00 Function disabled. 01 SMT2: 4; SMT4: 2. 10 SMT2: 5; SMT4: 3. 11 SMT2: 6; SMT4: 4.
12	External Boost Priority	External Boost Priority. If '1' and an external interrupt request is active and the corresponding threads' priority is less than normal priority, set the threads' priority to normal. **Note:** This does not change the value in PPR[11:13] for the affected thread.
13	Enable Forward Progress Count Flush	Enable Forward Progress Count Flush. **Note:** This bit only enables/disables the flush from occurring. The forward progress timer does not stop decrementing when set to '0'. SMT2 and higher: If one thread is not making progress, enable flushing the other active threads.
14	Decode Stop	Decode Stop. When set to '0', the forward progress timer (in PPR) is decremented even when the current thread is in decode stop state. When set to '1', the forward progress timer is not decremented when the current thread is in decode stop state.
15:16	Reserved	Reserved.
17	L2 Miss Decode Priority Control	L2 Miss Decode Priority Control. If all threads are at the same software set priority, then: 0 L2 miss is disabled for use in adjusting decode priority. 1 L2 miss is enabled for use in adjusting decode priority.
18	TLB Miss Decode Priority Control	TLB Miss Decode Priority Control. If all threads are at the same software set priority, then: 0 TLB miss is disabled for use in adjusting decode priority. 1 TLB miss is enabled for use in adjusting decode priority.
19	Multicycle Operation Decode Priority Control	If all threads are at the same software-set priority, then: 0 Multicycle operations are disabled for use in adjusting decode priority. 1 Multicycle operations are enabled for use in adjusting decode priority.
20	Dispatch Flush Sync Control Enable	Dispatch Flush Sync Control Enable. Stop decode if the mode for the thread with **sync** instruction is outstanding, (always on in shipping mode). Applies only to SMT2 and higher.
21:23	Reserved	Reserved.
24	PTEsync Dispatch Stall	Ptesync dispatch stall. Set to '1' to enable the following function for **ptesync** instruction. Hold the **ptesync** instruction at dispatch until LMQ is empty, SRQ is empty, no I-side table walk pending, and wait for a minimum of 15 cycles. After these conditions are satisfied, dispatch the **ptesync**. Afterwards, wait for LMQ empty and SRQ empty to continue with the dispatch of further instructions.
25:31	Reserved	Reserved.

The TSCR can be accessed with the **mtspr** and **mfspr** instructions using SPR 921.

An SCOM latch is added to make this register as read-only for debug purposes.

TSCR is initialized to x'0000_0000' at power-on.

The preferred setting is: x'802C_7880'.

5.7 Thread Time-Out Register (Hypervisor only)

The Thread Time-Out Register (TTR) is used to ensure forward progress. There is one TTR per core. For more information see *Section 5.9 Forward Progress Timer* on page 146.

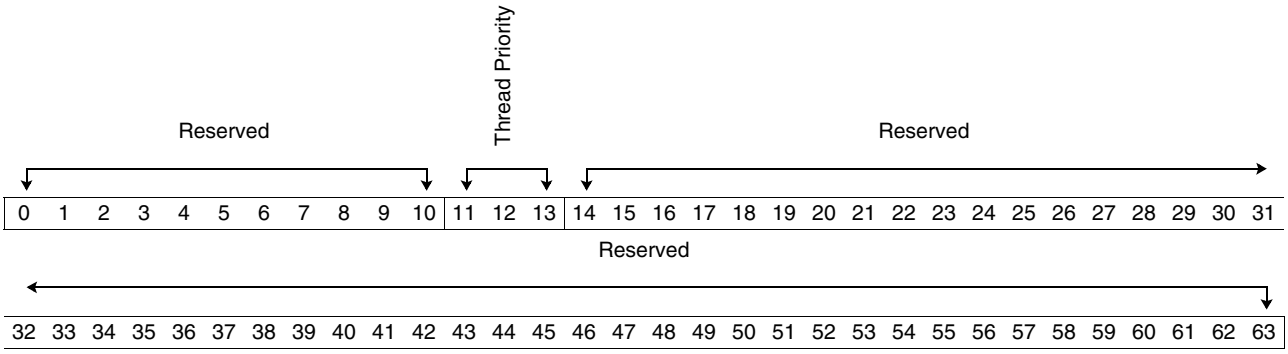
Bits	Field Name	Description
0:43	Reserved	Reserved.
44:63	Thread Time-out Flush Value.	Thread Time-out Flush Value. A x'00000' value generates a maximum count.

The TTR can be accessed with the **mtspr** and **mfspr** instructions using SPR 922.

The TTR is initialized to x'0000_0000_0000_0000' at power-on.

5.8 Program Priority Register (PPR)

Each thread has a 64-bit status register associated with it. Some bits are read-only, while other bits are read/write. There is one PPR per thread.



Bits	Fields Name	Description
0:10	Reserved	Reserved (not implemented).
11:13	Thread Priority	Thread Priority. 000 Not allowed 001 Very low 010 Low 011 Medium low 100 Normal 101 Medium high 110 High 111 Extra high Set to '100' on system reset interrupt.
14:63	Reserved	Reserved (not implemented).

The local PPR can be accessed with the **mtspr** or **mfspr** instructions using SPR 896.

The PPR for each thread is initialized to x'0010_0000_0000_0000' at power-on.

5.9 Forward Progress Timer

For the POWER9 core, the forward progress timer bits were moved out of the PPR Register. A nonarchitected latch bank holds these bits. For the POWER9 core, PPR[44:63] are reserved, non-implemented bits.

The forward progress latch bits are loaded from TTR[44:63] every time a group of instructions are retired on the current thread.

If the current thread is not in a decode stop state, the counter is decremented by '1' every time a completion occurs on another thread (see TSCR[14] in *Section 5.6 Thread Switch Control Register (Hypervisor Access Only)* on page 142).

For ST mode:
- Initialized by a scan flush to x'00000' (maximum count)
- Decrementer stops at x'00001' (minimum value)

For SMT2 and higher, a flush of the other active threads occurs when:
- The timer count reaches x'00001'.
- The forward progress count flush is enabled TSCR[13] = '1'.
- The group completes on another thread.

After the threads are flushed, no dispatch slots are given to the flushed thread until one group has completed for the current thread.

5.10 Thread Priority NOPs

The thread switch priority can be read or written by software using the **mfspr** and **mtspr** instruction to the Thread Status Register. Thread priority can also be altered by executing special forms of the **or x,x,x** NOP. The priority is changed upon completion of the operation, provided the function is enabled for the current privilege level. Thread priority can be set as follows:

- On the POWER9 core, problem-state programs can set their priority from very-low to medium priority.
- Supervisor programs can set their thread priority from very-low to high priority.
- Hypervisor code can set all levels.

Table 5-4 describes how to set the thread priority NOPs.

Table 5-4. Thread Priority Nops

Priority NOP/**mtSPR**	PPR[11:13]	Thread Priority	Required Privilege Level to Set Given Thread Priority Value
or 31,31,31 / **mtPPR**[11:13]	'001'	Very Low	Hypervisor, Supervisor, Problem
or 1,1,1 / **mtPPR**[11:13]	'010'	Low	Hypervisor, Supervisor, Problem
or 6,6,6 / **mtPPR**[11:13]	'011'	Medium Low	Hypervisor, Supervisor, Problem
or 2,2,2 / **mtPPR**[11:13]	'100'	Medium (Normal)	Hypervisor, Supervisor, Problem
or 5,5,5 / **mtPPR**[11:13]	'101'	Medium High	Hypervisor, Supervisor, Problem[1]
or 3,3,3 / **mtPPR**[11:13]	'110'	High	Hypervisor, Supervisor
or 7,7,7 / **mtPPR**[11:13]	'111'	Extra High	Hypervisor

1. See *Section 5.12* on page 147.

5.11 Thread Priority Boosting

Hardware typically does not change the thread priority value in the PPR, unless an **mtPPR** or one of the priority changing NOP instructions is committed. However, on the POWER9 core, problem-state programs can change the thread priority value to medium-high ('5') depending on the contents of the Problem-State Priority Boost Register (PSPBR). The problem-state boosting changes the contents of PPR[11:13].

The thread priority can be boosted internally by the hardware (in a software invisible manner) in certain cases (as described in *Section 5.13 Thread Priority Boosting on Asynchronous Interrupt* on page 148) to medium priority ('4'). The boosting of thread priority for pending asynchronous interrupts does not affect the actual architected thread priority value in the PPR. Therefore, if the software does an **mfPPR** at any time during the asynchronous boosting, it always gets the last priority value that is explicitly set by the software for that thread.

5.12 Priority Boosting to Medium-High in User Mode

The POWER9 core allows a problem-state program that executes on a thread to temporarily change the PPR thread priority value to medium high ('5') by executing an **mtPPR** or priority NOP. The temporary thread priority boost is controlled by a 32-bit privileged Problem-State Priority Boost (PSPB) Register. There is one PSBPR per thread, which is set by a move-to PSPB.

A problem state program can set the program priority to medium-high only when the PSPB of the thread contains a nonzero value. The maximum value to which the PSPB can be set must be a power of 2 minus 1. Bits that are not required to represent this maximum value must return '0's when read, regardless of what was written to them.

When the PSPB of the thread is set to a value less than its maximum value but greater than '0', its contents decrease monotonically at the same rate as the SPURR until its contents minus the amount it is to be decreased are '0' or less. The PSPB contents can decrease to less than zero when a problem state program is executing on the thread at a priority of medium high.

When the contents of the PSPB minus the amount it is to be decreased are '0' or less, its contents are replaced by '0'. When the PSPB is set to its maximum value or '0', its contents do not change until it is set to a different value.

Whenever the priority of a thread is medium high and either of the following conditions exist, hardware changes the priority to medium:

- PSPB counts down to '0'
- PSPB = '0' and the privilege state of the thread is changed to problem state (MSR[PR] = '1')

While in problem state at medium-high priority, there can be the potential of the PSPBR reaching '0' at the same time a priority NOP or **mtPPR** is trying to lower the thread priority to a value less than medium. If the attempted write to the PPR occurs in the same cycle, the priority NOP or **mtPPR** must update the PPR with its thread priority instead of allowing the PSPB reset to set the PPR to medium priority.

5.13 Thread Priority Boosting on Asynchronous Interrupt

If a thread has a priority less than medium ('4'), the priority of the thread is boosted on a pending asynchronous interrupt. This allows the interrupt to be serviced faster for a thread (that is waiting for the interrupt at a low-priority state). TSCR[12] is used to enable or disable priority boosting for any pending asynchronous interrupt. The boosting of thread priority does not affect the actual architected thread priority value in the PPR. Therefore, if the software does a move from PPR (**mfPPR**) at any time during asynchronous boosting, it always gets the last priority value explicitly set by the software for that thread.

5.13.1 When to Boost Thread Priority

Thread priority is boosted internally by the hardware on an asynchronous interrupt based on *Table 5-5* on page 148. After the priority is boosted, the hardware continues to treat the thread at medium ('4') priority, until there is an **mtPPR** or priority NOP instruction that changes the thread priority.

Table 5-5. Asynchronous Interrupt

Interrupt	MSR Bits
Hmaintenance	EE = 1 or HV = 0 or PR = 1
Directed External	(EE = 1 and not (HV = 1 and PR = 0 and HEIC = 1)) or (lpes0 = 0 and (HV = 0 or PR = 1))
Mediated External	EE = 1 and (HV = 0 or PR = 1)
Directed Privileged Doorbell	EE = 1
Directed Hypervisor Doorbell	EE = 1 or HV = 0 or PR = 1
Hypervisor Virtualization	(EE = 1 or HV=0 or PR = 1) and HVICE = 1
Perfmon	EE = 1
HDEC	(EE = 1 or HV = 0 or PR = 1) and HDICE = 1
DEC	EE = 1
System Reset	Always (ignores TSCR[12])

Thread priority is boosted to medium for enabled asynchronous interrupts. The reasons for not boosting the priority in the previous cases include:

- Operating system and hypervisor: Spinning on a lock, the priority is low and MSR[EE] = '0'. Priority must not be boosted because nothing useful is going to happen until the lock is acquired. Before the **stdcx** can get the lock, high priority is asserted. Therefore, if a thread is holding a lock, its priority does not need to be boosted when an external interrupt becomes pending.
- Operating system interrupt handler running with MSR[EE] = '0'. Priority is already at the desired level as a result of the implicit or explicit boost. No additional boost is required by the hardware.

5.14 Thread Prioritization Implementation

5.14.1 Thread Switch Fetch Priority

The thread priority is used to apportion fetch cycles. For example, if two threads have a priority weighting that is different, the ratio of those two weights determines the relative number of cycles those two threads will be given access to the I-cache. If all threads have equal priority, the threads are accessed in a round-robin manner.

The instruction fetcher makes no priority provisions for an asymmetric SMT environment. For example, if one side of the core has one thread, the other side has more then one thread, and all threads are of equal priority, then each thread gets an equal number of fetch cycles, even though there are more decode resources available for the thread that is on its own side.

If all of the threads have the same priority, the fetcher fetches the threads in an order that tries to swap from one core side to the other as much as it can. This is desirable because fetching multiple cycles on the same core side increases the chance that no instructions are available to decode/dispatch on the other core side.

Normally, a thread uses its fetch cycle if there is a chance that the fetch can result in a transfer. There are several cases where a thread relinquishes its fetch cycle and allows it to be skipped over. The cases where this happens are as follows:

- The thread is an I-cache miss pending or I-ERAT miss pending.
- The IBuffer is full for eight cycles, such that it is unlikely that there is room in the IBuffer when the instructions are fetched from the I-cache.
- There is a hold fetch from either the ISU or the pervasive core unit that indicates a fetch cannot occur on that thread.

When all the highest priority threads give up their cycle, there are times when the base hardware algorithm cannot assign another thread based on the priority. On the cycles when this occurs, the other threads that can be fetched take turns fetching, ignoring the thread priority.

If there is a flush on a thread and that thread is not already being selected, that thread is selected on the next IFM1 cycle, which is done to reduce the average latency on a flush.

In SMT mode, the target of a predicted taken branch can be fetched three cycles after the branch instruction is fetched. If threads are alternated in SMT2 mode, the earliest time that an instruction could be fetched would be allocated to the other thread, and thus the taken branch penalty goes from three to four cycles.

To reduce this effect, use a pattern in SMT2 mode that in most cases allows the same thread to be allocated every third cycle. The pattern implemented in SMT2 mode is '00100100 11011011'. This pattern causes the same thread to be assigned three cycles later 87.5% of the time.

5.14.2 Thread Switch Decode Priority

The decode and dispatch pipes in SMT2 mode cycle share their resources. In SMT4 mode, they are divided into two parallel groups. Each group has two threads that cycle share that half of their resource. This drives the need for a pair of decode priority engines. In SMT2 mode, one engine controls the two threads. In SMT4 mode, both engines run in parallel: one for the threads assigned to thread set 0 and one for the threads assigned to thread set 1. In SMT modes, each thread set operates in its own half of the dispatch group, independent of the other thread set.

To support two independent thread sets dispatching in parallel, two SMT2 thread priority engines are used. One controls the decode cycles of the threads assigned to thread set 0. The other controls thread set 1. Each engine can manage 1 - 2 threads, depending on how many are assigned to the thread set.

In SMT mode, decode cycles (opportunities to form instruction groups out of the IBuffer) are given to a thread based on the following ordered criteria:

1. Thread enabled, no slots given to a stopped thread, CTRL[12:15].
2. Per thread decode stops for decode hold. See *Section 5.16 Controlling the Flow of Instructions in SMT* on page 152.
3. Instruction availability in the threads IBuffer. Used only if the priority in PPR[11:13] is equal for all enabled threads.
4. Software-set thread priority, controlled in Thread Status Register (PPR[11:13]). See *Section 5.14.3 Software-Set Thread Priority* on page 150.
5. Dynamically changing thread decode priority. See *Section 5.14.5 Dynamic Thread Priority* on page 151.

The first three criteria are only used to eliminate threads from consideration for the next decode cycle. If all available threads are eliminated based on those three criteria, no thread forms an instruction group next cycle. Otherwise, the remaining eligible threads are considered for the next decode cycle according to either their software-set thread priority or a dynamic thread priority algorithm using the current state of each eligible thread.

5.14.3 Software-Set Thread Priority

Software-set priority is used to determine the thread to receive the next decode cycle if at least one of the enabled threads has a different Thread Priority value in PPR[11:13] than the other enabled threads. The intention for the software-set priority algorithm is to divide decode cycles according to the relative values of the thread priority values.

POWER9 core control is given by allowing the user to define the weightings between the seven priorities. A 64-bit SPR, relative priority register (RPR), is provided for the user to set any 6-bit value (0 - 63) for each of the seven priority levels (very low, low, medium low, normal, medium high, high, extra high). Then, PPR[11:13] for each active thread determines which value to read from the RPR.

Each active thread receives a number of decode cycles, relative to the other threads, equal to their priority values. For example, within thread set 0, T0 has a relative priority of 17 (as defined by PPR[11:13] and the RPR), T3 has a relative priority of 6. Within a window of 17 + 6 = 23 decode cycles, T0 gets 17 cycles and T3 gets 6 cycles. The pattern repeats every 23 decode cycles. Additionally, the cycles given to each thread are distributed as evenly as is reasonably possible within the pattern.

5.14.4 Low-Power Modes for Application

The POWER9 core slows the rate of decode to reduce power whenever any enabled threads have a priority of '001', as set in each thread's PPR[11:13].

Any time an enabled thread is set to the lowest priority, it is limited to one decode cycle every 128 cycles.

5.14.5 Dynamic Thread Priority

If all enabled threads have the same thread priority value, a dynamic thread priority algorithm based on the state of the eligible threads determines which will get the next decode cycle. This algorithm uses a scoring system built on the resources occupied by each thread, whether the thread has any outstanding L2 or TLB misses, or if there is an active multicycle operation active for a thread. A final round-robin adder is used only to break any ties between threads. This algorithm is implemented twice, once per thread set, where each algorithm manages 1 - 2 threads. *Section 5.14.2 Thread Switch Decode Priority* on page 150.

The eligible thread with the highest overall score is given the next decode cycle. Note that the round-robin pointer only affects results in the event of a tie from the other three adders. To ensure fairness between threads when one or more threads are disabled, the round-robin pointer rotates between all threads that are available in the current SMT mode regardless of whether the thread is enabled.

5.15 Support for Multiple LPARs

The POWER9 core runs in 4LPAR mode. Each thread has its own partition resources. Fetch and decode cycles are handed out as described in the following sections.

5.15.1 Microcode Fairness

In multi-LPAR mode, the goal is to give each LPAR the same number of dispatch cycles. However, multi-cycle microcode instructions can cause an LPAR to consume multiple consecutive cycles. To compensate, the POWER9 core gives the other LPAR sharing the dispatch bandwidth extra decode cycles to compensate for the loss of decode cycles. When microcode operations are in flight, each operation generates 32 groups (such as, load multiples).

The counter handles $32 \times 12 = 384$ catch-up cycles for either LPAR. A 10-bit counter handles up to 512 cycles for each LPAR.

5.15.2 I-ERATs

In SMT2 and SMT4 mode, the instruction-side ERAT is split in two with even threads getting 32 entries and odd threads getting the other 32 entries.

5.16 Controlling the Flow of Instructions in SMT

The ability to control the relative flow of instructions in an SMT processor is important for optimal performance. When one thread is not making good progress due to reasons such as an L2 miss, TLB miss, **sync**, or other long-latency operations, the other thread can be given additional machine resources. The following features are built into the POWER9 core for controlling SMT instruction flow.

5.16.1 Dispatch Flush

A dispatch flush is a low-latency flush that flushes the decode pipe. **Ptesync**, **tlbie**, and instructions with the scoreboard bit set can cause a dispatch flush on the POWER9 core. Also, if enabled, a thread that was balanced flushed is dispatch flushed if the chosen thread is stalled at dispatch.

5.16.1.1 Dispatch Flush Rules

1. Dispatch flushes are disabled if the core is in single-thread mode, or if the core is in SMT2 or SMT4 mode and there is one or fewer than one threads active.
2. Dispatch flush occurs only if a thread shares a group with another thread.
3. If both threads are on group0, (not balanced), dispatch flush can still occur.
4. Never dispatch flush a thread if it is in the middle of a microcode. Dispatch flushes related to SMT performance are never done in the middle of a microcode dispatch. Other dispatch flushes can happen to microcode, such as quiesce, RAS, or a forward-progress time-out.
5. A **ptesync** instruction from thread A causes a dispatch flush of thread A (similarly, for other threads).
6. If the ICT thread is not empty, the **tlbie** instruction is dispatch flushed. The instruction following the **tlbie** is dispatch flushed if the **tlbie** instruction has not received tlbie acknowledge from the Nest through the LSU.
7. If thread A has its scoreboard bit set (such as, a non-renamed **mtspr** followed by **mfspr**), thread A is dispatch flushed (similarly, for other threads).
8. If TSCR[3] is enabled, dispatch flush the thread that was chosen for a balance flush if that thread is stalled at dispatch.
9. Dispatch flush any instructions that are marked dispatch serialize by the cores debug and decode mechanisms.

5.16.1.2 Stall at Dispatch

A thread can be *stalled at dispatch* due to unavailability of a shared resource that it needs for the next dispatch. When stalled, dispatch_hold is asserted to hold the decode pipe. The complete list of stall conditions follows:

- Required reservation queue entries are not available
- Not enough history buffers are available
- ICT is full

Note:

1. A 5-cycle delay is expected between counting the ICT entries and taking any action based on that count.
2. A 4-cycle delay is expected between the detection of a stall condition and causing a dispatch flush.

5.16.2 Decode Hold

1. After a dispatch flush, a **ptesync**, instruction is held at IBUF until its dispatch conditions are met. This is done irrespective of thread priority.
2. After a dispatch flush, a **tlbie** instruction is held at IBUF until the ICT thread is empty. After the **tlbie** instruction is dispatched, the next instruction is dispatch flushed and then held at IBUF until ICT and the SRQ are empty. This is done irrespective of thread priority.
3. After a balance flush due to an L3 or TLB miss, instructions on the balance flush thread are held at IBUF until the miss is resolved or until the balance flush miss counter reaches the threshold value as determined by TSCR[8:9].

5.16.2.1 Balance Flush

A balance flush is an NTC+1 flush that flushes all instructions that are younger than the next-to-complete instruction group on a selected thread. It flushes the execution units, ICT, and EAT for the selected thread. Threads are considered for a balance flush only if any thread is stalled at dispatch or the dispatch buffer is empty and the thread being considered has at least one instruction in the ICT. Balance flushes can be disabled using TSCR[2].

Criteria for Selecting a Thread to be Balanced Flushed

If the core is in SMT mode and more than one thread is active, perform the following steps on a dispatch stall:

1. Select the threads with any number L3 or TLB misses, regardless of the balance flush miss counter value. If enabled by a debug switch, select only the threads with any number of L3 or TLB misses, if the balance flush miss counter for the thread is less than the counter threshold as described by TSCR[8:9]. If the miss counter for the thread is greater than the threshold value, ignore the miss on that thread and do not consider the thread for a balance flush.
2. If only one thread has an L3 or TLB miss, select that thread to be balanced flushed. Raise the CLB hold on the thread that was chosen to be balanced flushed until either the miss has been resolved or the balance flush miss counter threshold has been reached as described by TSCR[8:9].
3. If in SMT4 mode and more than one thread is eligible to be balanced flushed based on having an L3 or TLB miss, select all eligible threads to be balanced flushed. Raise the CLB hold on the threads that were chosen to be balanced flushed until either the miss has been resolved or the balance flush miss counter threshold for that thread has been reached as described by TSCR[8:9].

4. If in SMT2 mode and both threads are eligible to be balanced flushed based on having an L3 or TLB miss, do not balance flush either thread. Do not raise the CLB hold.
5. If the thread that is stalled at dispatch is also the thread that was chosen to be balanced flushed, then also do a dispatch flush on that thread if TSCR[3] is enabled. Otherwise, do only a balance flush on the chosen thread.

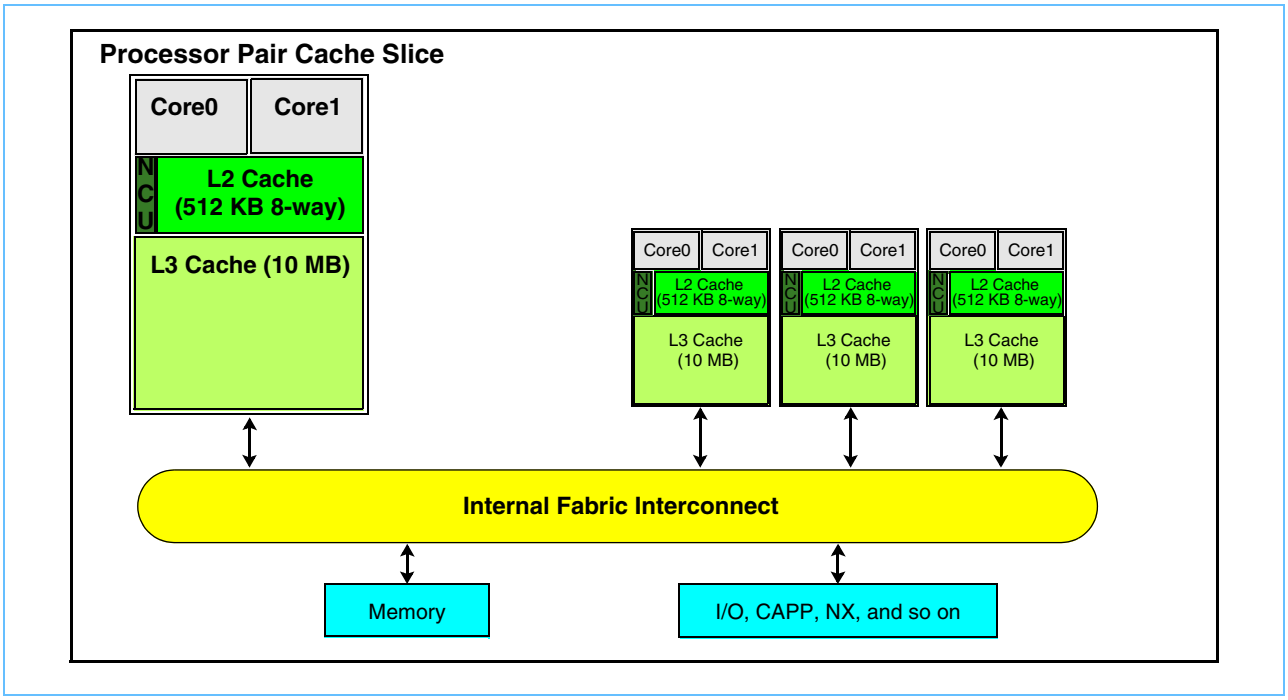
6. L2 Cache

6.1 Overview

The L2 cache unit is contained within the processor pair cache slice, which consists of: two POWER9 cores, one 512 KB L2 cache, one 10 MB L3 cache, one NCU, and a portion of the internal Fabric interconnect logic that is referred to as PBex. The L2 cache is a unified cache that is accessed privately by a given pair of POWER9 cores. The L2 cache maintains full-hardware coherence within the system and can supply cache on-chip intervention data to other cores on this POWER9 chip or off-chip intervention data to other POWER9 chips. The L2 unit is a store-in cache that is fully inclusive for a pair of POWER9 cores, each of which have an L1 D-cache and I-cache (note that the POWER9 core is based on a store-through L1 D-cache design). The L2 unit also supports private bus access to a 10 MB L3 cache that is also private to this pair of POWER9 cores for fast L3 hit data access and for storage of L2 victimization data.

Figure 6-1 shows a high-level POWER9 chip diagram with multiple POWER9 processor-pair cache slices interconnected via the internal Fabric.

Figure 6-1. POWER9 Block Diagram of a Multiple Processor-Pair Cache Slice Interconnected via the Internal Fabric



L2 cache feature summary:

- 512 KB private cache per core-pair:
  - 128-byte line, 8-way set associative
  - Both I-side and D-side inclusive for the pair of POWER9 4-threaded cores
  - Double-banked cache design interleaved on even/odd cache-line boundary
  - L2 cache can perform a read from one cache bank while writing to the other cache bank
- 8-way directory, dual-banked multi-ported:
  - One processor read port, two snoop read ports, and one write port per physical bank
  - The processor port operates at ½ the processor clock rate into a given bank (initiated on 2:1 clock boundary)
  - The snoop port into a given bank operates at ½ the processor clock rate (initiated on 2:1 clock boundary) allowing for up to four snoops per 2:1 clock across the four banks.
  - The dual banked directory can initiate:
    - Up to five directory reads in a given 2:1 cycle (four on the snoop ports and one on the processor port)
    - One write in a given p-clocks cycle (where directory writes are scheduled on the second half of a 2:1 cycle, such that they never conflict with directory reads)
- 512 × 13-bit LRU arrays (logical configuration)
  - 2 × 4 LRU vector tracking tree with cache invalidate state biasing
  - Supports LRU, direct map, single-member, and pseudo-random modes
- Point of global coherency
- Reservation stations: one per processor thread (eight total across the pair of four threaded POWER9 cores)
- Support for transactional memory operations
- Four snoop-bus ports selected by the cache-line "real-address" bits [55:56]
- Support strong-access ordering (SAO) support
- Hardware directory line delete capabilities to support faulty L2 cache elements

L2 Feature RAS Summary

- Directory-array data protected by SECDED ECC
- Directory-array single-bit stuck-bit detection and correction
- Directory line-delete support
- Cache-array data protected by 8-byte SECDED ECC
- 8-byte ECC throughout the internal L2 data-flow and migration flow to the internal Fabric interconnect or L3 cache
- FIR/SCOM support
- L2-cache purge support
- Various L2 hardware end-to-end type control checkers (for example, end-to-end type protocol checking that checks for unexpected internal Fabric interconnect cresp or data)

6.2 L2 Unit Internal Resources

The L2 unit contains a set of internal machine and enqueuing resources scheduled to handle the processing of load and store type requests from the POWER9 cores and requests from the internal Fabric interconnect. *Table 6-1* lists the type of resource and the size and function of these resources within the L2 unit.

Table 6-1. L2 Resources (Share between a Pair of POWER9 Cores)

Resource	Description	Size	Hash Bits
CIU_Load Request Queues	Handle the staging for load requests from the POWER9 cores that are pending access to the Read Claim (RC) machines dispatch pipe dispatch pipe to be assigned to an RC machine or sent to an L3 prefetch machine.	16 demand 8 data prefetch 8 instruction fetch/prefetch 4 translate	none
L2 CIU_Store Queue	Intermediate buffer for stores from a given POWER9 core. The CIU_Store Queue contains six 16-byte entries per POWER9 core. This buffer unloads its stores into the L2 store gather station as room becomes available.	2 banks of 6 × 16 bytes	core ID
L2 Store Queue (Gather Station)	Store requests from the POWER9 cores and gathers stores that are from the same core thread and are to the same 128-byte cache line. The L2 store-queues gathering mechanism gathers stores into a single 64-byte block with up to two sectors of clustering to be processed by the L2's RC machine.	2 banks of 28 × 64 bytes	addr(56)
Read Claim (RC) Machines	Read claim machines manage all cacheable operations initiated by the local core pair. The RC handles: • Gaining ownership of the line (either via an L2 hit, L3 hit, or internal Fabric interconnect access). • Updating the core with data for its request. • Updating the L2 cache with the data. • Updating the L2 directory with the current coherent state and inclusivity information for this line. • Issuing any required I-side or D-side kills to the L1 caches in the core.	2 banks of 8	addr(56)
Cast Out (CO) Machines	Castout machines manage moving victimized lines from the L2 cache to the L3 cache and sending kills to the POWER9 core when the L2 cache is no longer tracking this line. Under certain conditions, the L2 castout machines can select to move the line to the memory controller.	2 banks of 8	addr(56)
Snoop (SNP) Machines	Snoop machines manage all cacheable operations initiated by the incoming internal Fabric interconnect operations. The snoop is responsible for: • Representing the current L2 directory state to the internal Fabric interconnect. • Intervening data to the internal Fabric interconnect when necessary. • Updating the L2 directory with the current coherent state for this line. • Issuing any required I-side or D-side kills to the L1 caches in the core based on internal Fabric interconnect activity. • Pushing modified data to memory when required to by the snooped operation.	4 banks of 4	addr(55:56)
Reservation	The reservation logic provides for execution of **larx**/**stcx** instructions. A reservation is provided each thread within the core pair	2 (cores) × 4 (threads)	thread
Transactional Memory	Transactional memory tracking CAM for tracking the TM load and store footprint structures across all threads along with an LVDIR structure for tracking a larger load footprint for two threads.	TMDIR: 4 banks of 16 entries (shared by 8 threads)) LVDIR: 512 × 8-way entries (shared by 2 threads)	addr(55:56)

6.2.1 Description of L2 Control Flow

The L2 control flow handles the coordination of dispatching load and store requests from the pair of POWER9 cores and dispatching snoop operations from the internal Fabric interconnect that were initiated by other cores or I/O type devices. This L2 control logic manages these accesses such that when contention or ordering is required among commands, the proper dispatch collision and ordering detection occurs such that the RC/CO machines and SNP machines perform their data references and storage updates in a coherent and consistent manner.

The L2 CIU_store queue consists of a "2 core banks × 6-entry × 16 bytes" buffer that has a bank dedicated per core. Each bank serves as an intermediate buffering station for stores that are then forwarded onto the L2 store queue where they can be gathered with prior stores. The L2 CIU_store queue has six entries per core. This buffer structure can unload up to two entries (one per core bank) per 1:1 clock cycle provided these requests are going to different cache-line banks in the L2 store queue. The L2 CIU_store queue maximizes its unload efficiency by selecting first entries from the two different core banks that are scheduled to go to different L2 store queue cache-line banks.

The L2 store queue is a "2 bank × 28-entry × 64-byte" buffer that allows storage updates to the same cache line from a given thread to be gathered before being issued into the RC machine dispatch pipe. The L2 store queue issues stores into the RC dispatch pipe honoring the barriers that have been inserted by software. The L2 store queue takes full advantage of the weakly-ordered nature of the PowerPC Architecture by allowing as many RC machines to be running in parallel where ordering is not required.

The CIU load queues manage the enqueuing of load-type requests from the two POWER9 cores for I-side, D-side, translate, and prefetch type requests that cannot be immediately serviced by the RC machines. When a backlog of load-type requests occur, due to resource contention, the CIU manages the detection of when the resource is free and the priority for which requests should be re-issued next into the RC dispatch pipe to achieve best performance and for fairness.

The set of sixteen RC machines are dispatched on behalf of load/stores from its private core pair. This set is responsible for acquiring the proper coherent authority to complete the command and is a unified cache that is accessed privately by a given pair of POWER9 cores. Along with these RC machines, the CO machines also are conditionally dispatched to move any victimized lines out of the L2 cache to make room for the new line the RC is bringing in.

When a given command is sent down the RC dispatch pipe, one RC machine is assigned the command. The RC machine then determines where it must go to acquire the proper coherency authority to perform the command and, if required, acquire a copy of the data. The RC machine can find the line that is already in the L2 cache (for example, an L2 hit), or in the L3 cache (for example, an L3 hit), or the RC machine might have to proceed to the internal Fabric interconnect to gain ownership of the line. In addition, when an RC machine is dispatched, a CO machine might also be assigned at RC dispatch time if the L2 cache must create a victim line to make room for the line the RC is now installing. The L2 CO machine has the resources to work independent of the RC machine and thus allow the CO machine to work in parallel to migrate the line down to its private L3 cache. The L2 CO machines are also responsible for sending any required invalidates to the POWER9 core pairs when lines are invalidated from the L2 cache.

The L2 unit has four SNP dispatch pipes that control the assignment of the SNP machines to work on commands from the internal Fabric interconnect. These commands, initiated by internal Fabric interconnect masters, might include other L2/L3 unit masters (on behalf of their cores commands) or by other masters such as I/O type devices. The L2 SNP machines are used to service these requests (that is, L2 hits) by granting coherent authority from the L2 directory during SNP dispatch and (if required) providing a copy of the

data via data intervention back to the requesting internal Fabric interconnect master. The L2 SNP machines are also responsible for sending any required invalidates to the POWER9 core pairs when lines are invalidated from the L2 cache.

Both the RC dispatch pipe and the SNP dispatch pipe have the ability to detect when a given line has already been assigned to either an RC, CO, or SNP machine. Contention by the subsequent command for the same line is detected and thus delayed until after the previous machine has completed its work on the command it is working on.

6.3 Interfaces

On the POWER9 chip, the L2 caches have interfaces to communicate with their respective POWER9 core pair, its private L3 cache, and the internal Fabric interconnect (that provides access to the neighboring on-chip and off-chip L2/L3 caches). These interfaces are described as follows:

- Core load request interface: Interface for loads, IFetch, and various prefetch operations that allow for requests from the respective units to be sent every core clock cycle.
- Core reload bus: The reload bus is used to return data and L1 cache invalidate commands to the core. The reload bus contains a data bus that is 64 bytes wide and runs at a 1:1 core-clock rate. Both data and invalidates are returned to the core over this bus in a non-blocking fashion into the core (that is, no flow control required).
- Core store interface: Each core has its own dedicated store request interface that provides one 16-byte wide store data bus (per core) clocked at a 1:1 clock rate. Each of these interfaces is also used to send barrier operations and various PowerPC architecture-specific commands from a POWER9 core. Flow control is achieved through a core push/pop protocol, where the core holds an "L2 queue capacity counter" to track the L2 store queue capacity.
- L2-to-L3 read interface: One private L2-to-L3 request interface that is used by the L2 cache to look up the L3 cache on behalf of a request from this private core pair. This private interface serves as a dedicated fastpath 64-byte data interface for L3 hit data transfer back to the L2 cache and core pair at a 2:1 clock rate.
- L2-to-L3 castout interface: One private L2-to-L3 request interface that is used by the L2 cache for casting data out of the L2 cache and into the L3 cache. This private interface consists of a dedicated 64-byte data interface from the L2 to the L3 cache at a 2:1 clock rate.
- L2-to-internal Fabric interconnect: Request ports are provided for both address and data requests to be sent to other nest units in the system (both on and off chip). The data interface consists of: one outbound 32-byte data port that runs at a 2:1 ratio to the core and one inbound 32-byte data port that also runs at a 2:1 clock ratio to the core. The internal Fabric interconnect also provides four instances of the snoop interface for processing system coherence and control commands. An interface instance consists of three ports: an address/cmd snoop port, a unit response port, and a system combined response port.

6.4 Operational Flows and Bandwidths

Figure 6-2 shows the general control and data flow within the POWER9 processor-pair cache slice. The figure shows the internal buffer structures within the L2 and L3 cache and how these buffer structures are connected within the private L2 and L3 units for processing L2 and L3 hits and any resulting castouts from the L2 to the L3 or interaction with the internal Fabric interconnect.

Figure 6-2. High-Level Dataflow within the L2, L3, and NCU for a Processor Pair Cache Slice

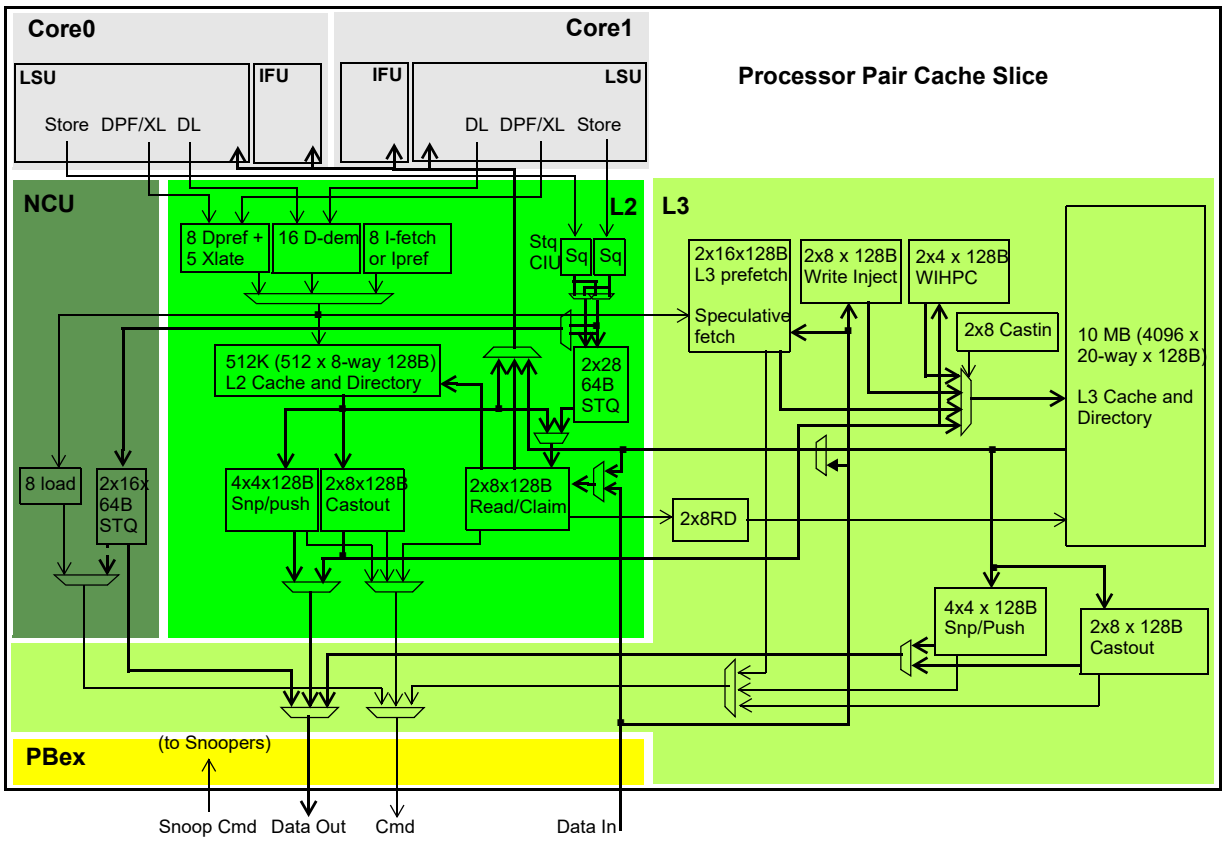


Figure 6-3 on page 161 and *Figure 6-4* on page 162 show additional details about the L2 unit. *Figure 6-3* shows the details of the internal L2 dataflow. *Figure 6-4* shows the L2 units major input/output buses and the data bus bandwidth that represents the overall bandwidth capabilities in/out of the L2 cache.

Figure 6-3 shows the data flow within the L2 cache unit.

Figure 6-3. L2 Data Flow Overview

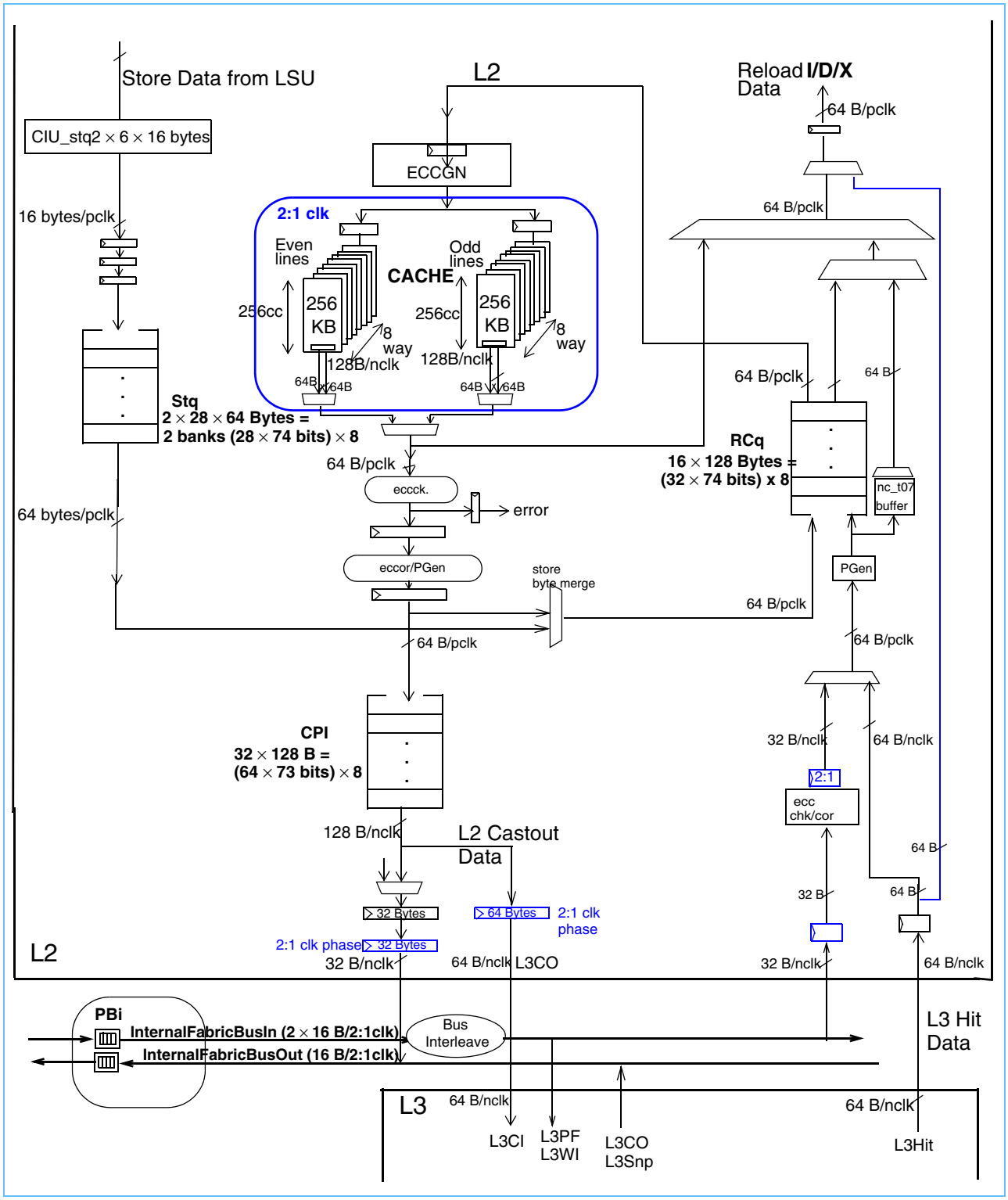
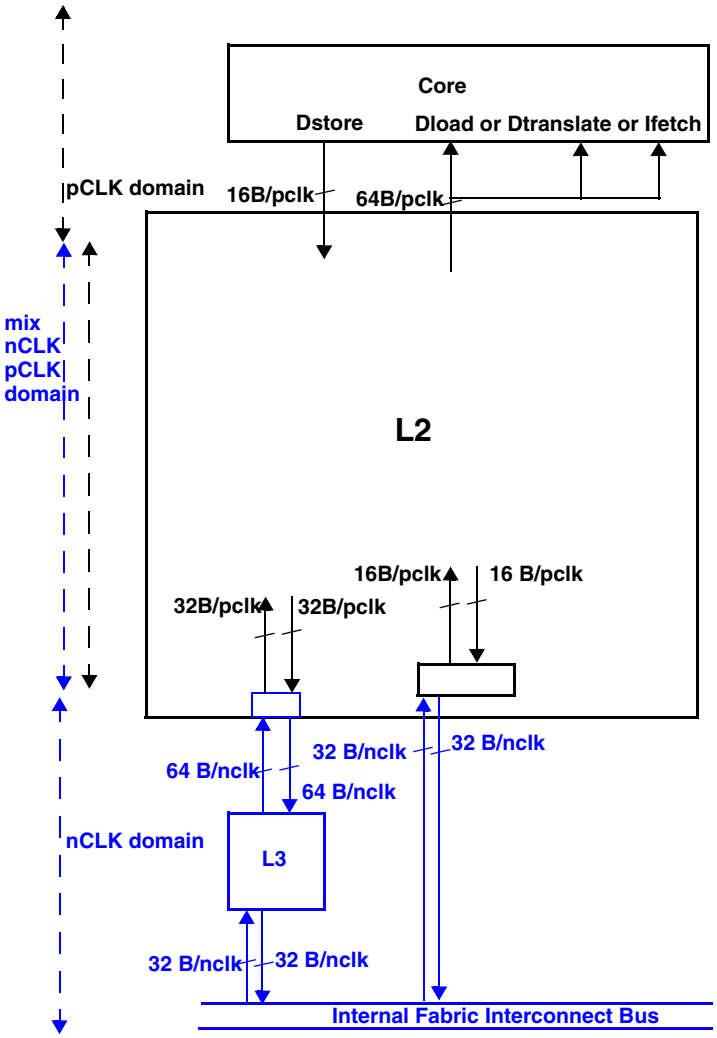


Figure 6-4 shows the various control machines and address flow internal to the L2 cache unit.

Figure 6-4. L2 Bus Bandwidths



6.5 LRU

6.5.1 LRU modes

- Normal LRU mode (13 bit 2 × 4-way pointer mechanism)
- Direct Map: uses addr [45:47] (just above L2 cgc)
- Single-Member Mode: configuration register specifies one of eight members
- Pseudo-Random Static Mode: LFSR based.

6.5.2 Policies

- invalid_line_lru_bias: Moves members in I state toward LRU.
- id_state_mru_bias: Moves members in invalid line deleted (Id) state toward MRU.

6.5.3 Line Disable

- Directory support for line delete (Id) on a per directory location granularity.
- Hardware or software-initiated line delete policies.

6.6 Transactional Memory Support

6.6.1 Basic Policy

Transactional memory (TM) support is provided in a transaction tracking structure that exists in the L1 and L2 cache. With this tracking structure, the L2 cache is considered the final point of commitment for the TM image, but the image is also allowed to initially bind in the L1 cache. The L1 and L2 caches each have a TM tracking structure that tracks the overall footprint a given thread has accessed and if that set of lines that makes up the footprint has been accessed as part of a TM load or TM store access. This TM mechanism provide a means for tracking the TM load and store footprint, which is limited to the size of the L2 TM tracking structure. In addition to the tracking structure, the L2 cache uses the L3 cache to hold any dirty lines that are associated with the L2 cache's pending speculative TM store footprint, such that if the L2 TM sequence fails, the L3 image has preserved the original dirty cache lines.

6.6.2 L1 TM Filter Structure and L2 TM Tracking Structure

- L1 Set-P directory bits: 32 × 8-way × 4 bits. This Set-P structure (which is used for normal L1hit use) also includes a TM bit, a Private bit, and a 2-bit thread ID.
- L2 TMCAM: 4 × 16-entry pointer structure, which is shared by the two POWER9 cores that manage the tracking of the TM load and store footprint tracking and is shared by all eight threads across the two POWER9 cores.
- L2 LVDIR: 512 KB tracking structure, which is shared by the two POWER9 cores that manage the tracking of a larger load footprint for up to two threads at any given time.

The L1 and L2 CAM structures hold an address and a set of status bits that indicate if this cache line is currently associated with a TM load or TM store footprint for a given thread.

L1 Load/Store Footprint Filter Tracking

Structure
- Marker bits in existing Set-P directory (for example, Set-P directory is the size of the L1 directory)

Entry tracking
- A load/store footprint entry only tracks an entry for one thread at any given time.
- A given cache-line address is tracked as part of the load/store footprint after a transactional memory load or store is performed to that line.
- The L1 TM tracking does not track TM for pass/fail correctness; TM correctness tracking is done by the L2 tracking.

L2 Load Footprint TMDIR Tracking

Structure
- CAM of 16 entries × 4 banks (address[55:56] defines the four banks) that tracks both the load and store footprints.

Entry tracking
- Each entry is a "shared pool" type for a load footprint where a given entry can track when it is associated with one or more threads each sharing this line as part of their TM load footprint.

L2 Store Footprint TMDIR Tracking

Structure
- CAM of 16 entries × 4 banks (address[55:56] defines the four banks) that tracks both the load and store footprints.

Entry tracking
- A given cache-line address is tracked as part of the store footprint after a transaction memory store is performed to that line.

L2 Load Footprint LVDIR Tracking

Structure
- This array structure is 512 entries × 8 ways × 2 threads, which is the same size as the L2 director and is used for tracking large TM load footprints.

Entry tracking
- Threads are assigned to this structure when the L2 detects TM activity from any given thread.
- The LVDIR array can track the TM load footprint image that is the size of the L2 cache for up to two threads at any given time.

L3 Cleaned Footprint Tracking

Structure
- CAM of 16 entries × 4 banks (address[56:57] defines the four banks) with a thread identifier field.

Entry tracking
- This structure tracks lines cleaned by the L2 cache when the L2 cache formed the store footprint on a line that was in either this core's L2 or L3 cache at the time of the TM store. These lines represent the "true image of memory", which is re-exposed if the store footprint for these lines is discarded due to a TM transaction failing for a given thread.

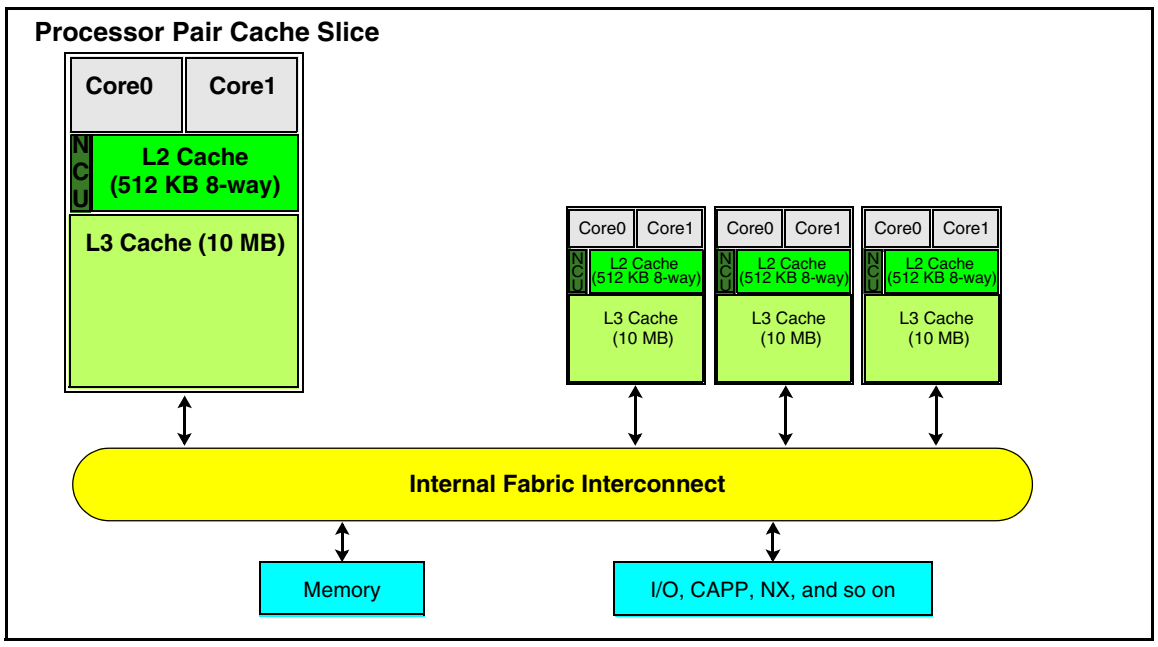
7. L3 Cache

7.1 Overview

The L3 cache unit is contained within the processor-pair cache slice, which consists of: two POWER9 cores, one 512 KB L2 cache, one 10 MB L3 cache, one NCU, and a portion of the internal Fabric interconnect logic, which is referred to as PBex. The L3 cache is a unified cache that is accessed privately by the L2 cache in the same processor-pair cache slice and nonprivately by other unit masters in the system via the internal Fabric interconnect. The L3 cache maintains full hardware coherency within the system and can supply intervention data to other unit masters in the system. The L3 cache is a victim cache and thus typically holds cache lines that are different from those held by the L2 cache or the core L1 cache.

Figure 7-1 shows a high-level POWER9 chip diagram, that includes multiple processor-pair cache slices interconnected via the internal Fabric.

Figure 7-1. Block Diagram of Multiple Processor-Pair Cache Slice Interconnected via the Internal Fabric



7.2 Interfaces

The interface to the L2 cache is used to service read (which include both load and store requests) and castout requests from the L2 cache. The interface to the internal Fabric interconnect is used to snoop requests from the internal Fabric, to prefetch cache lines into the L3 cache based on requests from the core, and to castout evicted cache lines. Snooping can lead to providing intervention data to the internal Fabric interconnect or generating cache-line pushes to the internal Fabric interconnect. The L3 cache also accepts lateral-castouts from other on-chip L3 caches.

The L3 cache supports a fast broadcast request interface to other on-chip caches and the on-chip memory controllers for high priority (demand) read requests, that are L2 and L3 misses. Upon the occurrence of such a miss, the L3 cache broadcasts to these devices, ahead of the request to the slower coherent internal Fabric interconnect. If any of the caches are in a state to deliver early data directly, it does so on the internal Fabric interconnect data buses. That cache then waits for the coherent request on the internal Fabric interconnect and responds in a manner that is consistent with its response to the fast request. In parallel to the cache activity, the memory controller begins to fetch the data. It then examines the combined response to determine if it should provide data.

7.3 List of Features and Resources

- Features:
  - Private 10 MB L3 cache/shared L3.1 (lateral-castout target cache from other on-chip L3 caches).
  - 20-way set associative.
  - 128-byte cache lines with 64-byte sector support.
  - 10 EDRAM banks (interleaved for access overlapping).
  - 64-byte wide data bus to L2 for reads.
  - 64-byte wide data bus from L2 for L2 castouts.
  - Eighty, 1 Mb EDRAM macros configured in ten banks, with each bank having a 64-byte wide data bus.
  - All cache accesses have the same latency.
  - 20-way directory organized as four banks, with up to 4 reads or 2 reads and 2 writes every 2 pclks to differing banks. The directory is physically implemented as twenty $1024 \times 4w \times 51$ bits SRAMs.
  - LRU algorithm for castin selection using 4-bit per member recency/frequency utilization tracking.
- Functionality:
  - Victim cache for local L2 cache (L3.0).
  - Victim cache for other on-chip L3 caches (L3.1).
  - Services read requests from the local L2 cache due to core loads or stores that miss in the L2 cache.
  - Services prefetch requests that originate from a local core and are pass through by the local L2 cache.
  - Four snoop ports, address banked by address bits 55 and 56.
  - Internal Fabric interconnect support.
  - Cache inject support, including partial line inject on any byte boundary.

  - Dual-class L3.0/L3.1 LRU support and L3.1 activity throttling
  - Support for speculative memory transactions by acting as backing store of previous version of cache lines until outcome of transaction is known. 64-entry CAM to track Sc lines.
  - Fast broadcast on-chip load request to other caches and memory controllers for L2 demand load that miss in the L3 cache.
  - 3-hop internal Fabric interconnect topology with multi-level command scope support.
  - Entire L3 clocked at one half core frequency.
- Peak bandwidths:
  - 64 bytes, 2:1 clock L2 reload bus capacity
  - 64 bytes, 2:1 clock L2 castout capacity
  - 32 bytes, 2:1 clock prefetch capacity
  - 32 bytes, 2:1 clock L3 castout capacity
  - 32 bytes, 2:1 clock full-line cache-inject capacity
  - 32 bytes, 2:1 clock intervention capacity
- RAS features:
  - Data cache contents protected by 8 bits per 8 bytes SECDED ECC
  - Directory contents protected by SECDED ECC
  - ECC propagation throughout interface dataflow and buffers
  - Support for directory single stuck bit
  - Physical data line delete support

7.4 Queues

The L3 cache has multiple sets of state machines that act as queues for various types of requests.

7.4.1 Read Machines

The L3 has 16 instances of read (RD) state machines that handle L2 read requests. L2 read requests are a consequence of core loads and stores that miss in the L2 cache. The L3 first attempts to service an L2 read request from the L3 cache. If the request misses the L3 cache, the request is either sent to the internal Fabric interconnect (though handled by the L2 controller thereafter) or returned (with a miss indicator) to the L2 cache, which subsequently sends a request to the internal Fabric interconnect. Details about the RD state machines follow:

- 16 RD machines.
- Address protection provided.
- Carries out an L3 directory invalidate on an L3 hit.
- CI/CO machines are not coupled to an RD machine.
- No speculative cache reads. Directory is read first.
- Request to the fast bus when an L2 read request misses the L3 cache and the L2 read request enables fast handling. Request to the fast bus is in parallel with an L3 miss indication to the L2 cache.

- L3 cache data is initially sent to the L2 cache on a cache hit without ECC correction. In the case of a correctable data error, the RD machine re-reads the cache line from the L3 cache and re-sends the data to the L2 cache after running it through an inline ECC correction.

7.4.2 Castin/Castout Machines

The L3 cache has 16 CI state machines that handle L2 castouts, prefetched data that arrives from the internal Fabric interconnect, lateral castouts (LCOs) that arrive from the internal Fabric interconnect, and full or partial cache-line injections that arrive from the internal Fabric interconnect. The castin state machines read the L3 directory, then write the directory and the data cache.

The L3 cache has 16 CO state machines that handle writing displaced dirty cache lines (displaced by a castin) to the internal Fabric interconnect. Details about the CI and CO state machines follow:

- CI machines:
  - 16 CI machines.
  - Address protection provided.
  - Handles 128/64-byte CI that results from an L2 CO, prefetches, incoming LCOs, and incoming partial or full cache-line injects.
  - Private control/data interface for moving data from the L2 CO buffers to the L3 cache (no extra storage in the L3 cache).
  - TM footprint (Sc) established in the L3 cache at the CI allocation time to provide protection and collision detection. The cache line is held in the Sc state until the completion point of the transaction is detected. The Sc cache line can be evicted to memory if necessary.
  - Flow control mechanisms to provide fairness and to prevent L2 COs, PF, and WI from overflowing CI machines.
  - Handles purge/flush function initiated via the SCOM register interface.
- CO machines:
  - 16 CO machines. Each CO machine is paired with a CI machine.
  - Address protection provided.
  - Handles L3 COs due to L3 CIs.
  - Handles L3 cast-through operations of cache lines not to be installed in the L3 cache.
  - Detects overflow of the TM footprint.
  - 1×128-byte buffer per CO machine (physically in the CPI buffer).
  - Cache interlock: CI cache write held off until a CO cache read is done.
  - Directory interlock: CO active for protection until a CI completes a directory write to destroy an old entry.

7.4.3 Prefetch Machines

The L3 cache has 32 prefetch (PF) state machines that make read requests to the internal Fabric interconnect. Each PF machine starts with a request that originates from a local core and is passed through the L2 cache. When a PF state machine receives a request to make a prefetch, it first reads the L3 directory. If the L3 cache already has the cache line, the PF machine does nothing further and goes idle. If the cache line is not in the L3 cache, the PF machine makes a read request to the internal Fabric interconnect. The internal Fabric interconnect request might indicate that the memory controller is allowed to discard the request if the memory controller is currently busy with higher priority requests. If prefetch data arrives from the internal Fabric interconnect, the PF machine determines if a demand request has arrived for that cache line while the prefetch request to the internal Fabric interconnect was pending. If no such request is pending, the PF machine forwards the request to a CI machine to have it put in the L3 cache. However, if a demand request has arrived, the data is forwarded directly to the L2 cache and is not installed in the L3 cache. Details about the PF state machines follow:

- 32 L3 PF machines.
- Address protection provided.
- Handles prefetch requests from a local core.
- For entries that are L3.0 misses, the PF machine sends a request to the internal Fabric interconnect.
- PF data bypasses to the L2 cache without installing the cache line in the L3 cache when the RD machine is waiting for data.
- When no RD machine is waiting for the data, the L3 PF machine uses the CI/CO machine to move data from the PFWI buffer to the L3 data cache.

7.4.4 Snoop Machines

The L3 cache has four snoop dispatch pipes that handle incoming reflected-command (snoop) requests from the internal Fabric interconnect. The snoop dispatch pipes perform an L3 directory read to determine how to handle the request. If the request requires either sending intervention data, execution of a snoop push, or invalidation of the associated cache line, then the snoop dispatch pipes forward the request to one of 16 snoop state machines (SN). If the request is an incoming lateral castout (LCO) or cache-injection that is accepted, the request is forwarded to the write inject (WI) state machine. A request to an SN machine to invalidate the cache line can occur in parallel to a request to a WI machine. Details about the SN state machines follow:

- 16 SN machines.
- Address protection provided.
- Handles intervention for snoops.
- Handles LCOs without data movement (state merge).
- Handles detection of bus operation collisions with local speculative TM footprint.

7.4.5 Write Machines

There are 16 write inject (WI) state machines that are started by the snoop dispatch pipes on incoming LCO requests or incoming write injection requests. For these types of requests, the WI machine waits for data to arrive from the internal Fabric interconnect, then passes the request to a CI machine to have the cache line inserted into the L3 cache. Details about the WI state machines follow:

- 16 WI machines, eight that can be used for partial-line cache injects.
- Address protection provided.
- Handles LCOs with data movement from on-chip L3 caches and cache injects.
- L3 WI machine uses CI/CO machine to move data from PFWI buffer to the L3 data cache.

7.4.6 Transaction Memory Machines

There are four transactional memory (TM) state machines for handling transactional memory requests from the L2 cache. Details about the TM state machines follow:

- Four transactional memory machines.
- 64 CAM entries (16 entries per each of the four directory banks). Flash validate and pseudo-flash invalidate via the Store Invalid (SInv) bit.
- Tend_pass handling for clearing the L3 cache of speculative Sc cache lines.
- Tend_fail handling for flash clearing of the Sc lines to restore them to the normal state.

8. SMP Interconnect

The POWER9 SMP interconnect is the underlying hardware used to create a scaleable cache-coherent multi-processor system. The POWER9 SMP interconnect controller provides coherent and non-coherent memory access, I/O operations, interrupt communication, and system controller communication. The SMP interconnect provides all of the interfaces, buffering, and sequencing of command and data operations within the storage subsystem. The SMP interconnect is integrated on the POWER9 chip with 24 processor cores and an on-chip memory subsystem. The POWER9 chip has up to two SMP external links that can be used to connect to other POWER9 chips.

The external SMP interconnect link is a split-transaction, multiplexed command and data bus that can support up to two POWER9 chips in a system. Aggregation of data links between the same source and destination chips is supported to increase data bandwidth.

Cache coherence is maintained by using a snooping protocol. Address broadcasts are sent to the snoopers, snoop responses are sent back in order to the initiating chip, and a combined snoop-response broadcast is sent back to all of the snoopers. Multiple levels of snoop filtering are supported to take advantage of the locality of data and processing threads. This approach reduces the amount of interlink bandwidth required, reduces the bandwidth required for system-wide command broadcasts, and maintains hardware enforced coherency using a single-snooping protocol. When the transaction cannot be completed coherently using chip scope, the coherency protocol forces the command to be re-issued to an increased scope of the system.

8.1 SMP Interconnect Features

8.1.1 General Features

- Master command/data request arbitration.
- Command requests are tagged and broadcast using a snooping protocol that enables high-speed cache-to-cache transfers.
- Multiple command scopes are used to reduce the bus-utilizations system wide. The SMP interconnect architecture uses cache states indicating the last known location of a line (sent off chip), information maintained in the system memory (memory domain indicator [MDI] bits), a coarse grained directory that indicates when a line has gone off the chip, and combined response equations that indicate if the scope of the command is sufficient to complete the command or if a larger scope is necessary.
- The command snoop responses specified by the SMP interconnect implementation are used to create a combined response that is broadcast to maintain system cache state coherency. Combined responses are not tagged. Instead, the order of commands from a chip source, using a specific command-broadcast scope, is the same order that combined responses are issued from that source. The order is also affected by the snoop bus usage as well.
- Data is tagged and routed along a dynamically selected path using staging/buffering along the way to overcome data routing collisions.
- Command throttling and retry command back-off mechanisms for livelock prevention.
- Multiple data links between chips are supported (link aggregation).

8.1.2 POWER9-Specific Features

Some features for the POWER9 SMP interconnect are as follows:

- Command broadcast scopes (such as, snoop filtering)
  - Local Node Scope (LNS): Broadcast within a local chip with nodal scope. Node is defined as one chip.
  - Remote Node Scope (RNS): Broadcast to a local chip and targeted chip on a remote group.
  - Group Scope (GS): Broadcast to a local chip with access to the memory coherency directory (MCD).
  - Vectored Group Scope (VGS): Broadcast to a local chip and targeted remote chip.
- 1 - 2 socket system configuration support
- 4× snoop bus support
- MC FastPath support
- 256 Local master (LM) system-pump queue size (64 per snoop bus)
- 256 Group master (NM) group-pump queue size (64 per snoop bus)
- Service processor accessible SCOM registers for configuration setup

8.1.3 On-Chip Features

- Six EQ core chiplets. Each chiplet contains four cores with a shared PBIEQ chiplet interface.
  - Asynchronous chiplet with a128-byte incoming data port and 64-byte outgoing data port.
  - Four cache-line PBI data buffers (incoming and outgoing data ramps to and from the EQ chiplet)
- Four memory controller (MC) chiplets (2× data port)
- Synchronous chiplet I/F (3× NPU, 3× PE, 2× CXA, nMMU, INT, MCD, VAS, NX, TP, Fabric master) (1× data port)
- Centralized command-request arbitration
- Dynamic command rate throttling
- TLBI tokenizer
- Decentralized data-request arbitration
- Eight horizontal, 32-byte data buses
- 32-byte data arbitration size; unit specifies total transfer size

8.1.4 Off-Chip External SMP Features

- 2× (X bus) 2 × 15-bit or 1 × 15-bit EDI + inter-group links at 16 Gbps (asynchronous clocking)
  - 1.0 M length (module + board)
- Aggregate data-link support

8.1.5 Power Management Features

- 1× - 4× core chiplet frequency support
- EQ chiplet power-management support
- Dynamic PHY power-savings support

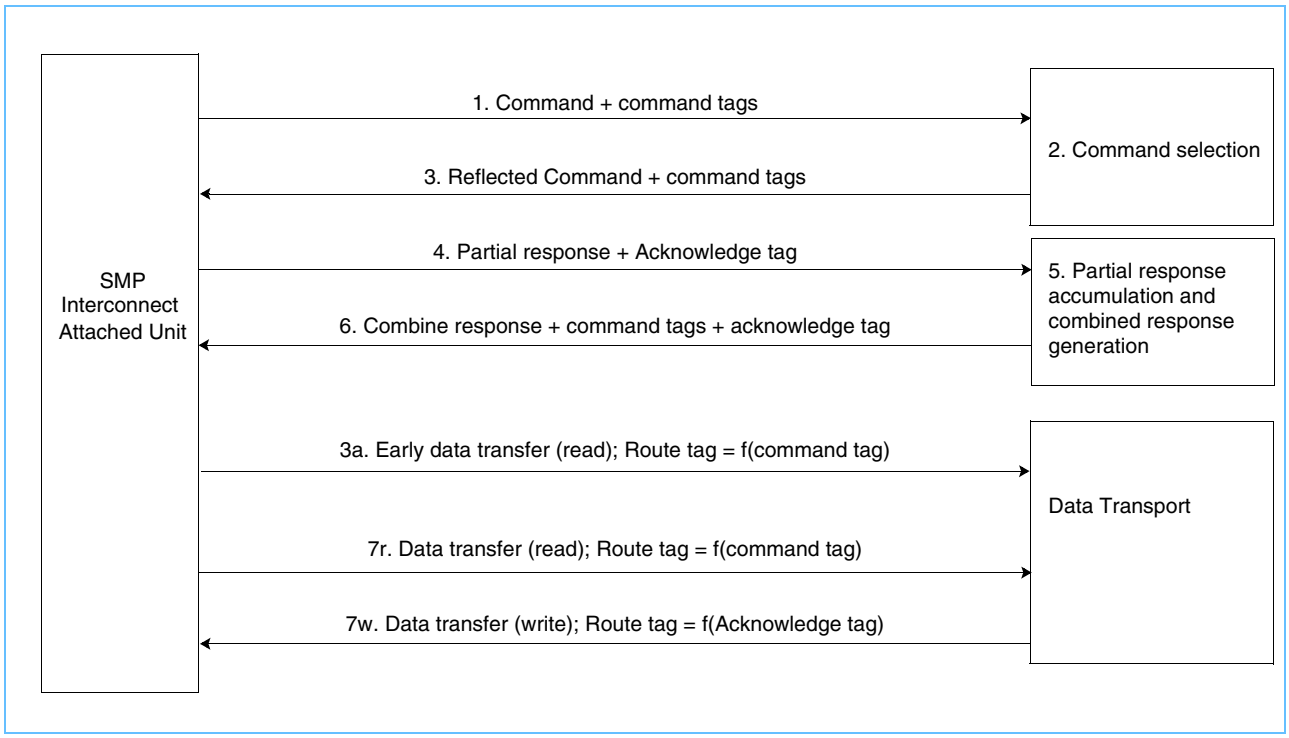
8.1.6 RAS Features

- CRC link-level retry on external SMP links
- 100% ECC protection on internal data flow
- Hang recovery mechanism
- Trace array
- Performance monitor
- FIR error reporting
  - Protocol errors
  - Underflow/overflow checkers
  - Asynchronous drop/repeat checkers
  - Parity checkers on coherency register files
- Error injection
  - Single- and double-bit errors on external SMP links

8.2 SMP Interconnect Architecture Coherency Protocol

Figure 8-1 illustrates the SMP interconnect command and data sequence. To simplify the following discussion, the topology of the system assumes that all SMP interconnect attached units are serviced by a single instance of the SMP interconnect controller, and all responses from the SMP interconnect controller are seen by all SMP interconnect attached units in the same cycle.

Figure 8-1. SMP Interconnect Coherency Protocol



The block on the left of the figure represents all the masters and slave attached units in the system. The blocks on the right of the diagram indicate functions provided by an SMP interconnect implementation. A brief overview of the protocol follows:

1 A unit attached to the SMP interconnect (master) places a command request on the command interface to the SMP interconnect. The command request specifies the transaction type (tType), as well as an identification of the requester that is provided in the transfer tag (tTag).

2 The SMP interconnect control logic selects one of the commands presented by all of the masters of this chip and reflected commands received from other chips as the next command to be issued.

3 The selected command becomes a reflected command and is visible to all SMP interconnect attached units (snoopers). If the selected command is from one of the masters of this chip, the reflected command is queued for transmission to all of the other chips in the system.

3a For read operations, there are cases where a holder of the data can determine without waiting for the combined response that it is the source of the data. In those cases, the holder of the data transfers the data before the SMP interconnect controller issues a combined response. This is referred to as "early data".

The SMP interconnect specifies that data routing is based on destination addressing. The address contained in the route tag specifies the destination SMP interconnect attached unit. The route tag is derived from the transfer tag described in step 1.

The order in which read data is returned to the master might not be in command order. Because the route tag is the same as the original command tag, the unit is responsible for associating the data with the command.

4 A fixed amount of time (t_{snoop}) after the reflected command has been issued, all SMP interconnect attached units (snoopers) on this chip provide a partial response and an acknowledge tag. The acknowledge tag is provided only for write operations.

5 The SMP interconnect control logic combines the partial responses from all the chips within the commands original broadcast scope and generates a combined response.

6 The combined response, with the original command tag and the acknowledge tag, is sent to all snoopers on this chip and queued for transmission to all the other chips in the system.

The combined response indicates to the master the success or failure of the operation and, if any, the state transition for any line requested as well as any subsequent action the master takes.

Units that hold data that is specified by the operation, but were not able to determine if they were to provide the data based solely on the command and the state of the line held, examine the combined response to determine if they are to provide the data.

7r At some later time, read data is moved for the read command. The route tag used is derived from the original command tag. The order in which read data is returned to the master might not be in command order. The route tag contains the original command tag, which allows for this out-of-order property.

7w At some later time, the write data is moved for the write command. The route tag is derived from the acknowledge tag that was provided by the slave performing the write operation. The master provides the data. Note that the order in which write data is provided to the slave might not be in command order. The route tag contains the acknowledge tag, which allows for this out-of-order property.

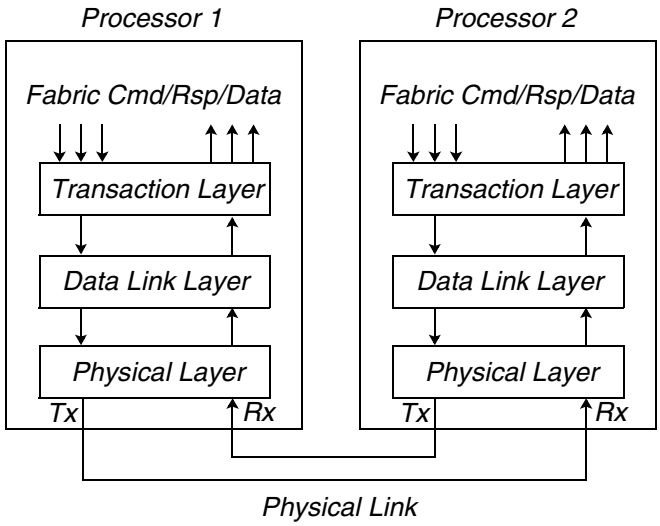
Steps 3a, 7r, and 7w are data transfer steps and occur only for read or write type operations. The order of data movement is not required to follow the order that reflected commands are issued. This is accomplished by tagging the data with an identifier and allows for more efficient use of the data transport facilities provided by the SMP interconnect.

8.3 External POWER9 Fabric

The off-chip POWER9 Fabric supports up to two SMP X-bus links (X1:X2). The X-bus links connect up to two POWER9 processor chips in a system. One X link carries both coherency traffic and data and is interchangeable as inter-group processor links. The second X link can be configured as an aggregate data-only links.

Figure 8-1 shows the protocol layers.

Figure 8-2. Protocol Layers



8.4 Terminology

Table 8-1 defines some common terms used in this section.

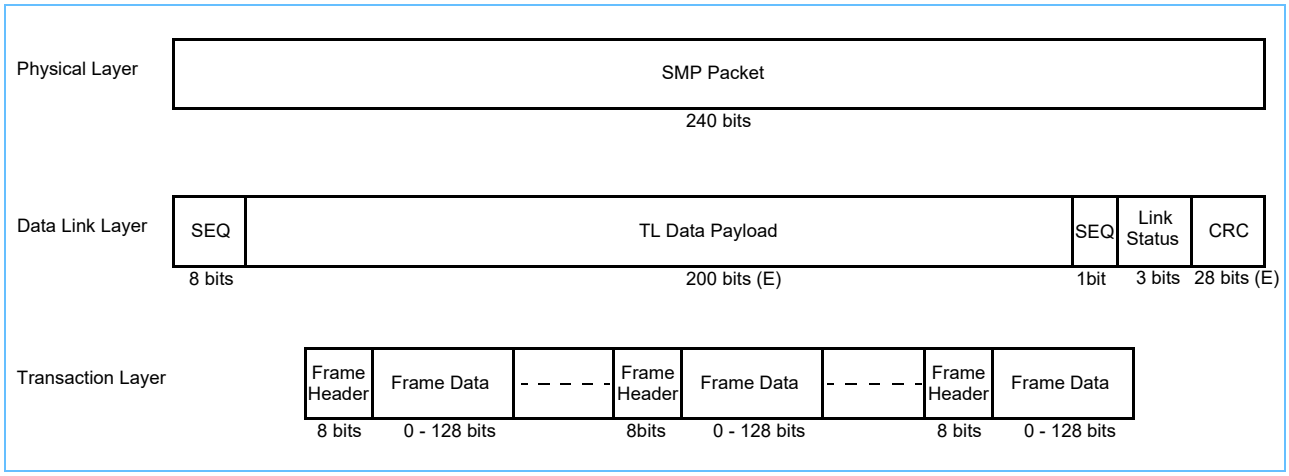
Table 8-1. Terminology

Term	Description
Lane	Single Tx/Rx bit.
Link	Consists of multiple bit lanes organized by protocol layer in packets (DLL).
Single Link	Single link interconnect between two processors.
Paired Link	Pair of link interconnects between two processors.
Packet	Data payload generated by the data link layer.
Frame	Data payload generated by the transaction layer.

8.5 Protocol Layer Payload

Figure 8-3 illustrates the protocol layer payload.

Figure 8-3. Protocol Layer Payload



8.5.1 Physical Layer

The PHY layers supported for fabric external SMP links on the die are listed as follows:

- 2 × EDI+ at 16 Gbps (also known as, Electrical X1:X2) {Asynchronous to Nest Clock}
  (2 × 2 × 15 bit for the X-bus)

The PHY layer includes the transmitter and receiver, serializer/deserializer, and receiver clock recovery circuitry. The PHY initialization includes continuity checking, calibration, equalization, and deskew.

8.5.2 Data Link Layer

8.5.2.1 Electrical Data Link Layer

The electrical data link layer (ELL) interfaces to the EDI+ PHY. The latter is a "thick" PHY class that includes all spare/shadow maintenance. The ELL is responsible only for the CRC insertion/checking, replay buffers, and link layer retry protocols.

8.5.3 Data Link Layer Packet Format

The data link layer packet is 30 bytes (240 bits).

The service packet identifies the special packets required for training and other uses. The payload of a TOD packet contains a byte plus an ECC to correct errors and make the TOD packet more reliable. The link fail packet has data to indicate the reason the link should be brought down. The T-start and T-complete packets have a payload data byte used to match a received T-complete with the transmitted T-start. The other service packets have no other data payload (set to '0's).

For the electrical links, the ACK for a transmitted packet appears a fixed number of cycles after the packet is transmitted (determined during initialization), because the two sides are in synchronization. When a packet is received, the ACK/NAK indication for it is placed in the next packet transmitted.

The logical link number keeps the links and their sequence numbers separate, even when both halves of a link pair are physically sent on one link. When a packet has good CRC, it is checked for the sequence number based on the link number received, and then passed to that extractor for processing.

A 9-bit sequence number supports 256 outstanding packets. By not using half the sequence numbers, a received packet can be determined to be a duplicate or missing packet. Using a sequence number encode for service packets means 255 packets can be outstanding.

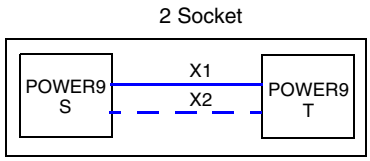
8.5.4 Transaction Layer

The transaction layer (TL) has a common 128-bit interface to the electrical data link layer. The transaction layer includes the framing function (transmit) and parsing function (receive). The TL interfaces with the internal SMP interconnect command/response/data buses.

8.5.5 POWER9 Fabric SMP Topology

Figure 8-4 illustrates the external SMP topology.

Figure 8-4. External SMP Topology



Note: Dotted line indicates aggregate data-only link.

8.5.6 Protocol and Data Routing in Multi-Chip Configurations

The SMP ports configured for coherency are used for both data and control information transport. The use of the buses is as follows:

1. The chip containing the master that is the source of the command issues the reflected command and the combined response to all other chips in the SMP system. Partial responses are collected and returned to the chip containing the master.
2. Data is moved point-to-point. For read operations, the chip containing the source of the data directs the data to the chip containing the master. For write operations, the chip containing the master, directs the data to the slave that performs the write operation. The routing tag contains the chip and unit identifier information for this purpose.

8.6 POWER9 Coherency Flow

8.6.1 Broadcast Scope Definition

Table 8-2 describes the physical broadcast scope and the equivalent coherency scope.

Table 8-2. Broadcast Scope Definition

Command Scope		Physical Broadcast
LNS	Local node scope	Local chip
GS	Group scope	Local chip
RNS	Remote node scope	Local chip and targeted remote chip[1]
VGS	Vectored group scope	Local chip and remote chip

1. Requires X-bus BAR lookup.

8.6.2 Address Definition

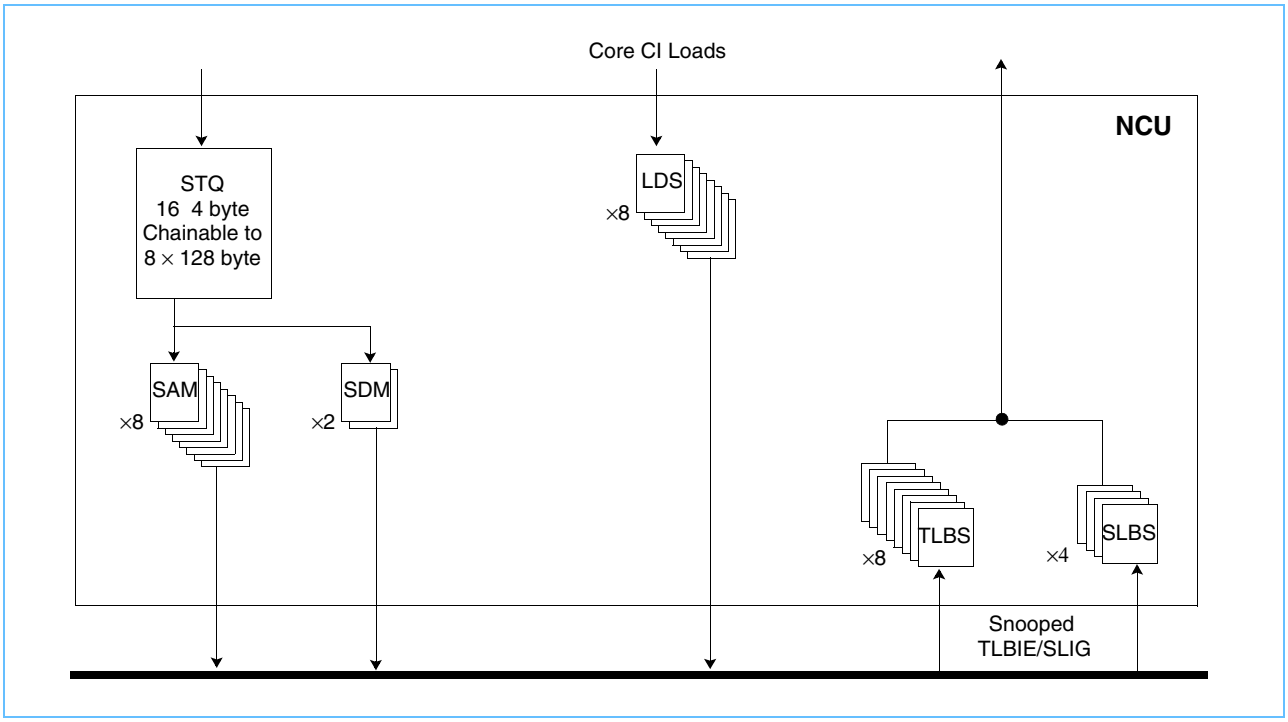
Figure 8-5 illustrates the POWER9 system real-address map.

Figure 8-5. POWER9 System Real-Address Map

9. NCU

The POWER9 Non-Cacheable Unit (NCU) is responsible for processing noncacheable load and store operations (load and store operations with a WIMG "I" bit setting of '1' as described in the Power ISA), word and doubleword load and store atomic instructions (**lwat**, **ldat**, **stwat**, **stdat**), and certain other uncacheable operations such as **tlbie**, portions of the various **sync** and **ptesync** instructions, and so on. All of these instructions support the behavioral definitions given in the Power ISA. One NCU unit is instantiated per pair of simple four thread cores. This NCU handles appropriate operations for all eight threads in the two associated simple cores.

Figure 9-1. NCU Block Diagram



The POWER9 NCU provides one dedicated cache-inhibited load station (LDS) per thread to process cache-inhibited loads and load word or doubleword atomics (**lwat**, **ldat**). Cache inhibited loads (whether guarded [meaning the G bit of WIMG is set to '1'] or not) and load atomics are neither gathered nor are they reordered in the POWER9 implementation. Although, with the exception of guarded cache-inhibited loads, they might be in future implementations. If ordering and/or non-gathering is required on unguarded caching-inhibited loads or on load atomics, appropriate barriers should be inserted to ensure future compatibility.

For cache-inhibited stores and store word and doubleword atomics (**stwat**, **stdat**), a store queue (STQ) consisting of sixteen 64-byte store gather stations is provided. The store gather stations are shared across the eight core threads and hardware prevents any thread from blocking other threads in the store queue. A pair of 64-byte stations can "chain" together to gather up to 128 bytes.

The POWER9 NCU supports gathering and reordering for cache-inhibited stores in the unguarded caching-inhibited (IG = '10') space. In caching-inhibited, but guarded space (IG = '11'), cache-inhibited stores are neither reordered nor gathered as required by the architecture. Similarly, atomic word and doubleword stores (**stwat**, **stdat**) are never gathered, but might be re-ordered.

The POWER9 NCU only gathers naturally aligned 4-, 8-, or 16-byte unguarded cache-inhibited stores. Gathering starts at the first such 4-, 8-, or 16-byte store in a 128-byte region and continues if the next such store (which might be of a different size than the previously gathered store) is contiguous with the previously gathered store and does not cross over a 128-byte boundary.

The POWER9 NCU provides eight store address machines (SAM) that manage the address tenure of the store allowing for up to eight outstanding cache-inhibited or store atomic word or doubleword instructions (**stwat**, **stdat**). A set of two store data machines (SDM) are used to manage the data tenures for the store address machines after the address tenures are complete.[1]

The POWER9 NCU also masters hypervisor broadcast MSGSEND instruction through the store queue and store address and data machines. MSGSEND instructions are treated as a special type of store instruction.

Finally, the NCU provides eight snoop queues (TLBS) to process snooped TLBIE operations and four snoop queues (SLBS) to process SLBIE operations and forward these to the core. The instruction sequences provided in the Power ISA documentation for page table modification should be followed in using the TLBIE and SLBIEG instructions that cause these bus operations.

9.1 NCU Characteristics

9.1.1 Store Queue (STQ)

- 16 $\times$ 64 byte store gather stations (chainable to 8 $\times$ 128 byte gathered stores).
- The gather stations are shared across threads.

9.1.2 Store Modes (IG = '1X')

- IG = '11' mode, stores are done in-order and no gathering is allowed.
- IG = '10' mode, stores can be gathered and reordered.
- Atomic stores (**stwat**, **stdat**) are not gathered but can be reordered.

9.1.3 LOADS

- One outstanding cache-inhibited (guarded or unguarded) load or atomic word or doubleword load (**lwat**, **ldat**) per thread.

1. Address tenures take longer than data tenures and data tenures can be fully satisfied by two store data machines.

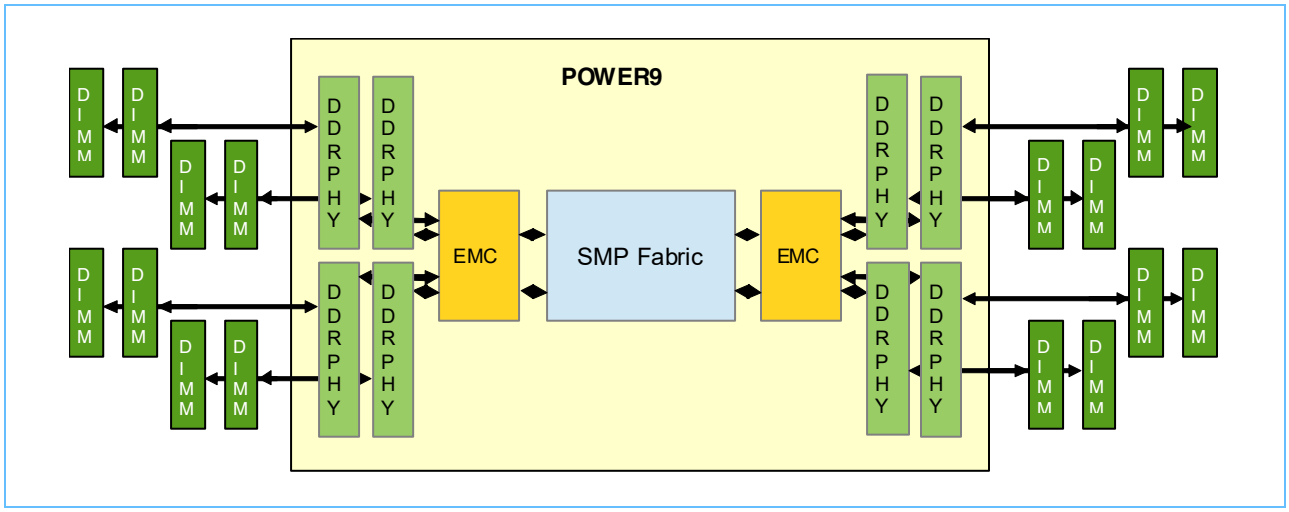
10. Memory Controller

The POWER9 memory controller unit (MCU) provides the system memory interface between the on-chip SMP interconnect fabric and the DDR PHY unit, which in turn directly connects to industry standard memory DDR4 DIMM interfaces. The MCU acts as a slave only. It does not source any commands to the SMP fabric. There are logically eight essentially independent MCUs on the chip interfacing to eight 9-byte wide DDR4 JEDEC standard memory buses. Each memory channel (or 'port') supports up to two DDR4 DIMM slots. Physically, the MCUs are grouped into two instances of an EMC chiplet. Each EMC chiplet contains four MCUs. The EMC is simply a physical level of hierarchy on the chip that contains the MCU plus pervasive logic. The MCUs process 64-byte and 128-byte read and write requests from processor cores and I/O host bridges, 1 - 128-byte partial-line writes, atomic memory operations (AMOs), and also handle address-only operations for the purpose of address protection, acting as the lowest-point of coherency (LPC).

While executing these operations, the MCU is also managing DRAM memory refresh, DRAM power states, and also communicates with the DDR PHY to initiate periodic memory bus calibration sequences.

The eight MCUs on the chip can be configured into one or more address interleave groups. Within each group, the address space is divided into portions, such that each sequential portion is handled by a different MCU in a round-robin fashion. The maximum memory addressing capability per interleave group is 4 TB. The maximum memory addressing per POWER9 chip is 8 TB.

Figure 10-1. POWER9 Memory Controller



10.1 EMC Major Features

- Physical Organization:
  - Each POWER9 chip contains two EMC chiplets
  - Each EMC chiplet contains:
    - Four memory controller units (MCUs)
    - One MCBIST/maintenance/CCS unit shared across the four MCUs
    - Pervasive chiplet infrastructure (scan, trace, and so on)
- SMP Fabric Interface:
  - Four each reflected command, partial response, early combined response, and combined response interfaces. All four buses are snooped in parallel.
  - Four 32-byte read data ramps (one per memory port).
  - Four 32-byte write data ramps (shared across four memory ports).
  - Speculative fastpath read command bypass.
  - Speculative fetch filtering: Five types:
    - SMP Fabric ttype based
    - SMP Fabric hint bit based
    - SMP Fabric command source and scope based
    - Command list queue fullness based (threshold exceeded)
    - Hashed address range based (2048 ranges)
  - Fastpath interface (from local chip L3 caches).
  - Support for 64-byte and 128-byte data transfers.
  - Cache-line interleaving on a 32-byte basis when returning read data to the SMP fabric.
  - Hardware management of domain bits for multi-node systems.
  - LPC coherency support for address-only commands.
  - Delivery of the critical 32 bytes of read data not gated by reading the entire cache line.
  - Secure memory facility (SMF) support. This support accomplishes enabling a programmable configuration of ranges of memory as secure or not secure; and preventing access of secure memory by entities that are not secure.
  - Prefetch drop protocol support.
  - Prefetch promote protocol support.
  - I-side speculation avoidance supported via partial response.
  - Memory ECC bypass protocol for improved latency.
- Pervasive Interfaces:
  - Performance monitor interface
  - Nest (sync) and memory (async) clock domain SCOM interfaces
  - Nest domain debug bus interfaces (to shared trace arrays in SMP fabric logic)
  - Memory domain debug bus interfaces (to EMC contained trace arrays)

- Resources (per MCU port):
  - 76 snoop command list entries
    - 64 read/write memory entries
    - 12 address protection-only entries
  - Sixteen 128-byte read data buffers (managed as thirty-two 64-byte buffers)
    - Read commands to memory stalled on full condition
  - Thirty-two 128-byte write data buffers (managed as sixty-four 64-byte buffers)
    - SMP fabric write operations retried on full condition
  - Thirty-two 128-byte read-modify-write (RMW) data buffers:
    - One of 32 buffers reserved for maintenance operations
    - Managed as a cache to provide for high-throughput AMOs.
    - Up to 62 concurrent 64-byte AMO operations.
- DRAM interface (per MCU port)
- DRAM widths supported: ×4, ×8
  - DRAM densities supported: 4 Gb, 8 Gb, 16 Gb
  - ISDIMM types supported: DDR4 RDIMM, DDR4 LRDIMM (with and without 3D stacking)
  - Maximum DIMM size: 256 GB (2 master ranks × 4-high stacked 16 Gb DRAMs)
  - DRAM command scheduling
  - Support for page-mode reads and writes
  - 1 TB maximum memory capacity per port
  - Speculative read cancel protocol
    - Provides for cancellation of speculative reads if a bad combined response is received before the issuance of the read command to memory.
  - 17 read reorder queue entries (one of 17 reserved for maintenance operations)
  - 17 write reorder queue entries (one of 17 reserved for maintenance operations)
  - DRAM power state controller
    - Power control state status register
    - Programmable dynamic command throttling: credit based.
    - Self-Timed-Refresh (STR) based power reduction.
    - C/A outputs tri-stated in PD or STR modes
    - Programmable minimum/maximum ranks allowed to be powered up
    - Programmable number of idle cycles between commands
    - Emergency power throttle mode
  - DRAM refresh controller
    - Thermal-based dynamic tREFI setting.
    - Hardware support only. Also requires OCC code support.

  - DRAM interface calibration timers.
  - Synchronous and asynchronous operation relative to the nest clock.
  - Support DDR4 bin speeds: 1866, 2133, 2400, 2667.
- RAS features
  - 64-byte memory ECC
    - Dual packet analysis for 128-byte reads
    - Address parity encoded into ECC code
    - Correction of up to one symbol in a known location plus up to two unknown symbol errors
    - Correction of up to one symbol in a known location plus a new ×4 chip kill
    - Correction of one ×4 chip in a known location plus either a known symbol or one unknown symbol error
  - Flexible chip and symbol marking storage
    - Per bank, bank group, slave rank, master rank, DIMM, and port granularities
    - CE retry before new mark placed for confirmation
  - Read operation retry upon detection of uncorrectable errors (UEs)
    - 64-byte reads retried as 128-byte reads under some conditions for improved RAS
  - DDR4 CRC support on writes
  - Maintenance engine supports runtime diagnostics and error logging
    - Includes background memory scrubbing
    - Eight MCE symbol counters per port.
  - Time-out counters for various events.
  - DIMM RCD parity error handling
- Manufacturing, test, and bringup
  - Firmware driven system memory built-in-self-test (MCBIST) engine
  - Maintenance engine for IPL memory initialization and diagnostics
  - Configured command sequencer (CCS)
    - Provides user-defined sequences for memory interface debug
    - Can be used concurrently with mainline operations (for MRS commands, and so on)
  - Extensive error-injection capabilities to provide means to test all FIR bits
  - Debug buses and trace arrays

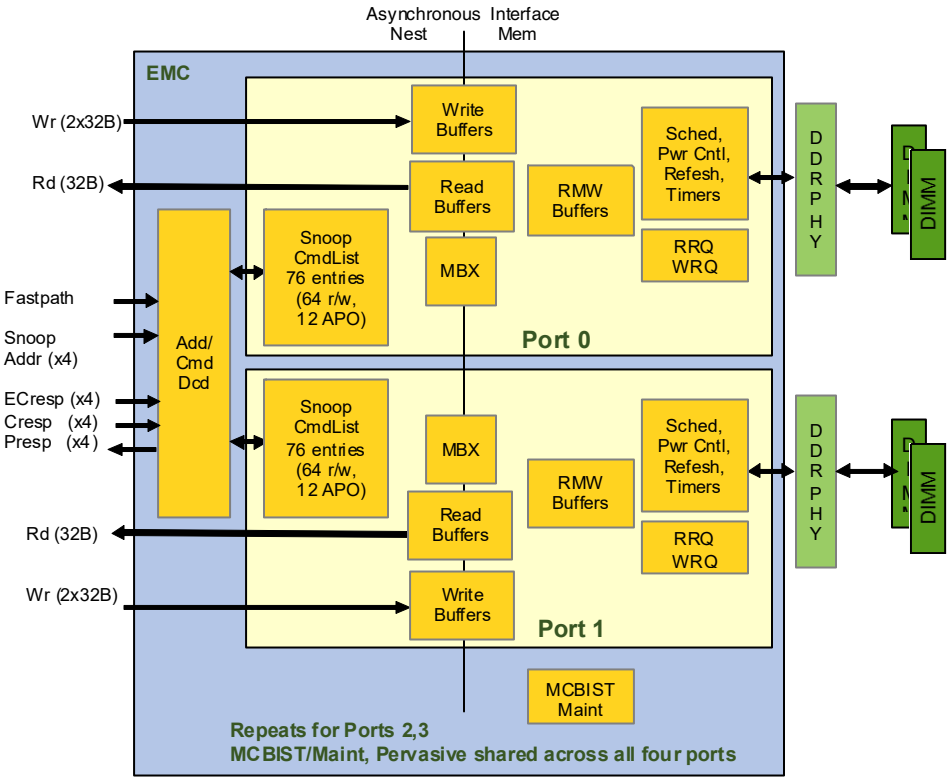
Table 10-1. Frequencies

Synchronous/Asynchronous Mode	Nest Frequency (GHz)	Bin Speed (MTps)
Asynchronous	1.600	1866, 2133, 2400
Asynchronous	1.866	1866, 2133, 2400, 2667
Asynchronous	2.000	1866, 2133, 2400, 2667
Asynchronous	2.133	1866, 2133, 2400, 2667
Asynchronous	2.400	1866, 2133, 2400, 2667
Synchronous	1.866	1866
Synchronous	2.133	2133
Synchronous	2.400	2400

Table 10-2. Allowable DIMM Mixing

Type of Mixing	On any Single Port	Across Different Ports	Comments
DRAM width (×4, ×8)	No	Yes	
DIMM size (# ranks)	No (1)	Yes	1. Single-port mixing barred due to loading issues.
DIMM size (DRAM density)	Yes (2)	Yes	2. Only power of 2 densities can be mixed.
DIMM type: RDIMM and LRDIMM	No	Yes	
Stacking: 3DS and non-3DS	No	Yes	
DIMM speeds	No (3)	No (3)	3. However, ports controlled by one EMC can run at a different speed than ports controlled by the other EMC.

Figure 10-2. EMC Logical Partitioning



10.2 Basic Configuration/Grouping

The MCU architecture allows for 1, 2, 3, 4, 6, or 8 MCUs to be grouped together for address interleaving. As a group, the MCU ports then hash a contiguous address space amongst themselves to more efficiently distribute the memory workload.

For 2, 3, 4, 6, or 8 MCUs to be grouped, the total amount of memory defined by each MCU's primary memory configuration facilities must be the same. The DIMM configurations and sizes that make up the total amount of memory can be different, but the total memory size plugged behind each MCU in the group must be the same.

The total amount of physical memory behind an MCU can be less than the memory size specified by that MCU's primary memory configuration register. However, if the MCU decodes an address that falls within its programmed address range, but does not decode to a valid physical DRAM address, a Fault Isolation Register (FIR) bit is set.

Each MCU receives a 56-bit address for each snoop operation from the SMP fabric, and forwards a 40-bit (1 TB) logical address to the DRAM address translation logic. This logic converts the real address to a DRAM rank/bank/row/column address.

The address interleaving granularity, meaning the amount of contiguous addresses handled by each group member, is programmable from 128 bytes to 32 KB. For example, if set to 512 bytes, memory accesses to a contiguous 512-byte block of memory are all handled by one group member MCU. Then, the next 512-byte block is handled by the next MCU in the group, and so on.

10.3 Command Dispatch and Snoop Pipeline Collision Detection

For a snooped operation to be dispatched into a MCU command list queue entry, the operation must:

1. First successfully pass through the snoop pipeline collision detection logic.
2. Then pass through the command list fullness logic and L3 prefetch limit logic.
3. Then successfully pass through the MCU address collision rules table.

The memory controller logic compares incoming reflected command and fastpath command addresses to queued addresses in the CAR1 cycle. However, newly dispatched operations do not begin address protection in the ACAM until the CAR4 cycle. The MCU compares addresses in the CAR2 and CAR3 pipeline stages to incoming CAR1 addresses. If the operation in CAR2 or CAR3 matches the CAR1 address, and the CAR2 or CAR3 operation is dispatched, then a retry response is driven for the CAR1 address operation.

The MCU logic compares incoming RCMD ttags to queued tags in the CAR1 cycle to associate a RCMD with a previously dispatched fastpath read. However, ttags for newly dispatched operations do not begin ttag comparison until the CAR4 cycle. The MCU logic compares ttags in the CAR2 and CAR3 pipeline stages to incoming CAR1 ttags. If the operation in CAR2 or CAR3 matches the CAR1 ttag, the CAR2 or CAR3 fastpath read operations is dispatched, and then the command in CAR1 is associated with the CAR2 or CAR3 fastpath operation.

10.4 Epsilon Protection

The MCU provides epsilon protection for command list operations. Epsilon protection guarantees that the command list stays active; protecting a cache-line address from the time a good combined response is received to the time the epsilon counter expires. The epsilon counters are loaded when a combined response is received and decremented once every four nest cycles. The starting point of epsilon counting occurs in the cac3 cycle of a good combined response, which might not align with the epsilon count decrement pulse. Thus, the actual epsilon protection window could be up to four nest cycles greater than the programmed value.

The epsilon protection window ends when the epsilon count reaches zero. The epsilon protection window can end before the command list queue entry goes idle. An example is a non-speculative read operation. Read data is returned after the epsilon window expires for cases where the epsilon value is less than the read latency of the operation.

10.5 Read Speculation Filtering

The MCU employs read speculation under certain conditions to improve average memory latency. A speculative read is a memory read operation where the read to memory is initiated before the coherency response is received from the SMP fabric. Speculation filtering is used to determine when to speculate on a read operation or not. There are five types of filtering:

- Filtering by Ttype: The enabling of read speculation for various fabric ttypes is programmable. Because only speculative read types are received on the fastpath interface, the entire fastpath interface can also be disabled.
- Filtering by command source and scope: The enabling of speculative reads for non-DMA groups or system pump operations from a local chip master is programmable. A group or system pump read is likely generated because the cache line was not found on-node. Group and system pump reads issued by off-node masters are issued speculatively if the read operation passes the other speculation filters.
- Filtering using the SMP fabric hint bit: Read speculation is disabled if the fabric hint bit is set, indicating a local cache found the line in the In/Ig state. This indicates that the line has been intervened on-node (In), or off-node (Ig). In either case, a cache is likely to contain the cache line. The enabling of this type of filtering is programmable. The hint bit is defined for certain ttypes as specified in the SMP fabric architecture document.
- Filtering if the number of read command-list entries exceeds a threshold: The number of active read operations queued in the command list beyond which additional reads will not be performed speculatively is programmable.
- Filtering by hashed-address range: The MCU contains 2048 latches that are used to aid in the speculation decision. Each of the 2K latches represents a specific address hash. If the MCU returns read data whose address hashes to a particular bit, that bit is set. If a read that was initiated speculatively receives a coherency response indicating that the MCU will not return data, the bit selected by that read's address hash is reset. This setting/resetting of a bit in the filter can be modified by an LFSR increment/decrement filter.

Latches can be reset immediately if a snoop operation with the hint bit set is dispatched to the counter's hashed address range. If the latch is set, speculative reads to that hashed address range are allowed. If not set, speculative reads to the hashed address range are inhibited.

10.6 SMP Fabric Fastpath Interface

The fastpath interface is an early version of the reflected command (rcmd) interface, and is sourced from the L3 cache units. Only read commands are sent via the fastpath interface. If no reflected commands targeting an MCU are received in the same cycle as a fastpath read, the fastpath read is allowed into the snoop pipeline. Otherwise, it is discarded. If the fastpath command passes the address protection and speculation filters, it is dispatched to the command list queue and forwarded to the DRAM interface command scheduler. At the time a fastpath read command is dispatched, address protection for the fastpath operation starts, and the ttag is loaded into the tag CAM register associated with the dispatched command list index. A partial response is not generated.

A reflected command and combined response always follows a fastpath command. If the fastpath command has been dispatched to a command list entry, the subsequent reflected command is detected with a ttag compare. At the time of the reflected command, the operation ticket is loaded into the ticket register associated with the command, and a SMP fabric partial response is generated. The reflected command can be retried subject to the address-collision rules.

A fastpath command can be dropped depending in the command list fullness, or if a spare rcmd slot is not available at the time the fastpath command is received.

A fastpath command is always issued with nodal scope. The rcmd associated with the fastpath command is also always issued with nodal scope. If a fastpath command is issued with non-nodal scope, the MCU sets a FIR bit.

10.7 Read Data ECC Bypass

To improve read latency from memory, data returned from a read operation can be forwarded directly to the SMP fabric, bypassing the memory ECC check logic and the read data buffers, if all of the following are true:

- The 'speculative' (ECC bypass capable) bit is set in the fabric secondary command tsize encode.
- The targeted DRAM rank is in an ECC state that supports ECC bypass (that is, no chip mark placed).
- Good combined response is received before the arrival of the read data from memory or the read data is of the type exclusive.
- The scope of the read operation is nodal.
- ECC bypass is enabled for the read ttype.

10.8 Atomic Memory Operations

Each MCU supports the processing of a set of atomic memory operations (AMOs). AMOs are read-modify-write type operations, and as such have store data associated with them. The MCU implements an arithmetic logic unit (ALU) that operates on the store and memory fetch data. The intent is to provide high throughput of multiple AMOs targeting the same address. There are two major features that enable this high level of throughput:

- Multiple AMOs targeting the same address are allowed to be queued in the command list queue such that they are not serialized via retries on the SMP fabric interface. Each AMO command must be performed atomically, but there is no requirement that they be executed in the order in which there are snooped on the SMP fabric.
- The 31 entry $\times$ 128-byte RMW buffer is managed as a cache. When one AMO completes, its data is maintained in the RMW buffer so that a subsequent AMO to the same address receives its data directly from the buffer instead of having to re-fetch the data from memory. This caching capability is also used for partial writes, providing for the 'gathering' of multiple partial writes before writing back to memory, and for MDI bit updates. The command list logic controls the deallocation of the RMW buffer entries in response to snoop traffic and requests from the memory interface sequencer (SRQ) to free up room in the RMW buffer.

AMOs are only supported in 4 and 8 byte sizes. Both big-endian and little-endian modes are supported. There are 23 different AMO types supported:

- Store Add
- Store XOR
- Store OR
- Store AND
- Fetch and ADD
- Fetch and XOR
- Fetch and OR
- Fetch and AND
- Compare and swap equal
- Compare and swap not equal
- Swap unconditional
- Fetch and increment bounded
- Fetch and increment equal
- Fetch and decrement bounded

- Store twin
- Store maximum unsigned
- Store maximum signed
- Store minimum unsigned
- Store minimum signed
- Fetch and maximum unsigned
- Fetch and maximum signed
- Fetch and minimum unsigned
- Fetch and minimum signed

All of the "fetch and …" operations return the unmodified memory data to the SMP fabric, except the fetch and increment bounded, fetch and increment equal, and fetch and decrement bounded. Those three operations return either the unmodified memory data, or the minimum unsigned-integer value based on the result of a compare of two adjacent granules of memory data.

From an MCU data flow/sequencing perspective, AMO operations are always 64 bytes in length, and partial writes and MDI updates are always 128 bytes in length.

10.9 Write Operations

The MCU command list machines wait for both a good combined response and write data arrival before issuing a write command to the memory interface command sequencer/scheduler (SRQ). Up to four CRESPs and four write data available indications can occur in a single cycle.

The SRQ treats 128-byte writes as two 64-byte writes, and signals write done on a per 64-byte basis. The write buffer allocation logic deallocates a 64-byte write buffer with each write done, and the command list counts two write done indications for each 128-byte write operation.

Each MCU port command list dispatch block maintains a count of available 64-byte write buffers. The dispatch block is responsible for not overrunning the write data buffer. The dispatch block always waits for at least two 64-byte buffer slots to be free before dispatching a new write operation to avoid a stream of 64-byte writes blocking a 128-byte write.

The write done indicator from the SRQ is used to decrement the count of 64-byte write buffer slots available. Write done is issued after the SRQ has waited a programmable guard time to ensure that the write to memory does not have to be retried due to a DIMM RCD error.

10.10 Prefetch Promote/Drop Protocol

To increase the efficiency of pre-fetch commands to memory, the MCU supports a prefetch drop/promote protocol. If a prefetch operation is experiencing a long latency and a demand load to the same cache line is issued before the return of the prefetch data, the prefetch can be promoted to demand read status by increasing the priority of the queued command in the MCU. Conversely, the longer an outstanding prefetch sits in a queue without completing, the less likely it is that the prefetch data will ever be needed by the core. In this case, the MCU has the ability to drop a prefetch based on a decrementing timer.

10.10.1 Prefetch Promote

The L3 cache can send a prefetch promote command after combined response for the original prefetch command if it chooses to update a low or high priority prefetch to a low or high priority demand read. This increases the priority of the prefetch command and prevents it from potentially being dropped. The prefetch promote occurs in the MCU non-speculative queues and the SRQ read reorder queues.

The SMP fabric prefetch promote command contains the master tag associated with the prefetch. If the rtag is active in the MCU tag storage at the time of the promote command, and the prefetch operation is found in the MCU command-list queue, the operation is updated from low or high priority prefetch, to a low priority demand read. If the rtag is not active in the TCAM at the time of the promote command or the operation cannot be found in the command list queue, no action is taken.

The promote operation is not always precise in the MCU. The promote can be missed if the prefetch command is crossing the asynchronous boundary (Nest → Mem) between the command list queue and the SRQ or can be applied to the next prefetch operation using the same command list entry. If this happens, data integrity is maintained, but the promote might be missed. However, for the vast majority of the timing cases, the promote is precise. The enabling/disabling of the MCU prefetch promotion function is programmable.

10.10.2 Prefetch Drop

The SMP fabric prefetch commands contain a 2-bit 'confidence level' that is encoded into the secondary encode field of the ttype. The MCU converts this value into a 3-bit 'prefetch-drop' field that is carried through the command list and SRQ queues.

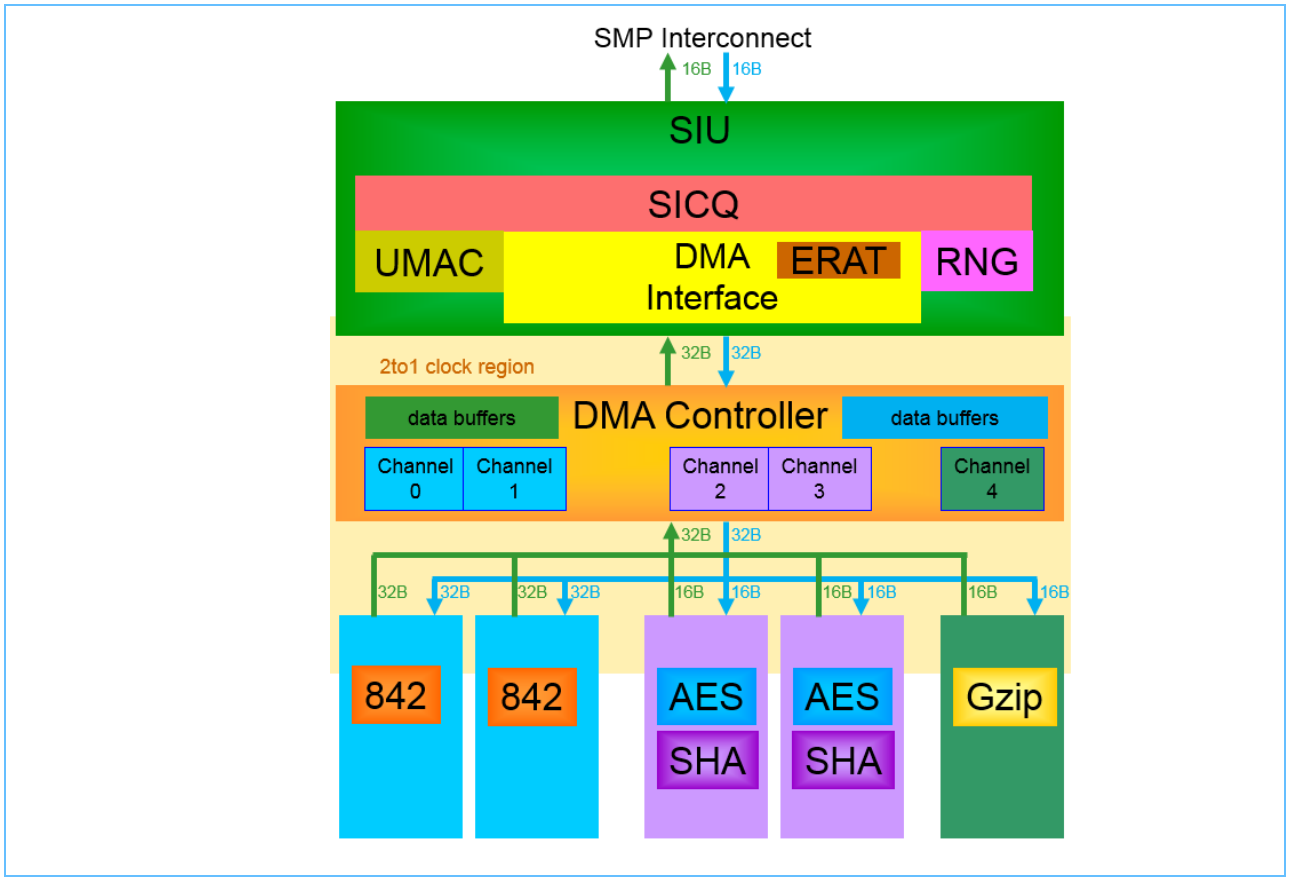
The MCU command queue prefetch drop value is decremented every "N" nest cycles (where N is programmable). When the drop value reaches '000', the prefetch operation is in the drop state and is not forwarded to the next command queue or to memory. If the prefetch is dropped in the command list queue or the SRQ's read-reorder queue, a command is sent to the read data flow logic instructing it to send a 32-byte dummy read-data packet back to the requesting prefetch master. The SRQ prefetch drop value is decremented at half the rate of the command-list queue's drop value.

The drop operation is not always precise in the MCU. The prefetch drop value decrement could be missed if the prefetch command is crossing the asynchronous boundary (Nest → Mem) between the command list queue and the SRQ at the time the decrement pulse is active. If this happens, data integrity is maintained, but it could take longer for the prefetch to be dropped. However, for the vast majority of the timing cases, the drop value decrement is precise. The enabling/disabling of the MCU prefetch drop function is programmable.

11. Nest Accelerator

The Nest Accelerator unit (NX) consists of cryptographic and memory compression/decompression engines (coprocessors) with support hardware. *Figure 11-1* shows a block diagram of the NX.

Figure 11-1. NX Block Diagram



11.1 Features

The NX coprocessors and their features are as follows:

Cryptographic Engines

Advanced encryption standard (AES) engine:

- Modes: Electronic Codebook (ECB), Cipher Block Chaining (CBC), Count (CTR), Counter with CBC-MAC (CCM), Galois Counter Mode (GCM), XCBC-MAC-96 (XMAC)
- Key lengths: 128 bits, 192 bits, 256 bits
- Two engines

Secure Hash Algorithm (SHA) engine:

- Modes: SHA-1, SHA-256, SHA-512, Message Digest 5 (MD5).
- Keyed-hash message authentication code (HMAC) supported for SHA.
- Two engines.
- Combined AES and SHA operations are supported on a single movement of data through an AES/SHA engine pair.

Random Number Generator (RNG):

- All digital design with dual noise sources
- NIST 800-90B draft standard compliant
- Hardware random number (RN) conditioning
- Continuously running health tests with noise source failover on test failure
- Produces 64-bit conditioned random numbers (CRN) and raw random numbers (RRN) readable by the **darn** instruction

Compression and Decompression:

- 842 compression/decompression
- IBM-proprietary algorithm with 8-byte, 4-byte, and 2-byte phrase parsings
- High throughput and low latency with good compression and silicon efficiency
- Two engines

Gzip Compression and Decompression:

- Industry standard DEFLATE RFC 1951 compliant
- Supports RFC 1950 (zlib) and RFC 1952 (gzip) file formats
- Fixed and dynamic Huffman table support
- Assist for dynamic Huffman table creation
- Ability to suspend an operation when a byte count limit is hit
  - Software can resume operation from a suspend point after adjusting the job parameters
- Bypass mode for data move
- High throughput with better compression efficiency (relative to 842)

Each one of the AES/SHA, 842, and Gzip units consist of a coprocessor type (CT). As such, NX has three coprocessor types.

To support coprocessor invocation by user code, use of effective addresses, high-bandwidth storage accesses, and interrupt notification of job completion, NX includes the following support hardware:

SMP interconnect unit (SIU):

- Interfaces to SMP interconnect and direct memory access (DMA) controller
  - Provides 16-bytes per cycle data bandwidth per direction to both
- Employs SMP interconnect common queue (SICQ) multiple parallel read and write machine architecture to maximize bandwidth
  - 16 DMA read machines, each with one cache-line data buffer storage
  - 16 DMA write machines, each with one cache-line data buffer storage
  - Three UMAC read machines, each with one cache-line data buffer storage
  - Three UMAC write machines, each with one doubleword data buffer storage
  - Eight RN read machines with buffer storage for eight doubleword CRN plus eight doubleword RRN
- User-mode access control (UMAC) coprocessor invocation block
  - After the Virtual Accelerator Switchboard (VAS) accepts a CRB that was initiated by a copy/paste instruction, the UMAC snoops the VAS's notification for an available coprocessor request block (CRB or job).
  - Supports one high- and one low-priority queue per coprocessor type
  - Retrieves CRBs from queues and dispatches CRBs to the DMA controller
- Effective-to-real address translation (ERAT) table stores 32 recently used translations
  - Interfaces to nest memory management unit (NMMU) for address translation services
  - Translates all effective addresses (EA) from DMA controller to real addresses (RA)
  - Returns translation faults to DMA controller
- Snoops **darn** instruction command for RN delivery

DMA Controller:

- Decodes CRB to initiate coprocessor and move data on behalf of coprocessors
- Uses effective addresses for all CRB storage accesses
  - Issues paste command to VAS to dispense CRB with translation fault to per-partition fault queue
- 5-channel data mover, one per each instance of AES/SHA, 842, Gzip engine, with buffers for data to and from engines
- Two CRB queue positions per channel: one for current CRB (currently executing on a coprocessor) and one for pending CRB (awaiting execution)
  - Can prefetch coprocessor parameter block (CPB) and source data for pending CRB
- Provides prefetch hints to memory controller to reduce read latency
- Supports byte-aligned source and target data
- Supports scatter/gather through data descriptor list (DDL)
- Supports interrupt notification on CRB completion
- 16 bytes per cycle data bandwidth per direction to/from SIU
- 16 bytes per cycle data bandwidth toward 842 engines, 8 bytes per cycle toward AES/SHA, Gzip
- 16 bytes per cycle data bandwidth from 842 engines, 8 bytes per cycle from AES/SHA, Gzip

Most of the NX unit operates on the nest clock domain, but the DMA controller operates on a clock period twice that of the nest clock (see "2:1 clock domain" in *Figure 11-1* on page 195). To match the SIU data bandwidth, the DMA controller buses are twice as wide as the SIU buses, as shown in the figure.

11.2 Using NX Coprocessors

NX coprocessors can be invoked through library or operating system kernel calls that use the Power ISA copy/paste facility. See GITHUB for links to both Linux and AIX operating systems, as well as low-level gzip engine invocation details for library or kernel coders.

11.3 Reliability, Availability, and Serviceability

NX integrates many features to improve Reliability, Availability, and Serviceability (RAS), including the following:

- Error correction code (ECC) or parity on all data arrays
- ECC or parity on ERAT and other address-carrying structures
- Parity on key configuration registers
- Control checkers on many state machines and other control structures
- High degree of error tolerance
  - Many errors simply write error CC to CSB and hardware operation continues
  - Ability to unit checkstop on severe error and become benign on the SMP interconnect interface
  - Failover on single RNG noise source fail, fail-safe on dual fail, or other severe error
- Per-channel watchdog timers to detect and terminate hung coprocessor
- DMA hang timer to detect DMA controller hang
- Hang timers on SMP interconnect operations
- Unit checkstop upon VAS or NMMU unit checkstop

12. Virtual Accelerator Switchboard

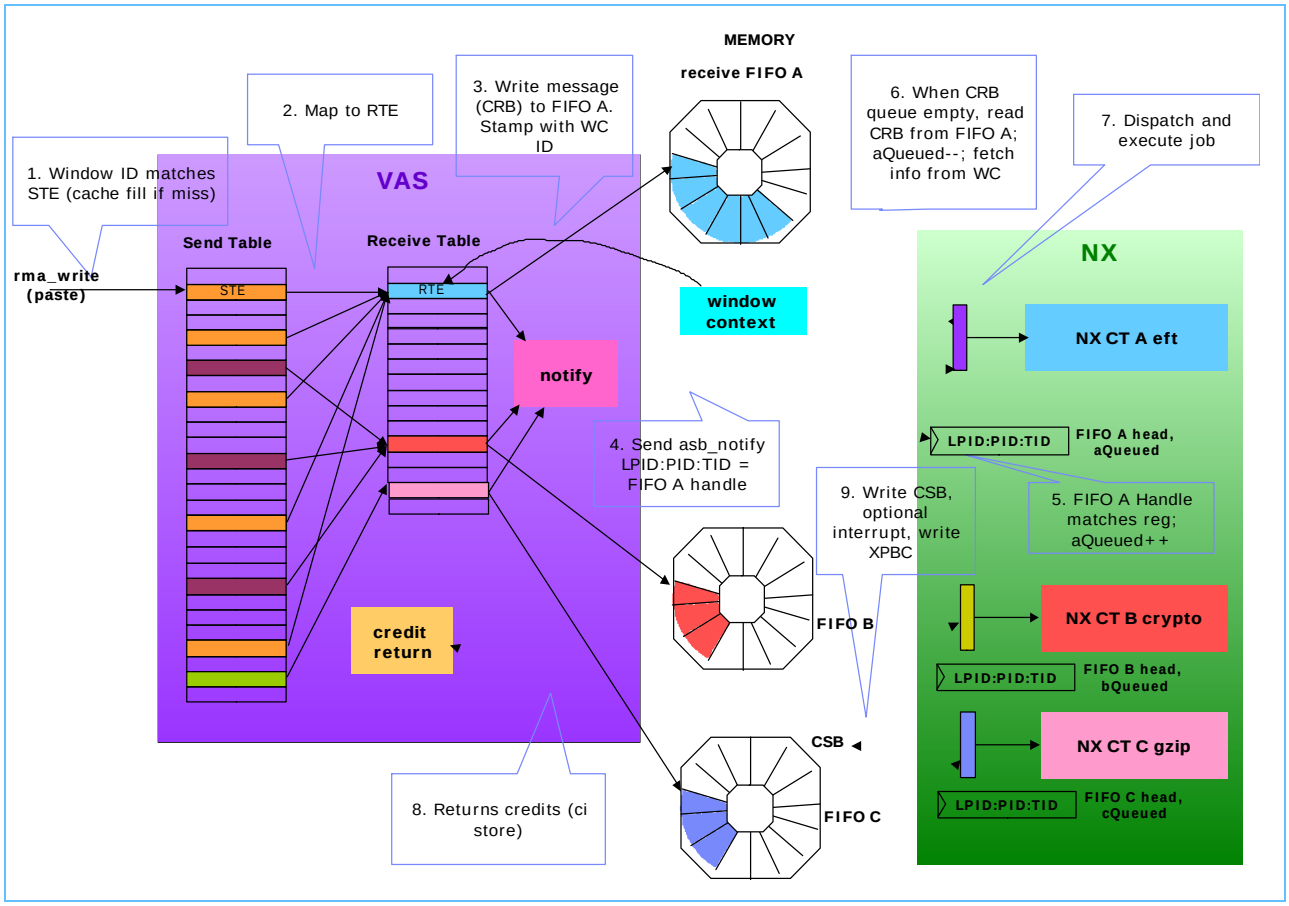
12.1 Overview

The main function of the Virtual Accelerator Switchboard (VAS) unit is to allow user-level software code running on a processor core direct access to the Nest Accelerator (NX) unit accelerator engines without the requirement for an expensive hypervisor call to initiate accelerator usage. This allows user-level code direct access to the cryptographic and compression accelerator engines on the POWER9 chip.

12.2 Flow for NX Invocation Through the VAS

When an operating system wants to initially allow a user process access to an NX accelerator, it must communicate with the hypervisor. The hypervisor sets up a Send Window Table Entry (STE). A send window can be unique per process, or can be shared by an operating system among its many user processes. When the hypervisor establishes the send window, it assigns it a quantity of credits indicating how many operations are allowed at one time, and returns an address handle to the operating system. Then the operating system returns an effective address to the user. This effective address allows the user process to directly post entries to the FIFOs associated with the NX accelerators. A separate send window is necessary for each accelerator a process uses.

Figure 12-1. Flow for NX Invocation through the VAS



1. After a send window has been established, the user process can begin using the NX accelerator. To do so, it must create a coprocessor request block (CRB). The NX specification defines the format of the CRB. This CRB is sent to the VAS unit by using the POWER9 copy/paste instructions. The copy instruction places the CRB into the copy buffer. The user process then issues a paste instruction using the effective address given by the operating system during send window creation to store the copy data to the VAS. The copy data contains the 128-byte CRB. The effective address is translated to a real address by translation hardware in the core. The store to the real address is issued to the SMP interconnect as a remote memory access write (RMA_write) command and has the send window identifier embedded within the real address. The 128-byte RMA_write payload (the CRB) is stored into one of 64 VAS data buffers.

   The VAS has the ability to hold 128 unique window contexts. Upon snooping the RMA_write, the VAS uses the send window identifier to fetch the Send Window Table Entry from memory if not already resident with the VAS window cache logic.

2. The VAS reads the Receive Window Identifier field in the send window context to determine which receive window the send window from the RMA_write points to. Each NX coprocessor type (CT) has a unique receive window corresponding to a unique FIFO for each of the accelerators.

   If the receive window is not cached, it will be fetched from memory.

As shown on the left side of *Figure 12-1* on page 200, many different send windows, each associated with a user process, can point to the same receive window. In fact, all send windows that are using the same NX CT can point to the same receive window, because there is one receive FIFO per CT.

3. Using the FIFO address from the receive window context, VAS stores the RMA_write payload to memory, thereby placing the CRB onto the NX accelerator FIFO. VAS stamps, or overlays, a portion of the CRB with the send and receive window identifiers. NX uses this information when processing the CRB. In particular, the send window identifier in the CRB is used by NX to fetch the send window and obtain translation information for the addresses contained within the CRB.

   The receive FIFOs are implemented as circular queues. After reaching the end of the FIFO, VAS wraps back to the beginning of the FIFO and writes the next entry.

4. After writing the CRB to the FIFO, VAS sends an ASB_notify command on the SMP interconnect. The ASB_notify contains a logical partition identifier (LPID), process identifier (PID), and thread identifier (TID).

5. Each NX FIFO has a particular LPID:PID:TID combination associated with it. When NX snoops an ASB_notify that matches its programmed LPID:PID:TID, it increments the corresponding counter for the associated FIFO, indicating a new work item has been placed on the accelerator FIFO.

6. When an NX CT queue is empty and its counter is nonzero, NX reads the next CRB from the receive FIFO. As soon as the CRB is read from the FIFO, NX does a memory mapped (MMIO) store to the VAS unit to return a credit. VAS ensures that the receive FIFO does not overflow by managing credits. The hypervisor initializes the receive window with credits equal to the number of CRBs that can be stored to the receive FIFO based on the size of the FIFO. VAS decrements the receive credit count when it stores a CRB to the receive FIFO and increments the count when NX returns a credit via MMIO store after NX pulls the CRB off of the FIFO.

   NX uses the stamped information from the CRB to read the send window context from memory and decrements its internal counter.

7. NX dispatches the job to the associated CT, which can have multiple acceleration engines, and executes the CRB.

8. Upon completion of the job, NX returns a send window credit to VAS via an MMIO store. Each send window, when created by the hypervisor, is assigned a number of send credits. This allows the hypervisor to implement quality of service by managing numerous users sharing the same accelerator resource, and preventing one process from using more than its share. When an RMA_write command is received by VAS, VAS decrements the send credit for the associated send window. VAS increments the count when NX completes the CRB and returns a send credit with an MMIO store.

9. NX writes a coprocessor status block (CSB) and can optionally send an interrupt, which notifies the user that the job has completed. NX also updates the accelerator processed byte count (XPBC) in the send window indicating the number of bytes that were processed on behalf of the user.

As shown in *Figure 12-1* on page 200, many different send windows can point to the same receive window. This occurs when many different processes are using and sharing the same NX CT. Each process writes a FIFO entry onto the NX queue, independently, with no ordering implied nor maintained between different processes/different send windows.

12.3 Core-Core Wakeup Via ASB_Notify

A copy-paste pair can be used to initiate core-core wake up with an ASB_Notify command. This can be used to wake up a core from the **wait** instruction. This feature is only supported on DD2.0 hardware.

An example implementation follows:

- Define a receive window for each core that it is desired to wake up and initialize its Local Notify Process ID Register, Local Notify Logical Partition ID Register, and Local Notify Thread ID Register to point to the core/thread that should be notified to wake up.
- Define a send window that points to this receive window.
- Initialize the send and receive windows to not use credits by setting bits [3:4] of the Window Control Register to zero. This step is not required. However, if this step is not done, credits must be returned by some mechanism or future paste operations will be retried indefinitely after credits are exhausted.
- Set the "Disable FIFO Writes" bit in the local DMA cache and FIFO Control Register of the receive window. This step is not required but reduces the SMP interconnect traffic and speeds up notification.
- Initialize the receive window to do the ASB_Notify (instead of an interrupt) notification.
- Initiate a copy-paste pair to the send window to cause an ASB_Notify command to be sent to the core.

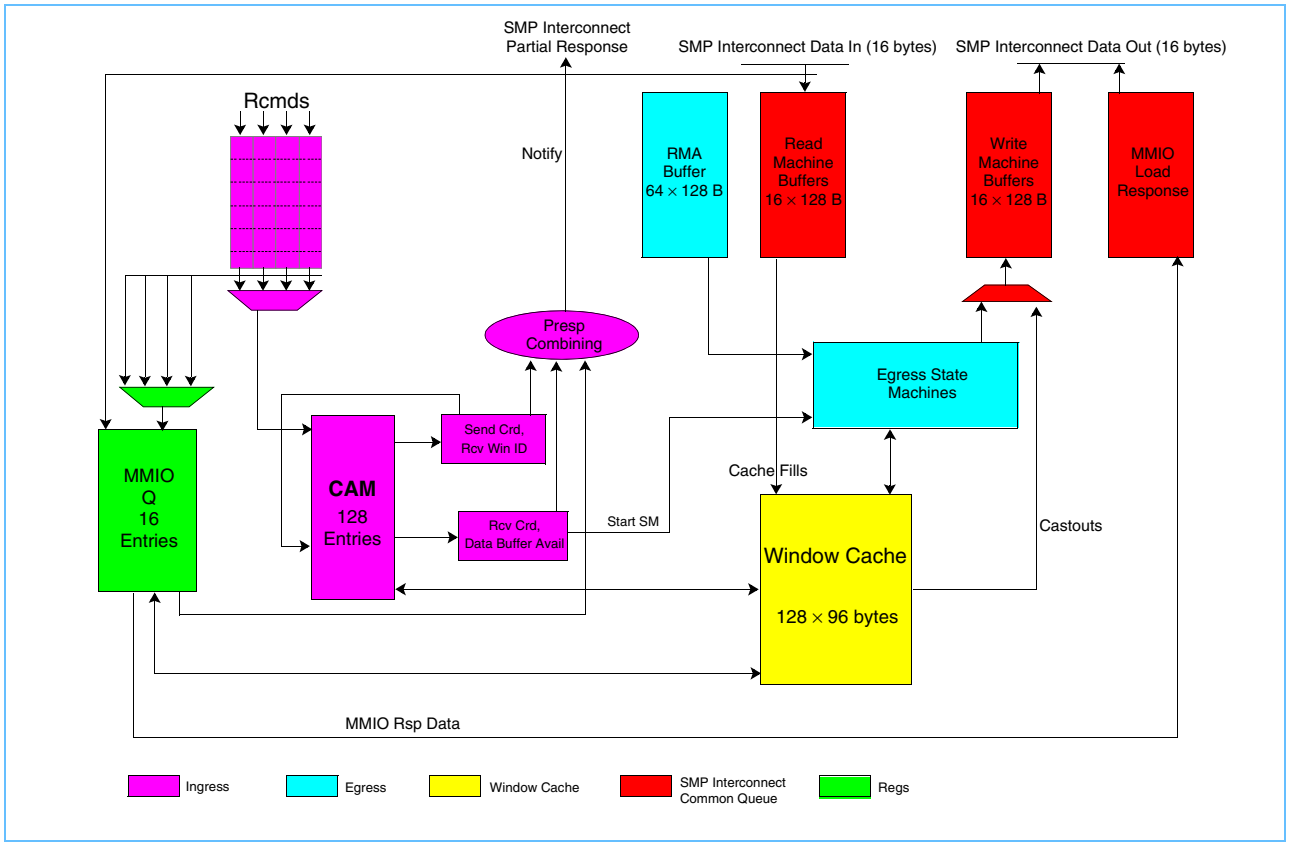
12.4 Features

The following features are implemented in support of NX:

- 64 KiB window support per VAS. For a large SMP with 16 VAS in the system, this gives 1 MiB window accessibility.
- Data stamping of send and receive window information into the CRB.
- Send window credits.
- Receive window credits.
- N $\rightarrow$ 1 support. Multiple sending windows are allowed to point to one receive window.
- NX utilization reporting (XPBC support).
- ASB_Notify notification.
- Ability to pin send or receive window contexts in the cache to avoid casting out frequently-used windows.
- Caching of 128 unique windows contexts.
- Core-core wakeup via ASB_Notify

Figure 12-2 shows a high-level block diagram of the VAS unit.

Figure 12-2. VAS Block Diagram



12.4.1 Ingress

Ingress snoops the SMP reflected command bus for RMA_write commands. RMA_write commands are only accepted if they arrive on reflected command bus 0 (address bits 55:56 = '00'). When Ingress detects that a command is for this VAS, it is sent to the Ingress pipe logic. The pipe logic verifies that both the send and receive windows are cached, performs credit checking/updating, and confirms buffer availability before sending the command to the Egress logic.

The Ingress contains logic for combining the partial responses from the MMIO and RMA_write snooping logic, and driving this partial response onto the SMP interconnect. Ingress also monitors the combined responses on the SMP interconnect to see the resolved response for RMA_write commands it has snooped and aborts RMA_write commands it had accepted if the combined response is not acceptable.

Ingress implements a logical CAM to manage the 128 window contexts that can be cached. Ingress checks the CAM to ensure both the send and receive windows are cached before accepting an incoming RMA_write command. If either window is not cached, Ingress initiates a cache fill and, if necessary, a castout operation to bring the required window contexts into window cache. Ingress implements a pseudo-least recently used (LRU) algorithm to determine which entry to replace on a castout.

VAS has the ability to pin entries in the CAM/Window Cache. Pinning an entry prevents it from being castout. It is expected that the NX receive windows will be pinned to avoid a cache miss penalty when using the NX accelerators.

12.4.2 Egress

Egress receives RMA_write commands from Ingress. Egress has 64 data buffers for holding RMA_write payload (CRB) data. Once a command is received from Ingress and it has received a good combined response as well as its payload data, the RMA_write command arbitrates for one of 16 Egress unload state machines.

The unload state machines request window context information from the window cache unit. Egress uses information in the window context to process the RMA_write command. In particular, Egress determines the receive FIFO write address from the base address register in the context, plus an offset indicating the current offset into the FIFO. Egress reads many other control fields (receive FIFO size, notification controls, and so on) which manage the operations that are performed as part of the RMA processing.

After reading the window context and determining the receive FIFO address, Egress issues a store to the SMP interconnect common queue logic to store the CRB payload to the receive FIFO. VAS then issues an ASB_Notify, using the LPID:PID:TID from the receive window context. This ASB_Notify informs NX that a new operation has been written to the receive FIFO.

After the SMP interconnect common queue completes the FIFO write and ASB_Notify, the Egress unload state machine frees up the data buffer and returns to idle, allowing it to begin processing another RMA_write command.

12.4.3 Window Cache

VAS implements a window context cache for holding window contexts that are frequently accessed. When VAS gets a hit in its window cache, it can avoid fetching a window context from memory every time an RMA_write command is accepted by VAS. VAS is able to hold a total of 128 window contexts, each of which is 96 bytes in length.

When a VAS subunit (Egress or MMIO) needs the window context, the window context cache provides the information. If the window context is not resident in the cache, it is fetched from main memory, casting out an entry to make room in the cache if necessary.

12.4.4 MMIO Registers

The MMIO unit handles MMIO loads and stores to context registers. To set up a send or receive window context, software does a series of a few dozen MMIO stores to the window context.

To reduce the amount of memory that is consumed when a window context must be stored back to memory for a castout, VAS does not store reserved or unused bits and packs the register data into a condensed format. For example, while the FIFO base address register is initialized as an 8-byte MMIO store, only 48 of the 64 bits are stored in the window context cache. The MMIO unit is responsible for understanding the packed format and updating the appropriate portion of the window context when an MMIO store is received.

The MMIO unit has a 16-deep queue for holding MMIO commands. A command reads one row of the window cache array to obtain the respective register's data. For an MMIO load, the appropriate bits of the packed data are selected and returned to the SMP interconnect as MMIO load payload data. For an MMIO store, the received store data is merged into the packed data, leaving adjacent register fields unaltered.

12.4.5 SMP Interconnect Common Queue

The VAS SMP interconnect common queue (CQ) logic contains 16 read machines and 16 write/ASB_Notify machines. These machines track outbound commands through their entire life cycle on the SMP interconnect. The 16 multipurpose write machines are shared by Egress and window cache, but only Egress issues ASB_Notify commands. The read machines are only used by the window cache for performing cache fill operations.

The CQ logic also handles inbound data and outbound data. An inbound data controller determines whether data belongs to a cache fill operation, an RMA_write operation, or to an MMIO store operation and routes the data accordingly. The outbound data controller takes requests from the write machines, as well as data returns for MMIO load commands.

12.5 Reliability and Serviceability (RAS) Features

VAS implements ECC on all arrays. The ECC algorithm is a single-bit error correct detect, double-bit error detect algorithm.

13. NVLink Processing Unit

This section describes the NVLink processing unit (NPU) and the corresponding 25G Link interface.

13.1 Overview

The POWER9 chip includes the high-speed 25G Link, which creates an interface between chips that provides both cache coherence and very high data bandwidth. For example, this structure can be used to connect a CPU chip to a cluster of GPU chips. The CPU and GPU cluster can coherently read and write to each other's memory. The GPU can use non-caching (DMA) reads and writes for high-bandwidth data moves between GPU memory and CPU memory. The 25G Link interconnect is included on the POWER9 chip and requires PHYs, datalink-layer logic, and transaction-layer logic. The PHYs are the physical connection to the 25G Link. The datalink layer provides link training, CRC generation and checking, and the replay of failed packets. The transaction layer executes the cache coherent and data movement commands on the POWER9 chip.

The 25G Link on the POWER9 chip supports the NVLink protocol. The NPU provides the transaction layer functionality for this protocol. The NPU accepts commands from the NVLink datalink logic and converts them into sequences of on-chip SMP interconnect commands. It then generates responses based on the results of the on-chip SMP interconnect commands. The responses are sent back to the 25G Link through the datalink logic. The supported commands include reads, writes, probes, and flushes. The NPU can send reads and writes. It also sends upgrade and downgrade commands to the GPU cluster over the 25G Link as a result of operations seen on the processor bus reflected command buses. In addition, transaction-layer command, response, and data credits are passed in both directions over the link.

The 25G Link can be made up of one or more units, which are referred to as bricks. Each brick provides a separate stream of commands. All ordering requirements are enforced independently for each brick. The POWER9 NPU supports up to six bricks. The six bricks can be connected to one or more external chips. If more than one external chip is attached to a POWER9 chip over multiple 25G Links, the external chips can be connected to each other using separate NVLink interconnects. To the POWER9 chip, they can appear to represent a single external memory region.

The NPU has three on-chip SMP interconnect unit interfaces. Each of the on-chip SMP interconnect interfaces supports two 25G Link bricks. The NPU implements a static connection of bricks to the on-chip SMP interconnect interfaces. For example, bricks 0 - 1 are connected to on-chip SMP interconnect interface 0, bricks 2 - 3 are connected to on-chip SMP interconnect interface 1, and bricks 4 - 5 are connected to on-chip SMP interconnect interface 2. If bricks 2 - 3 are not connected to an external chip or network of chips, on-chip SMP interconnect interface 1 goes (mostly) unused.

13.2 Features

An NPU feature summary follows:

- 25G Link brick bandwidths are as follows:
  - Peak read bandwidth per brick: 25 GBps
  - Peak write bandwidth per brick: 25 GBps
- The NPU supports up to six 25G Link bricks.
- On-chip SMP interconnect interface bandwidths are as follows:
  - Peak read bandwidth per interface: 64 GBps
  - Peak write bandwidth per interface: 64 GBps
- The NPU connects to three on-chip SMP interconnect interfaces.
- Effective bandwidths including command and response overhead are as follows:
  - Read bandwidth per brick: 23.5 GBps
  - Write bandwidth per brick: 21.1 GBps
  - Total read bandwidth for the NPU: 141 GBps
  - Total write bandwidth for the NPU: 127 GBps
- Transfer sizes supported are as follows:
  - Minimum transfer size: 1 byte
  - Maximum transfer size: 256 bytes
- The NPU performs coherent operations on 128-byte cache-line boundaries.
- Address translation services sizes are as follows:
  - Bus device functions supported: 15
  - Process address space IDs supported: 256
  - Logical partition ID/process ID pairs supported: 256
  - Translation contexts supported: 256
- Translation control element table sizes are as follows:
  - Translation validation table: 16 entries
  - Translation control element table: 4K - 1T entries

13.3 Interfaces

The interfaces to and from the NPU include the on-chip SMP interconnect command and data ports, NVLink transaction layer (NTL) receive and transmit ports, and the NVLink datalink layer (NDL)/PHY Private Register interface.

13.3.1 On-Chip SMP Interconnect Ports

The NPU attaches to three on-chip SMP interconnect ports. The three ports are independent from each other. Each port includes a command request interface, four snoop interfaces, and data in and out interfaces.

13.3.1.1 Command Request

On each of the three command request interfaces, the NPU must be able to request a command every two cycles. This gives an aggregate command rate for the NPU of 1.5 on-chip SMP interconnect commands every cycle.

13.3.1.2 Command Snoop

The NPU snoops the on-chip SMP interconnect for the following functions:

- Accesses to GPU memory
- MMIO loads and stores to NPU, NDL, and PHY registers

13.3.1.3 Data to On-Chip SMP Interconnect

The three data bus ports from the NPU to the on-chip SMP interconnect are each 32 bytes wide. To sustain the DMA write bandwidth from the 25G Link bricks, the NPU must be able to send cache lines to the on-chip SMP interconnect in back-to-back cycles for relatively long periods of time.

13.3.1.4 Data from On-Chip SMP Interconnect

The three data bus ports to the NPU are each 32 bytes wide. The NPU must be able to receive data for any octword of any outstanding read in any cycle.

13.3.2 NTL Interfaces

13.3.2.1 NTL Receive Interface

The unit of transfer across the NTL receive interface is a 16-byte flow control digit (FLIT). There are four parity bits associated with the FLIT. Each parity bit covers 32 bits of the FLIT.

13.3.2.2 NTL Transmit Interface

The unit of transfer across the NTL transmit interface is a 16-byte FLIT. There are four parity bits associated with the FLIT. Each parity bit covers 32 bits of the FLIT.

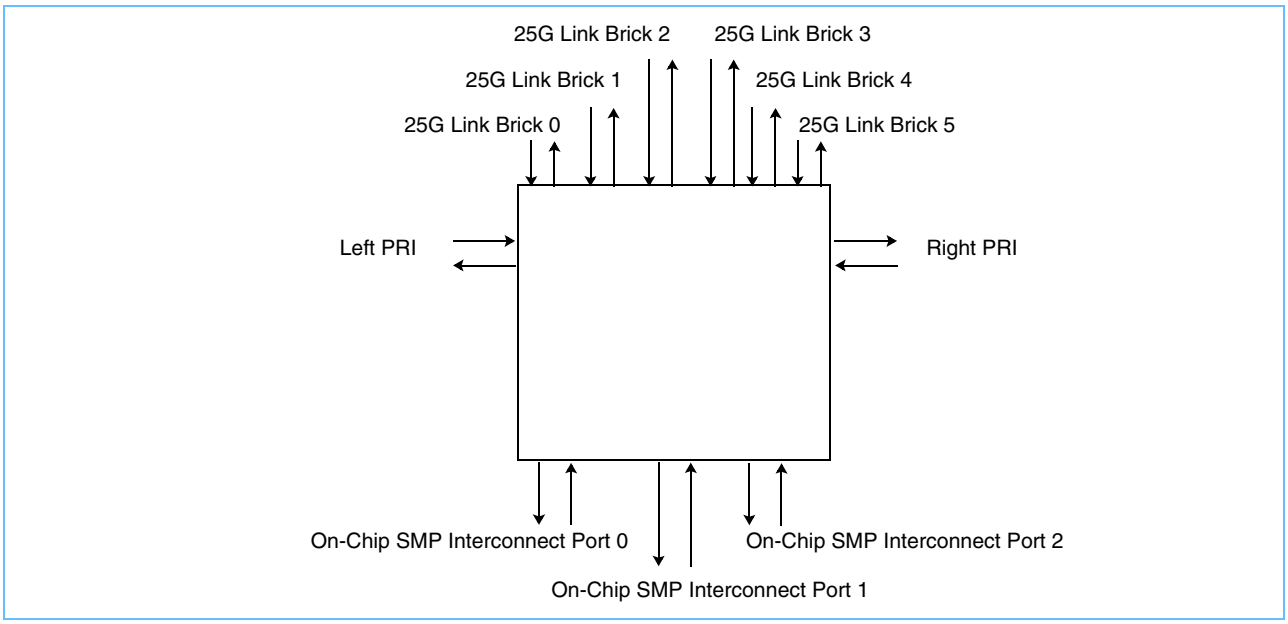
13.3.2.3 NDL/PHY Private Register Interface

The NPU provides MMIO register access to NDL and PHY registers using the private register interface (PRI). There are separate PRI interfaces for the NDLs located on the left and right sides of the POWER9 chip.

13.3.3 Interface Diagram

Figure 13-1 on page 210 shows the interfaces attached to the NPU unit.

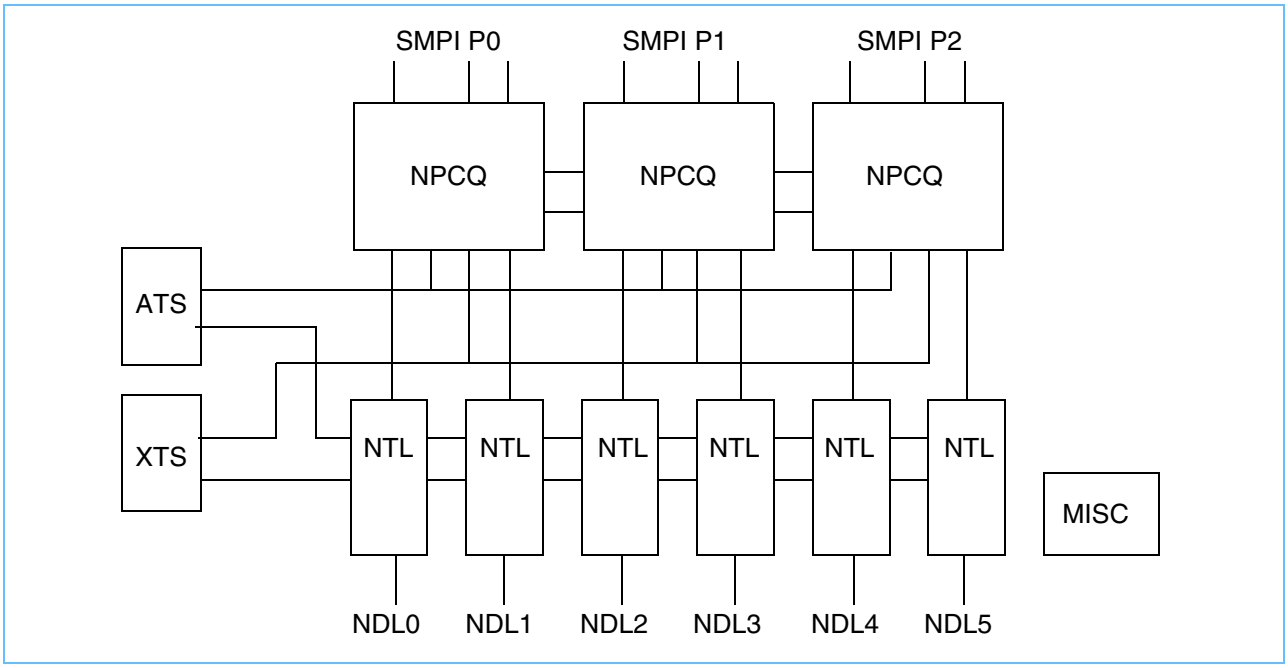
Figure 13-1. NPU Interface Diagram



13.4 Block Diagram

Figure 13-2 shows the major blocks within the NPU. *Section 13.4.1* on page 211 through *Section 13.4.5* on page 212 give a brief description of each of the blocks in the diagram.

Figure 13-2. NPU Block Diagram



13.4.1 NPU Common Queue

The NPU common queue (NPCQ) performs the following functions:

- Provides a command and data interface to the on-chip SMP interconnect (SMPI).
- Provides state machines for executing on-chip SMP interconnect and NVLink commands coherently between the POWER9 and GPU cluster.
- Performs buffering for data going to or coming from the SMPI.

There are three copies of this block in the NPU.

13.4.2 NVLink Transaction Layer

The NVLink transaction layer (NTL) block contains the receive and transmit interfaces between the NPU and the NDL blocks. The NTL performs the following functions:

- Validates commands and data from the 25G Link.
- Buffers commands and data from the 25G Link.
- Formats commands and responses going to the 25G Link.
- Manages transaction layer credits.
- Controls the Private Register Interface (PRI) to the 25G Link NDL and PHY functions

There are six copies of this block in the NPU.

13.4.3 Extended Translation Services

The extended translation services (XTS) block is used to support the 25G Link address translation services operations. This block accepts address translation requests from the GPU, looks up the necessary translation context, and creates translation requests for the POWER9 nest memory management unit (NMMU). Translation contexts are kept in a table in the XTS unit. When the NMMU responds to a translation request, the XTS block creates an address translation response that is sent to the GPU over the 25G Link. The XTS block also generates address shoot-down requests based on the snooping of translation lookaside-buffer invalidate (TLBI) operations on the on-chip SMP interconnect.

There is one copy of this block in the NPU.

13.4.4 Address Translation Services

The address translation services (ATS) block provides address relocation and validation when untranslated addresses are used in commands from the GPU. Relocation and validation are done using the translation control element (TCE) mechanism. The ATS block contains a cache of TCEs and performs a table search operation when a cache miss occurs. The ATS block can generate interrupts when certain error conditions occur.

There is one copy of this block in the NPU.

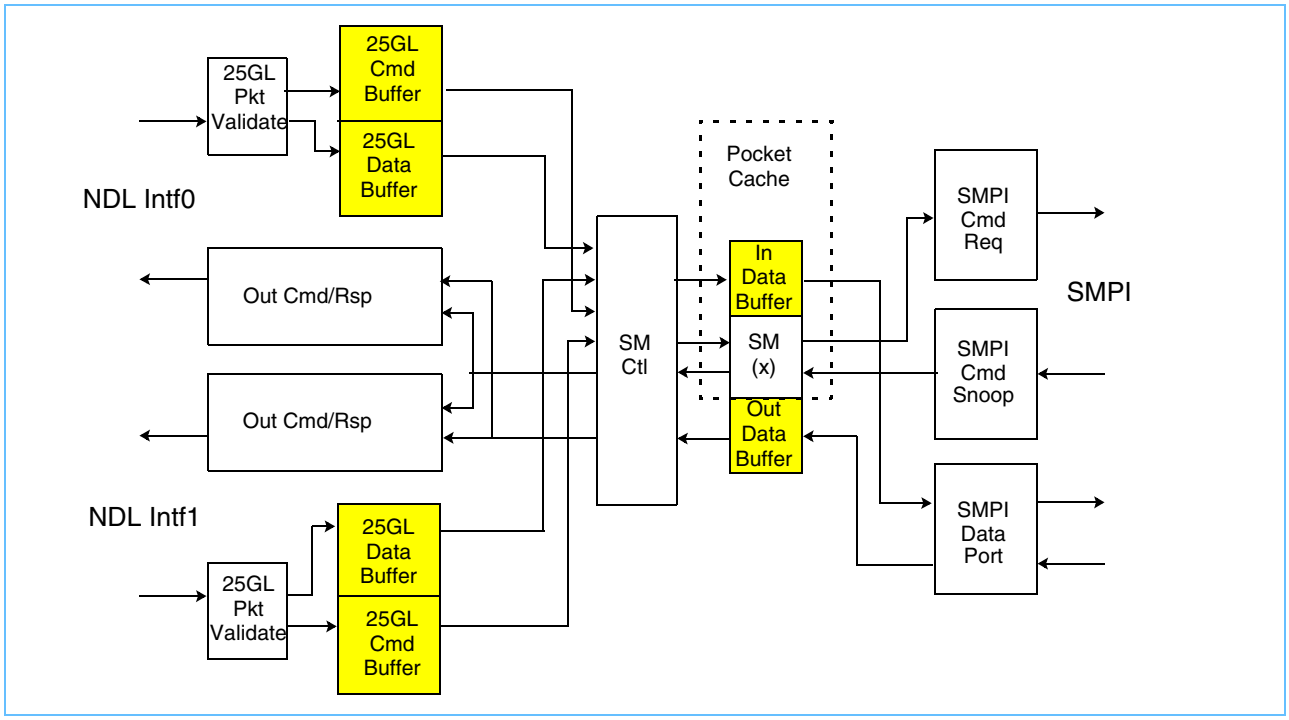
13.4.5 Miscellaneous

The miscellaneous (MISC) block includes the common functions for the NPU. These include the register access, array built-in self-test controller, and error gathering and reporting. There is one copy of this block in the NPU.

13.5 Logical Command/Data Flow

Figure 13-3 shows the logical command and data flow for the NPU. The diagram shows two NDL interfaces and one on-chip SMP interconnect port. This represents one third of the complete NPU flow.

Figure 13-3. NPU Command/Data Flow



In *Figure 13-3*, the blocks are represented as follows:

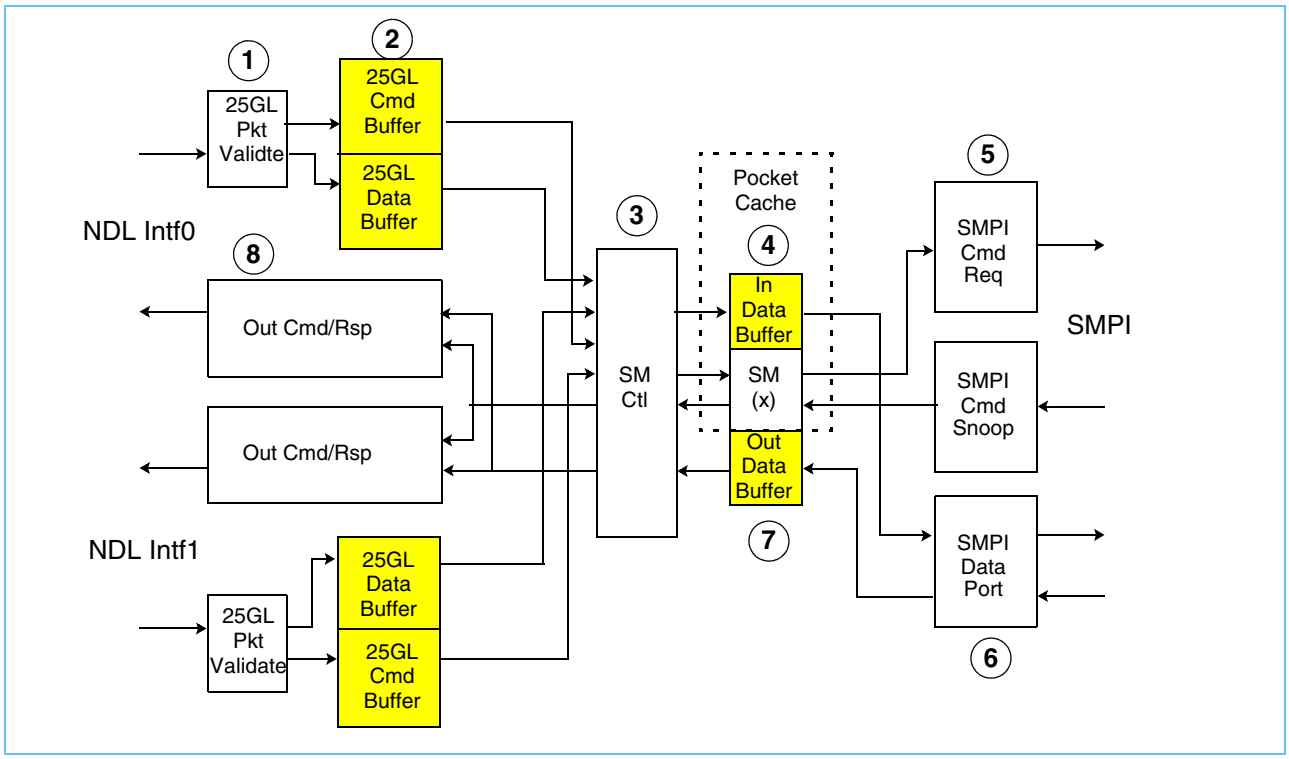
- The NTL is represented by: 25GL Pkt Validate, 25GL Cmd Buf, 25GL Data Buf, Out Cmd/Rsp
- The NPCQ is represented by: SM Ctl, SM, In Data Buf, Out Data Buf, SMPI Cmd Req, SMPI Cmd Snoop, SMPI Data Port

Also note that the Pocket Cache is shown as dashed lines because it is a logical entity that is actually made up of all of the state machines (SM) and their inbound data buffers (In Data Buf). If a cache line is valid in the pocket cache, a state machine and data buffer are being used to hold and check for snoop hits on the cache line.

13.5.1 Inbound Command/Data Flow

Figure 13-4 shows the flow for commands and data coming from the NVLink protocol over the 25G Link to the POWER9 chip.

Figure 13-4. NPU Inbound Command/Data Flow



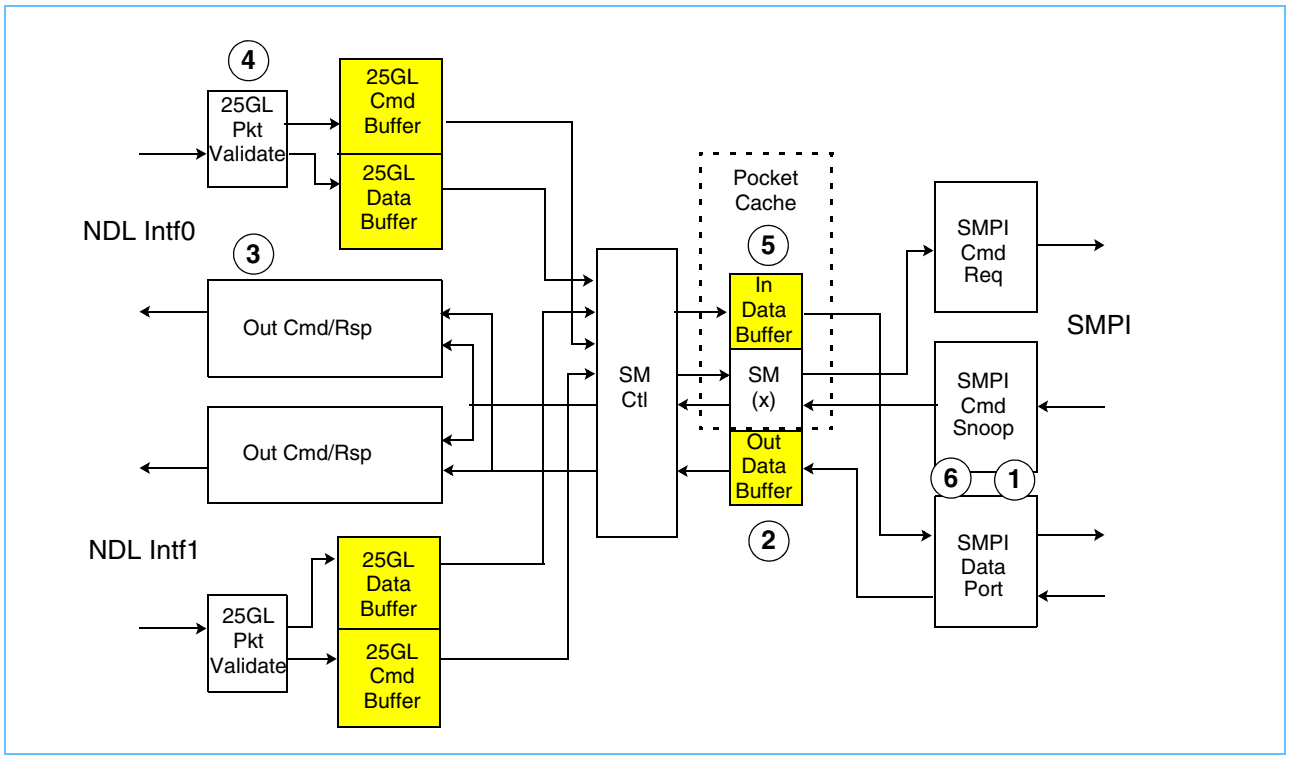
The flow in *Figure 13-4* is as follows:

1. A packet that arrives on the NDL interface must be validated. This includes checking for the correct parity and the CRC status, as well as checking for proper syntax.
2. Commands that pass the validation check are stored in the 25GL command buffer. For write commands, the data is stored in the 25GL data buffer. NTL credits are maintained based on available space in these buffers.
3. Commands in the 25GL command buffer are arbitrated against commands in the 25GL command buffer servicing the other NTL in this pair, as well as against commands coming from the on-chip SMP interconnect.
4. When a command in the 25GL command buffer wins arbitration, it is assigned to a state machine. For writes, the data is moved from the 25GL data buffer to the In data buffer.
5. The state machine controls the execution of the command. Any required on-chip SMP interconnect commands are sent to the SMPI command request port.
6. On-chip SMP interconnect data transfers related to requested SMPI commands are handled through the SMPI data ports.
7. Data for read commands is stored in the Out data buffer.
8. When the state machine indicates that it is time to send a response for the command, the Out command/response block generates the response and includes any required data from the Out data buffer.

13.5.2 Outbound Command/Data Flow

Figure 13-5 shows the command and data flow for commands coming from the on-chip SMP interconnect and going to the GPU.

Figure 13-5. NPU Outbound Command/Data Flow



The flow in *Figure 13-5* is as follows:

1. A command that accesses GPU memory space is seen by the SMPI command snoop logic. For write or push operations, data is received on the SMPI data port.
2. A state machine is assigned to process the command. Any data that is received is stored in the Out data buffer.
3. The state machine signals that a command must be sent to the GPU. The Out command/response function creates the command and includes any required data from the Out data buffer.
4. When the response for the command is received on the NDL interface, it must be validated, which includes checking parity and the CRC status as well as the response syntax.
5. If the response is validated, it is passed to the state machine that is processing the command. Any data associated with the response is placed in the In data buffer. When there is data returned as part of the response, it is held in the pocket cache until it is requested by another unit on the on-chip SMP interconnect.
6. Later, a command is seen by the SMPI command snoop logic that requires the data in the pocket cache. The command is acknowledged on the on-chip SMP interconnect, and the data is provided from the Out data buffer.

13.6 POWER9/GPU Transaction Examples

The following sections contain examples of transactions between the POWER9 and GPU chips using the 25G Link and the on-chip SMP interconnect.

13.6.1 GPU Read from POWER9 Memory

Table 13-1 shows an example of a 128-byte read command sent over a 25G Link brick.

Table 13-1. Example of 128-Byte Read Command

Step	GPU Drives 25GL	25G Link	NPU Drives 25GL	NPU Drives SMPI	SMPI	From SMPI	Comment
1	→	NON_CACHING_ READ		→	NON_CACHING_ READ		
2					Memory_Ack	←	Read acknowledged by memory.
3					Data	←	Memory sends data for the read.
4		READ_RESP DATA	←				NPU sends read response to GPU.

Note: 25GL = 25G Link; Ack = Acknowledge; Resp = Response

The steps for *Table 13-1* are explained as follows:

Step	Description
1	GPU sends a NON_CACHING_READ command to the POWER9 chip over the 25G Link. The NPU sends a NON_CACHING_READ on the SMPI.
2	The POWER9 memory controller acknowledges the command.
3	The POWER9 memory controller sends data for the read to the NPU over the SMPI data bus.
4	The NPU sends a response to the GPU for the read command.

13.6.2 GPU Posted Writes to the POWER9 Memory

Table 13-2 shows an example of a series of posted 128-byte write commands sent over the same 25G Link brick. Posted writes do not receive responses; therefore, they do not require transaction done commands to complete them.

Table 13-2. Example of Series of Posted 128-Byte Write Commands

Step	GPU Drives 25GL	25G Link	NPU Drives 25GL	NPU Drives SMPI	SMPI	From SMPI	Comment
1	→	WRITE_0 DATA		→	DMA_INJECT_0		Stream of noncaching writes.
2	→	WRITE_1 DATA		→	DMA_INJECT_1		
3	→	WRITE_2 DATA		→	DMA_INJECT_2		
4	→	FLUSH					GPU checks for write stream globally visible.
5					Cache_Ack_1	←	Writes can finish out of order. An inject hits the cache.
6					Memory_Ack_0	←	Inject handled by POWER9 memory.
7					Memory_Ack_2	←	
8		FLUSH.RESP	←	→	data_1		Data sent to memory in order of responses.
9				→	data_0		
10				→	data_2		
11	→	WRITE_3 DATA		→	DMA_INJECT_3		Write_3 becomes globally visible after writes 0 - 2.
12					Memory_Ack_3	←	
13				→	data_3		

Note: 25GL = 25G Link; Ack = Acknowledge; Resp = Response

The steps for *Table 13-2* are explained as follows:

1 - 3 The GPU sends a stream of three posted, noncaching writes on the same 25G Link brick. The writes are sent to the SMPI as DMA_INJECTs by the NPU.

4 The GPU sends a FLUSH command to determine when the previous writes are complete.

5 - 7 The SMPI responses arrive out-of-order. All responses indicate that the writes are globally visible.

8 The NPU can now respond to the FLUSH command. Also shown in this cycle, the NPU sends the data for the second write onto the SMPI.

9 - 10 The NPU sends the data onto the SMPI for the remaining two writes.

11 The GPU sends a fourth write that had an ordering dependency on the first three writes. The NPU converts the posted, noncaching write to a DMA_INJECT command on the SMPI.

12 The SMPI response for the fourth write indicates that the inject command is complete.

13 The NPU sends the data onto the SMPI for the fourth write.

13.6.3 POWER9 Caching Read from GPU Memory

Table 13-3 shows an example of a 128-byte read command that was sent onto the SMPI by a POWER9 cache and that references GPU memory.

Table 13-3. Example of a POWER9 Caching Read from GPU

Step	GPU Drives 25GL	25G Link	NPU Drives 25GL	NPU Drives SMPI	SMPI	From SMPI	Comment
1					CACHING_READ	←	Cache controller sends a caching read command.
2				→	NPU_Reject		NPU must reject the command and get ownership and data from the GPU.
3		CACHING_READ	←				NPU sends a caching read command to the GPU.
4	→	READ_RESP OWNERSHIP DATA					GPU sends a read response command to the NPU. The NPU stores the read data in its pocket cache.
5		READ_TDONE	←				The NPU completes the read with a transaction done command.
6					CACHING_READ	←	The cache controller resends its caching read command.
7				→	NPU_Ack		A read hits the NPU pocket cache.
8				→	Data		The NPU sends the data to the requesting cache over the SMPI.

Note: 25GL = 25G Link; Ack = Acknowledge; Resp = Response; Tdone = Transaction done

The steps for *Table 13-3* are explained as follows:

1 A cache controller on the POWER9 chip, while handling a cache miss, sends a CACHING_READ command over the SMPI. The read command references GPU memory; therefore, the NPU snoops the command.

2 The NPU cannot acknowledge the read command until it first receives ownership and data from the GPU. Therefore, the NPU must reject the read command from the SMPI. This causes the cache controller to resend the command later.

3 The NPU sends a CACHING_READ command over the 25G Link to the GPU.

4 The GPU sends a read response, which provides data and ownership for the cache line. The NPU stores the cache-line data into its pocket cache.

5 The NPU completes the 25G Link read command with a transaction done.

6 The cache controller resends the CACHING_READ command.

7 The CACHING_READ command hits the NPU pocket cache. Therefore, the NPU can acknowledge the command.

8 The NPU sends the data to the cache controller over the SMPI.

13.6.4 POWER9 Cache Releasing a Cache Line from GPU Memory

Table 13-4 shows an example of a cache controller on the POWER9 releasing ownership of a cache line that came from GPU memory. In this example, the data has been modified; therefore, it must be given back to the GPU. If the data has not been modified, the POWER9 cache controller can drop the cache line without informing the GPU.

Table 13-4. Example of Cache Controller on POWER9 Chip Releasing Cache Line from GPU

Step	GPU Drives 25GL	25G Link	NPU Drives 25GL	NPU Drives SMPI	SMPI	From SMPI	Comment
1					PUSH	←	Cache controller sends a push command to release a modified cache line.
2				→	NPU_Ack		The push command references GPU memory; therefore, the NPU acknowledges it.
3					Data	←	Cache controller sends data for the push command.
4		DOWNGRADE	←				NPU sends a downgrade command to the GPU.
5	→	DOWNGRADE_ RESP					GPU sends a response for the downgrade command.
6		DOWNGRADE_ TDONE DATA	←				NPU completes the downgrade command with a transaction done indication.

Note: 25GL = 25G Link; Ack = Acknowledge; Resp = Response; Tdone = Transaction done

The steps for *Table 13-4* are explained as follows:

1 A cache controller on the POWER9 chip releases ownership of a cache line by sending a PUSH command over the SMPI. The PUSH command references GPU memory; therefore, the NPU snoops the command.

2 The NPU acknowledges the PUSH command.

3 When the cache controller sees the NPU acknowledgment, it sends the data for the PUSH command to the NPU using the SMPI.

4 The NPU sends the PUSH command to the GPU using a DOWNGRADE command on the 25G Link.

5 The GPU responds to the DOWNGRADE command.

6 The NPU completes the DOWNGRADE command with a transaction done. On the 25G Link, the data for a DOWNGRADE command is sent with the transaction done indication.

13.6.5 GPU Reclaiming a Cache Line from GPU Memory

Table 13-5 shows an example of the GPU requesting the return of a cache line from GPU memory that is held in a cache on the POWER9 chip. In this example, the data has been modified; therefore, it must be given back to the GPU. The cache controller releases the data by sending a PUSH command on the SMPI, which the NPU forwards to the GPU as a DOWNGRADE. If the data had not been modified, no PUSH occurs and the NPU responds to the PROBE command without sending any data.

Table 13-5. Example of GPU Reclaiming a Cache Line from GPU Memory

Step	GPU Drives 25GL	25G Link	NPU Drives 25GL	NPU Drives SMPI	SMPI	From SMPI	Comment
1	→	PROBE		→	CACHE_FLUSH		GPU reclaims a cache line using a probe command. The NPU forwards the probe command to the SMPI as a cache_flush command.
2					CACHE_REJECT		The cache controller rejects the cache_-flush until it can push the cache line out.
3					PUSH	←	Cache controller pushes the cache line.
4				→	NPU_Ack		NPU acknowledges the push command.
5					Data	←	Cache controller sends data for the push command.
6		DOWNGRADE	←				NPU sends a downgrade command to the GPU.
7	→	DOWNGRADE_ RESP					GPU sends a response for the downgrade command.
8		DOWNGRADE_ TDONE DATA	←				NPU completes the downgrade command with a transaction done.
9				→	CACHE_FLUSH		NPU re-sends the cache flush command.
10					No_Hits	←	There are no hits in any cache.
11		PROBE_RESP	←				NPU responds to the probe command with no data.

Note: 25GL = 25G Link; Ack = Acknowledge; Resp = Response; Tdone = Transaction done

The flow for *Table 13-5* is explained as follows:

1 The GPU sends a PROBE command on the 25G Link to regain ownership of a cache-line that is held in a POWER9 cache. The NPU sends the command onto the SMPI as a CACHE_FLUSH command.

2 A cache controller for a cache that has the cache-line modified rejects the CACHE_FLUSH command until it can push the cache line out.

3 The cache controller releases ownership of the cache line by sending a PUSH command over the SMPI. The PUSH command references GPU memory; therefore, the NPU snoops the command.

4 The NPU acknowledges the PUSH command.

5 When the cache controller sees the NPU acknowledgment, it sends the data for the PUSH command to the NPU using the SMPI.

6 The NPU sends the PUSH command to the GPU using a DOWNGRADE command on the 25G Link.

7 The GPU responds to the DOWNGRADE command.

8 The NPU completes the DOWNGRADE command with a transaction done. On the 25G Link, the data for a DOWNGRADE command is sent with the transaction done indication.

9 The NPU resends the CACHE_FLUSH command onto the SMPI.

10 The NPU receives a response from the SMPI indicating that there are no cache hits on the resent CACHE_FLUSH command.

11 The NPU responds to the GPU's PROBE command and does not send any data. No data is required because the cache line has already been returned with the DOWNGRADE command.

14. OpenCAPI Processing in the POWERAccel Unit

This section describes the OpenCAPI interfaces and the parts of the POWERAccel unit (PAU) that provide the transaction layer functionality for those interfaces.

14.1 Overview

The Open Coherent Accelerator Processor Interface (OpenCAPI) enables an attached functional unit (AFU) to connect to the POWER9 on-chip SMP interconnect bus in a high-speed, cache-coherent manner. For example, this structure can be used to connect a POWER9 chip to an FPGA containing a DMA controller or a data storage function. The DMA controller and data storage function represent the AFUs on the FPGA. The POWER9 chip and the AFU have the ability to coherently read and write memory attached to either chip. The AFU can use noncaching (DMA) reads and writes for high-bandwidth data moves between AFU memory and POWER9 memory. Supporting OpenCAPI on the POWER9 chip requires OpenCAPI-capable PHYs, data-link-layer logic, and transaction-layer logic. The PHYs are the physical connection to the OpenCAPI interconnect. The datalink layer provides link training, CRC generation and checking, and the replay of failed packets. The transaction layer executes the cache-coherent and data-movement commands on the POWER9 chip.

The PAU provides provides the transaction layer functionality for the OpenCAPI links on the POWER9 chip. This functionality includes accepting commands from the OpenCAPI datalink logic, converting them into sequences of on-chip SMP interconnect commands, and then generating responses based on the results of the on-chip SMP interconnect commands. The responses are sent back to the OpenCAPI link through the datalink logic. The supported commands include reads, writes, and interrupts. The PAU sends reads and writes to the AFU over the OpenCAPI link as a result of operations seen on the on-chip SMP interconnect buses. In addition, transaction-layer command response and data credits are passed in both directions over the link.

An OpenCAPI connection to the POWER9 chip is composed of one or more individual links. Each link provides a separate stream of commands. All ordering requirements are enforced independently for each link. The POWER9 PAU supports up to four OpenCAPI links. The four links can be connected to one or more external chips.

The PAU has three on-chip SMP interconnect interfaces. Two of the on-chip SMP interconnect interfaces each support two OpenCAPI links. The third on-chip SMP interface is used for accessing PAU registers and for other maintenance functions. The PAU implements a static connection of links to on-chip SMP interconnect interfaces. For example, links 0 - 1 are connected to on-chip SMP interconnect interface 1, and links 2 - 3 are connected to on-chip SMP interconnect interface 2. If links 2 and 3 are not connected to an external chip or chips, on-chip SMP interconnect interface 2 is (mostly) unused.

14.2 Features

A summary of the features are as follows:

- OpenCAPI link bandwidths are as follows:
  - Peak read bandwidth per link: 25.78 GBps
  - Peak write bandwidth per link: 25.78 GBps
- The PAU supports four OpenCAPI links
- Effective bandwidths including command and response overhead are as follows:
  - Read bandwidth per link: 22.5 GBps
  - Write bandwidth per link: 22.5 GBps
  - Total read bandwidth for the PAU: 90 GBps
  - Total write bandwidth for the PAU: 90 GBps
- Address translation sizes and rates are as follows:
  - Effective-to-real address table (ERAT): 64 entry (16×4)
  - Context cache: 64 entry (16×4)

14.3 Interfaces

The interfaces to and from the PAU include on-chip SMP interconnect command and data ports, as well as the OpenCAPI transaction layer receive and transmit ports.

14.3.1 On-Chip SMP Interconnect Ports

The PAU attaches to three on-chip SMP interconnect ports. The three ramps are independent from each other. Each ramp includes a command request interface, four snoop interfaces, and data in and data out interfaces.

14.3.1.1 Command Request

For each of the three command request interfaces, the PAU must be able to request a command every two cycles. This gives an aggregate command rate for the PAU of 1.5 on-chip SMP interconnect commands every cycle.

14.3.1.2 Command Snoop

The PAU snoops the on-chip SMP interconnect buses for the following functions:

- Access to the AFU memory
- MMIO loads and stores to the AFU configuration space
- MMIO loads and stores to PAU registers

14.3.1.3 Data to On-Chip SMP Interconnect

The three data bus ports from the PAU to the on-chip SMP interconnect are each 32 bytes wide. To sustain DMA write bandwidth from the OpenCAPI links, the PAU must be able to send cache lines to the on-chip SMP interconnect in back-to-back cycles for relatively long periods of time.

14.3.1.4 Data from On-Chip SMP Interconnect

The three data bus ports to the PAU are each 32 bytes wide. The PAU must be able to receive data for any octword of any outstanding read in any cycle.

14.3.2 OpenCAPI Transaction Layer Interfaces

14.3.2.1 OTL Receive Interface

The unit of transfer across the OpenCAPI transaction layer (OTL) receive interface is a 64-byte flow control digit (FLIT).

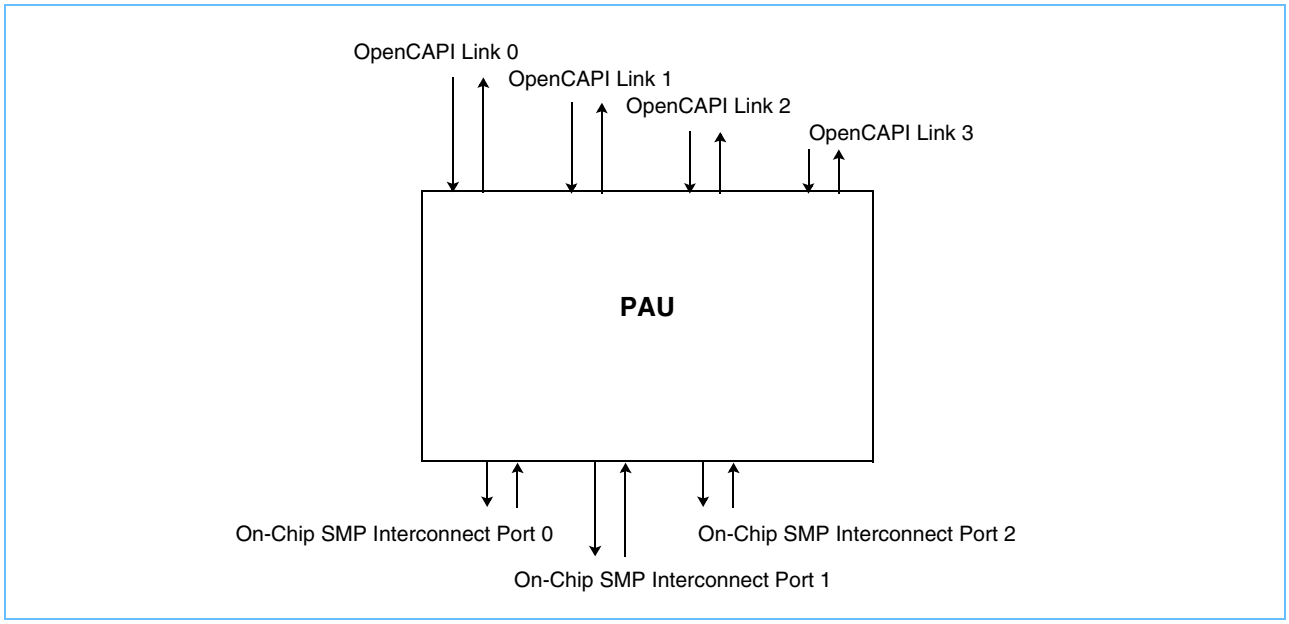
14.3.2.2 OTL Transmit Interface

The unit of transfer across the OpenCAPI transaction layer transmit interface is a 64-byte FLIT.

14.3.3 Interface Diagram

Figure 14-1 shows the interfaces attached to the PAU unit.

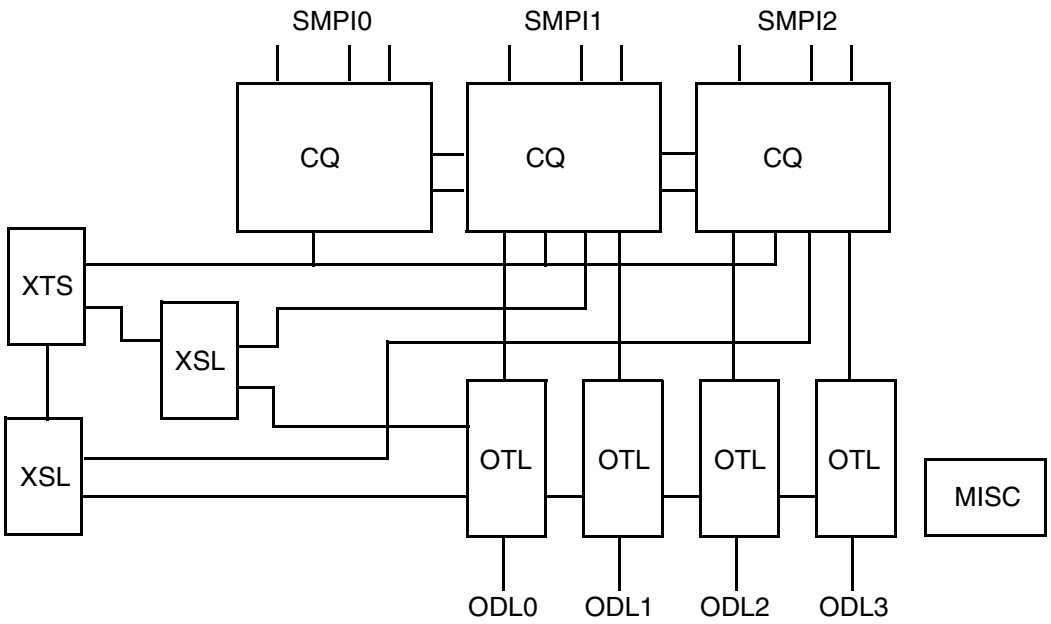
Figure 14-1. PAU Interface Diagram



14.4 Block Diagram

Figure 14-2 shows the major blocks within the PAU. The following sections give a brief description of each of the blocks in the diagram.

Figure 14-2. PAU Block Diagram



14.4.1 PAU Common Queue

The PAU common queue (CQ) performs the following functions:

- Provides a command and data interface to the on-chip SMP interconnect (SMPI).
- Provides state machines for executing on-chip SMP interconnect and OpenCAPI commands coherently between the POWER9 and AFU.
- Performs buffering for data going to or coming from the SMPI.

There are three copies of this block in the PAU.

14.4.2 OpenCAPI Transaction Layer

The OpenCAPI transaction layer (OTL) block contains the receive and transmit interfaces between the PAU and the OpenCAPI datalink layer (ODL) blocks. The OTL performs the following functions:

- Validates commands and data from the OpenCAPI link.
- Buffers commands and data from the OpenCAPI link.
- Formats commands and responses going to the OpenCAPI link.
- Manages transaction layer credits.

There are four copies of this block in the PAU.

14.4.3 Extended Translation Services

The extended translation services (XTS) block is used to support the OpenCAPI address translation operations. This block accepts address translation checkout requests from the address translation (XSL) blocks and creates translation requests for the POWER9 nest memory management unit (NMMU). When the NMMU responds to a translation request, the XTS block forwards it to the requesting XSL. The XTS block also snoops translation lookaside-buffer invalidate (TLBI) operations on the on-chip SMP interconnect and forwards them to the XSL blocks.

There is one copy of this block in the PAU.

14.4.4 Address Translation

The address translation (XSL) block provides address translation for the effective address that is sent with read, write, interrupt, and wake commands from the AFU. The XSL block includes a cache of the address translation contexts as well as an effective-to-real address table (ERAT). A miss on the context cache results in a read request being sent to the CQ block. A miss on the ERAT results in a checkout request being sent to the XTS block.

There are two copies of this block in the PAU.

14.4.5 Miscellaneous

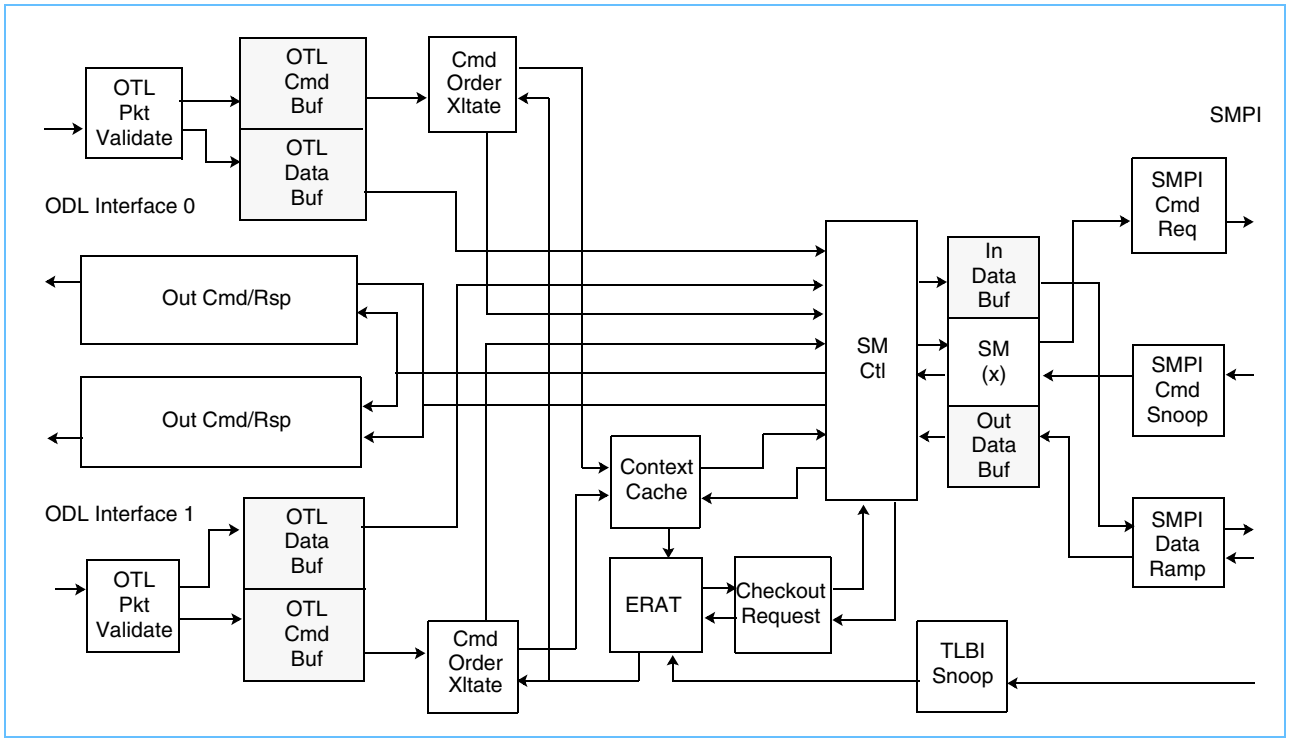
The miscellaneous (MISC) block includes the common functions for the PAU. These include the register access, array built-in-self-test, and error gathering and reporting.

There is one copy of this block in the PAU.

14.5 Logical Command/Data Flow

Figure 14-3 on page 228 shows the logical command and data flow for the PAU. The diagram shows two ODL interfaces and one on-chip SMP interconnect port. This represents half of the complete PAU flow.

Figure 14-3. PAU Command Data Flow



The OTL is represented by: OTL Pkt Validate, OTL Cmd Buf, OTL Data Buf, Cmd Order Xlate, Out Cmd/Rsp.

The XSL is represented by: Context Cache and the ERAT.

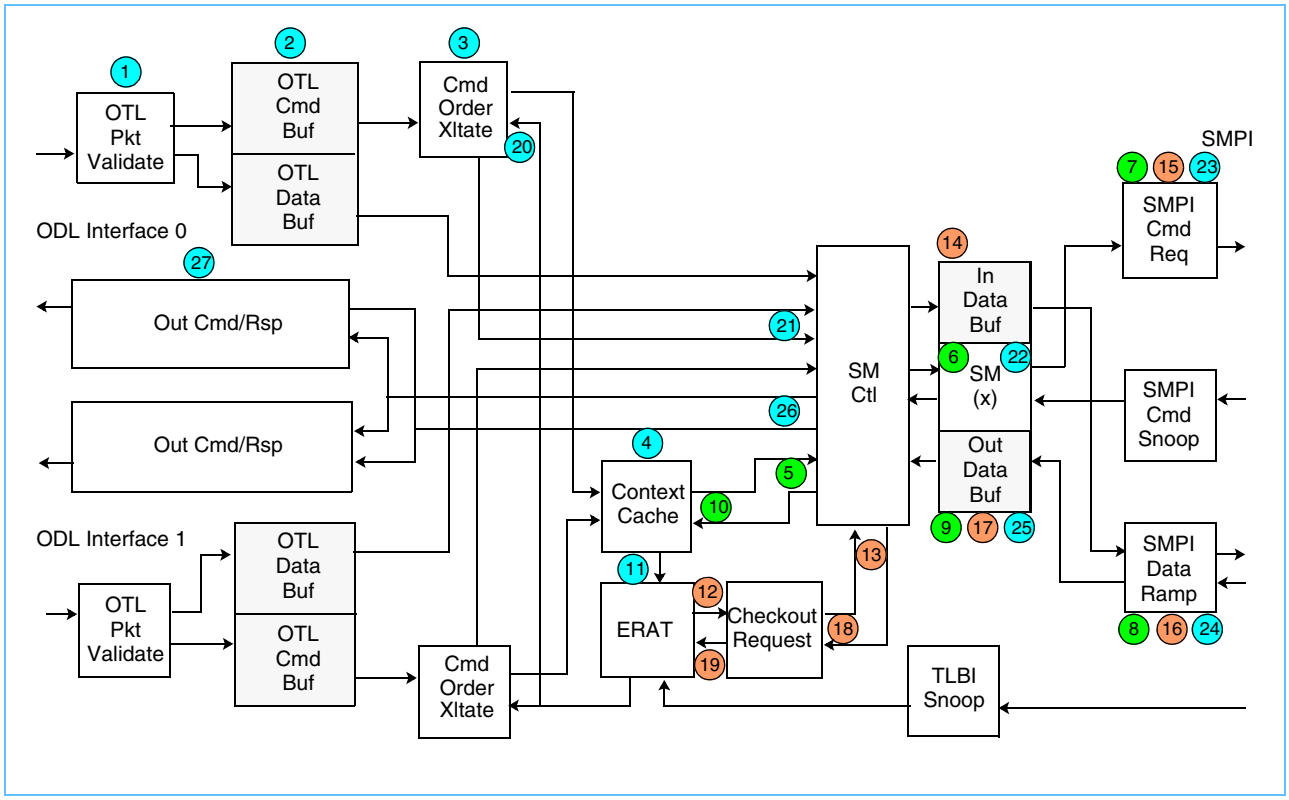
The XTS is represented by: Checkout Request and TLBI Snoop.

The CQ is represented by: SM Ctl, SM, In Data Buf, Out Data Buf, SMPI Cmd Req, SMPI Cmd Snoop, and SMPI Data Ramp.

14.5.1 Inbound Command/Data Flow

Figure 14-4 shows the flow for commands and data coming from the OpenCAPI link to the POWER9 chip.

Figure 14-4. PAU Inbound Command/Data Flow



The flow in *Figure 14-4* is described as follows:

1 A packet that arrives on the ODL interface must be validated. This includes checking the CRC status as well as checking for proper syntax. Additionally, the packet must be parsed into individual commands.

2 Commands that pass the validation check are stored in the OTL command buffer. For write commands, the write data is stored in the OTL data buffer. OpenCAPI transaction layer credits are maintained based on available space in these buffers.

3 OpenCAPI read and write commands contain effective addresses. These addresses must be translated to real addresses before the command can be sent to the on-chip SMP interconnect.

4 The first step in translating an effective address to a real address is to look up the translation context in a context cache. If the required context is found in the cache, the information is used in step 11. If the required context is not found in the cache, a context cache miss process is executed as shown in steps 5 - 10.

5 To load the required context into the context cache, a read from system memory must be done. The context cache control logic requests the read from the SM control logic.

6 The context read is assigned to a state machine (SM) for processing.

7 The state machine requests a read command on the on-chip SMP interconnect.

8 Data for the read is returned on the outbound on-chip SMP interconnect data port.

9 The data is stored in the outbound data buffer associated with the state machine that is processing the context read command.

10 The context data is sent to the context cache where it is loaded into the cache and used for step 11.

11 The context along with the effective address from the original command is used to find the real address in the ERAT. If the real address is found, it is returned to the command queuing structure in step 20. If the real address is not found in the ERAT, a checkout request must be sent to the nest MMU to obtain it. This process is contained in steps 12 - 19.

12 The ERAT control logic sends a checkout request to the checkout request tracking logic. The checkout request tracking logic manages the channels that can be used by the PAU to request real address checkouts from the nest MMU.

13 When a checkout channel to the NMMU is available, the check request tracking logic sends a checkout request to the state machine control (SM Ctl).

14 The checkout request is placed in a pre-reserved location in the inbound data buffer.

15 The checkout request is sent to the NMMU over the on-chip SMP interconnect inbound data ramp.

16 The response from the NMMU for the checkout request is returned over the on-chip SMP interconnect and arrives on the outbound data ramp.

17 The checkout response is placed in a pre-reserved location in the outbound data buffer.

18 The checkout response is sent to the checkout request tracking logic.

19 The checkout response is sent to the ERAT logic where it is loaded into the ERAT.

20 The real address is sent to the command queuing and ordering structure.

21 Commands in the OTL command queuing and ordering structure that are ready to be sent to the on-chip SMP interconnect are arbitrated against commands in the command queuing and ordering structure servicing the other OTL in this pair, as well as against commands coming from the on-chip SMP interconnect.

22 When a command in the OTL command queuing and ordering structure wins arbitration, it is assigned to a state machine. For writes, the data is moved from the OTL data buffer to the inbound data buffer.

23 The state machine controls the execution of the command. Any required on-chip SMP interconnect commands are sent to the on-chip SMP interconnect via the on-chip SMP interconnect command request port.

24 On-chip SMP interconnect data transfers related to requested on-chip SMP interconnect commands are handled through the on-chip SMP interconnect data ramps.

25 Data for read commands is stored in the outbound data buffer.

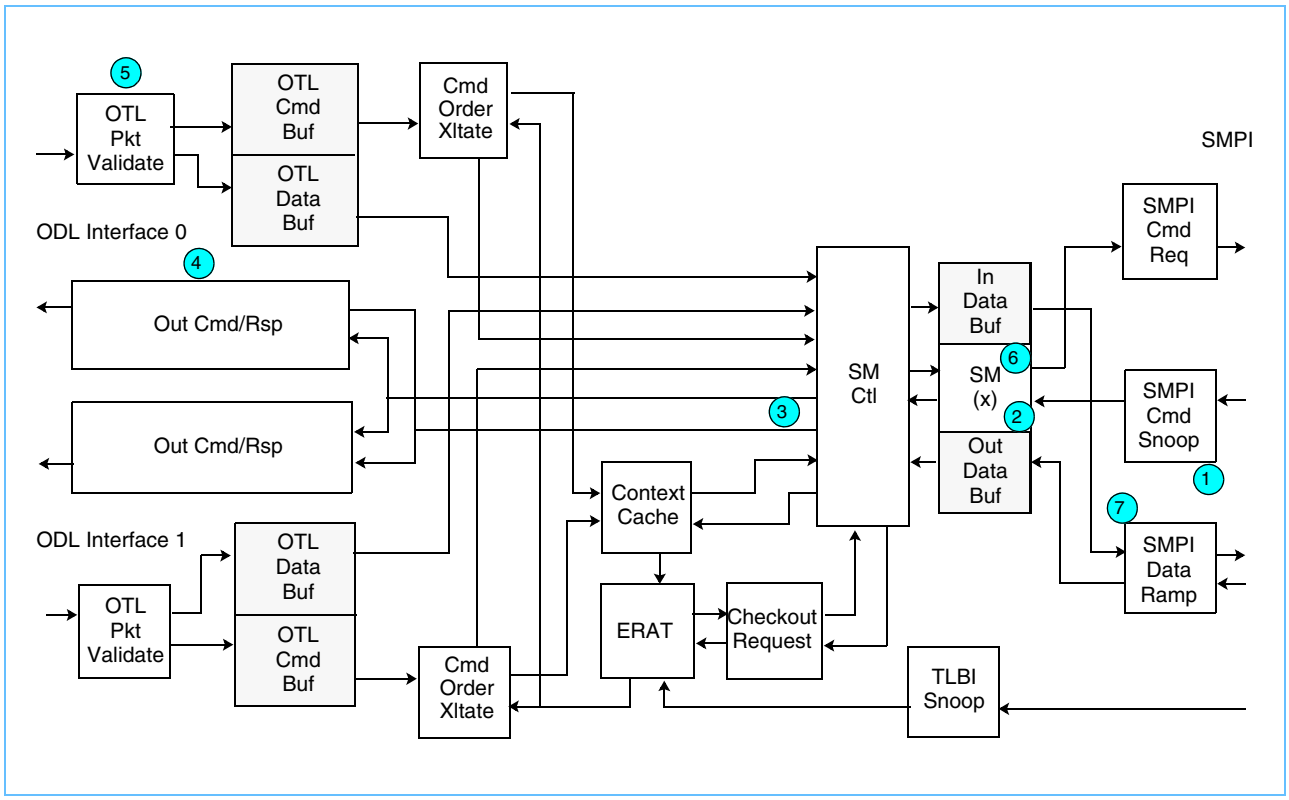
26 When all of the on-chip SMP interconnect command and data operations are complete for the command, the state machine indicates that it is time to send a response.

27 The out command/response block generates the response and includes any required data from the out-data buffer.

14.5.2 Outbound Command/Data Flow

Figure 14-5 shows the command and data flow for commands coming from the on-chip SMP interconnect and going to the AFU.

Figure 14-5. PAU Outbound Command/Data Flow



The flow in *Figure 14-5* is described as follows:

1 A command that accesses AFU memory space is seen by the on-chip SMP interconnect command snoop logic. For write operations, data is received on the on-chip SMP interconnect data ramp.

2 A state machine is assigned to process the command. Any data that is received is stored in the out-data buffer.

3 The state machine signals that a command must be sent to the AFU.

4 The out command/response function creates the command and includes any required data from the out-data buffer.

5 When the response for the command is received on the ODL interface, it must be validated. Validation includes checking the CRC status as well as the response syntax.

6 If the response is validated, it is passed to the state machine that is processing the command. Any data associated with the response is placed in the in-data buffer.

7 If the on-chip SMP interconnect command required data to be returned, the data is read from the in-data buffer and is put onto the inbound on-chip SMP interconnect data ramp.

14.6 POWER9 AFU Transaction Examples

The following sections contain examples of transactions between POWER9 and OpenCAPI AFU chips using the OpenCAPI link and the on-chip SMP interconnect.

14.6.1 Read from AFU to POWER9 Memory

Table 14-1 shows an example of a 128-byte read command sent over an OpenCAPI link.

Table 14-1. Read from AFU to POWER9 Memory

Step	AFU Drives OCL	OpenCAPI Link	PAU Drives OCL	PAU Drives SMPI	On-Chip SMP Interconnect	From SMPI	Comment
1	→	RD_WNITC		→	NON_CACHING_READ		The AFU requests a read-with-no-intent-to-cache (RD_WNITC). The PAU sends a non-caching read command to the on-chip SMP interconnect.
2					Memory_Ack	←	The POWER9 memory controller acknowledges the command. The read command is acknowledged by memory.
3					Data	←	The POWER9 memory controller sends data for the read command to the PAU over the SMPI data bus. Memory sends data for the read command.
4		READ_RESPONSE Data	←				The PAU sends a response containing the data over the OpenCAPI link to the AFU.

Note: OCL = OpenCAPI Link; Ack = Acknowledge; Resp = Response

14.6.2 AFU Writes to POWER9 Memory

Table 14-2 shows an example of a series of 128-byte write commands sent over the OpenCAPI link to the POWER9 chip. In this example, it is assumed that the final write must occur after the previous writes have completed. A case where this is required is when the final write is an indication to software that the previous writes were completed.

Table 14-2. AFU Writes to POWER9 Memory

Step	AFU Drives OCL	OpenCAPI Link	PAU Drives OCL	PAU Drives SMPI	On-Chip SMP Interconnect	from SMPI	Comment
1	→	DMA_W_0		→	DMA_INJECT_0		The AFU sends a stream of three DMA write commands on the same link. The PAU sends the write commands onto the on-chip SMP interconnect as DMA_INJECT commands.
2	→	DMA_W_1		→	DMA_INJECT_1		
3	→	DMA_W_2		→	DMA_INJECT_2		
4					Cache_Ack_1	←	The responses for the inject commands can arrive out of order. All responses indicate that the writes are globally visible. The inject hits the cache and is handled by the POWER9 memory.
5					Memory_Ack_0	←	
6					Memory_Ack_2	←	
7		Write-Response_1	←	→	data_1		The PAU sends the data for the inject commands and sends responses for each of the DMA write commands. Write responses can be sent on the OpenCAPI link out of order.
8		Write-Response_0	←	→	data_0		
9		Write-Response_2	←	→	data_2		
10	→	DMA_W_3		→	DMA_INJECT_3		Because the final write must occur after the previous writes are done, the AFU waits for all of the responses for the previous writes before sending the final one. The PAU sends this write command onto the on-chip SMP interconnect as a DMA_INJECT command. The write_3 command is sent after writes 0 - 2 have completed.
11					Memory_Ack_3	←	The combined response indicates that the inject command has completed on the on-chip SMP interconnect.
12		Write-Response_3	←	→	data_3		The PAU sends the data for the inject command and sends a response for the final write.

Note: OCL = OpenCAPI Link; Ack = Acknowledge; Resp = Response

14.6.3 Read from POWER9 to AFU Memory

Table 14-3 shows an example of a POWER9 read of the AFU memory.

Table 14-3. POWER9 Read of AFU Memory

Step	AFU Drives OCL	OpenCAPI Link	PAU Drives OCL	PAU Drives SMPI	On-Chip SMP Interconnect	From SMPI	Comment
1					READ	←	Read command from POWER9 requester. A read of AFU memory appears on the on-chip SMP interconnect.
2		RD_MEM	←	→	PAU_Ack		The PAU accepts the read command and sends the read to the AFU. The PAU responds with an acknowledge and sends a RD_MEM command on the OpenCAPI link.
3	→	MEM_RD_ RESPONSE		→	Data		When the AFU responds with the data for the read, the PAU places it on the on-chip SMP interconnect as the data transfer for the read command that was accepted in step 1 and sends the data to the requester.

Note: OCL = OpenCAPI Link; Ack = Acknowledge; Resp = Response

14.6.4 Write from POWER9 to AFU Memory

Table 14-4 shows an example of a POWER9 write to the AFU memory.

Table 14-4. POWER9 Write to the AFU Memory

Step	AFU Drives OCL	OpenCAPI Link	PAU Drives OCL	PAU Drives SMPI	On-Chip SMP Interconnect	from SMPI	Comment
1					WRITE	←	A write command from the POWER9 requester to the AFU memory appears on the on-chip SMP interconnect.
2				→	PAU_Ack		The PAU accepts the write command and responds with an acknowledge.
		WRITE_MEM	←		Data	←	The requester of the write command sends the write data to the PAU. The PAU then sends a write command to the AFU over the OpenCAPI link.
3	→	MEM_WRITE_ RESPONSE					When the AFU responds to the write, the command is complete.

Note: OCL = OpenCAPI link; Ack = Acknowledge; Resp = Response

15. CAPP

CAPP is the part of coherent accelerator processor interface (CAPI) that is on the POWER9 chip. CAPI is a means of attaching a remote accelerator to the POWER9 chip in a coherent manner. CAPP is the logic that connects to the local fabric that enables the remote accelerator to be able to be coherent. CAPP connects to the accelerators via PCIe.

CAPP supports a cache for the accelerators, as well as translation, and other miscellaneous functions.

16. Nest MMU

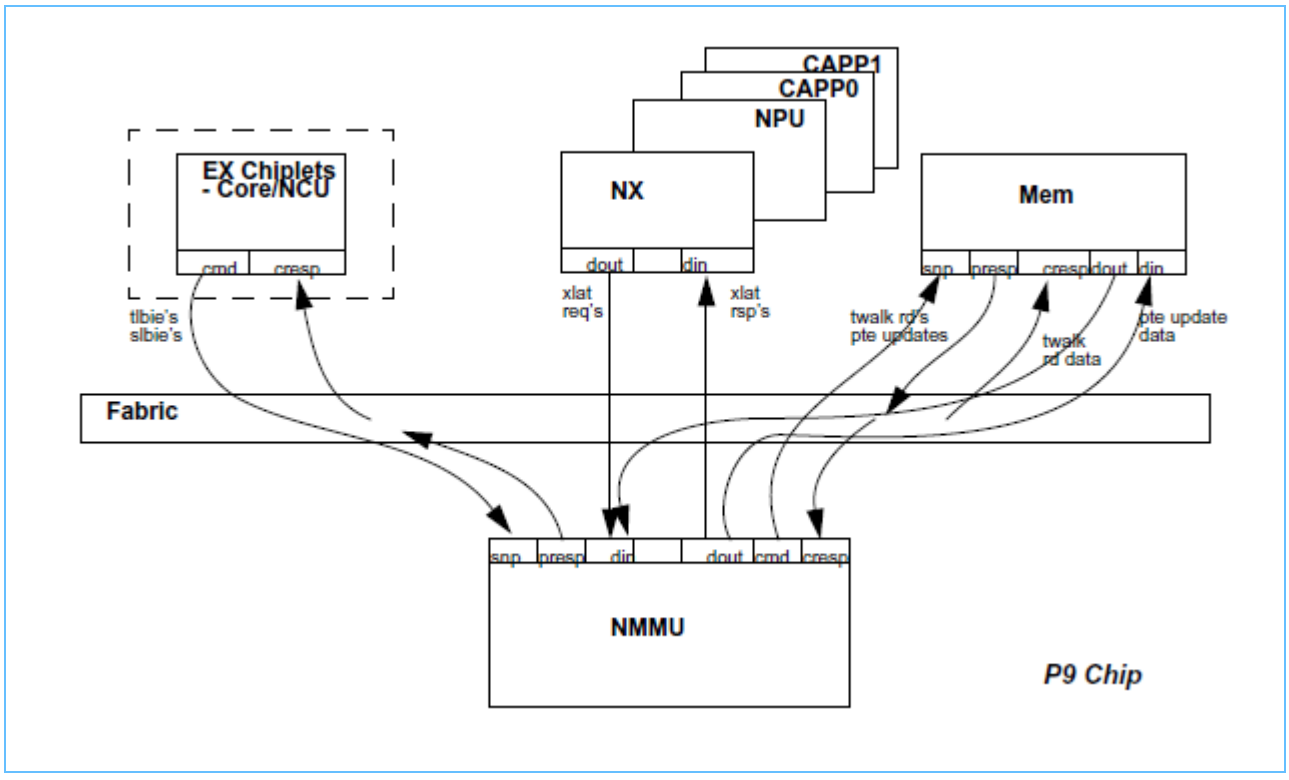
This chapter describes the overall microarchitecture for the nest memory management unit (NMMU) implemented in the POWER9 processor. The primary goal of this unit is to provide effective address (EA) to real address (RA) translation for the various accelerator agents within the processor's storage subsystem. In addition, the NMMU protects the pages that are being translated by ensuring that only tasks with the proper authorization are allowed to access them.

16.1 Overview

The NMMU resides within each POWER9 chip with coverage encompassing multiple classes of customers, including the on-chip NX, NPU, and CAPP0/1 units. The NMMU's primary function is to translate effective (logical) addresses into real (physical) addresses for memory accesses on behalf of these accelerator agents. The unit focuses on data accesses to memory generated by loads and stores. In its primary PowerPC mode, the NMMU's translation mechanism is defined by segment descriptors and page tables, as set up by the hypervisor. In addition to translation, the NMMU provides various levels of access protection on a per segment and page basis.

As shown in *Figure 16-1*, the POWER9 nest MMU primarily communicates with external units through the system bus (Fabric). All translation protocols with the accelerator units are run over the Fabric via data-only operations. The NMMU also interacts with memory to perform tablewalks and to update the page tables, as needed. In addition, cache management instructions (SLB and TLB invalidates) are sourced by the core/NCU of an EX chiplet in the system and are snooped and managed by the NMMU on behalf of the attached inclusive accelerator units.

Figure 16-1. POWER9 Nest MMU



16.2 NMMU Features

A summary of the NMMU features follows:

- Translation mechanisms supported:
  - Single-level translation
    - PowerPC hashed page table (HPT) approach (via POWERVM)
  - Dual-level translation
    - Radix-on-radix page table approach (via Linux over KVM)
- Functions supported:
  - EA-to-RA translations
  - Memory protection at segment and page levels
  - Page sizes supported:
    - Radix: 4 KB, 64 KB, 2 MB, 1 GB
    - HPT: 4 KB, 64 KB, 16 MB, 16 GB
  - Segment sizes supported: 256 MB, 1 TB for HPT translations
  - 64-bit effective address (EA), 68-bit virtual address (VA), 56-bit real address (RA)
  - Supports 12 simultaneous tablewalks
    - Responsible for acquiring segment table entries (STEs) and page table entries (PTEs) from segment and page tables residing in main memory
  - Optional TLB/SLB invalidation management on behalf of the inclusive accelerators (NMMU supports **slbie** and **tlbie**)
- Translation protocol:
  - Checkout phase
    - Agent requests that the tandem NMMU resolve a given translation
  - Check-in phase (applies to inclusive agents only)
    - Upon completion of relevant processing (active eviction) or local cache castout (passive eviction), the agent signals that the translation is no longer in use by checking it back into the NMMU's TLB.
  - Invalidation phase (applies to inclusive agents only)
    - Due to SLB/TLB invalidations snooped by the NMMU (**slbie**, **tlbie**)
    - Due to LRU castout of NMMU cache (TLB/SLB)
- Primary customers (nest accelerators that require translation):
  - NX: eight concurrent checkout/check-in operations total, one invalidate/barrier operation
  - NPU: eight checkout operations
  - CAPP0, CAPP1: eight checkout operations per unit
- Accelerator agent/NMMU interface communication mechanism
  - Via Fabric data bus (for common platform, floorplan flexibility, and future extendability)
- Cache resources available for translations
  - Local cache (ERAT) resides within accelerator agent
    - NX: 32-entry local cache (ERAT) arranged in a CAM/RAM structure
    - CAPP0/1: Assuming a 32-deep intermediate buffer for each CAPP ahead of the PSL ERAT
    - NPU: Assuming a 64-deep intermediate buffer ahead of the ERAT held in NVIDIA's GPU chip

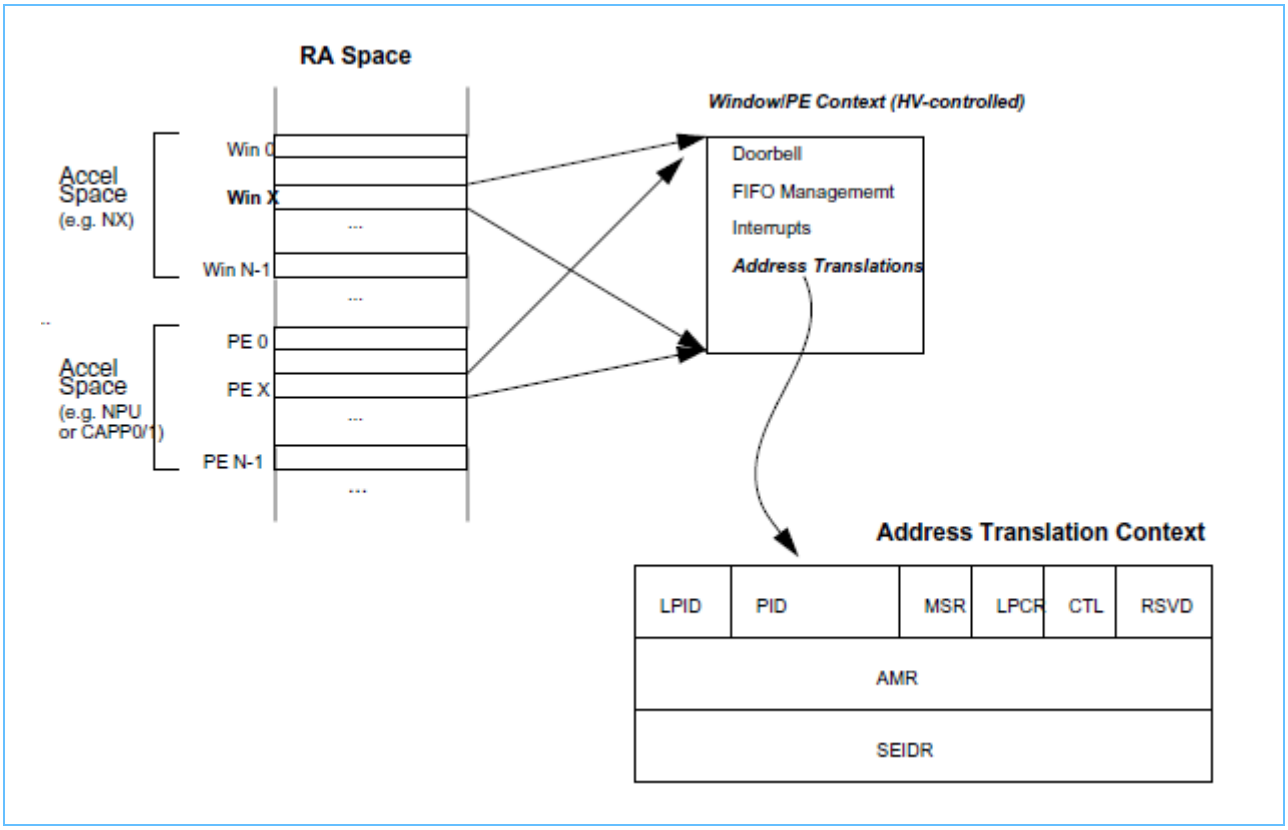
  - SLB
    - 256-entry, 16-way set associative (number of entries targeted at matching number of cached windows)
    - Used to cache the most recent copies of STEs
  - TLB
    - 8192-entry, 16-way set associative
    - Used to cache the most recent copies of PTEs
    - Inclusive of NX agent's ERAT
    - Partially inclusive for NPU/CAPP agents due to the volume of cached translations required and agent self-management of invalidations

- Radix Page Walk Cache (PWC):
  - L1: 256-entry, 16-way set associative (128 LPID/PID pairs per guest and host)
  - L2: 512-entry, 16-way set associative (128 pairs × 2 branches per guest and host)
  - L3: 1024-entry, 16-way set associative (128 pairs × 2 branches × 2 sequential entries/branch per guest/host)
  - L4: 2048-entry, 16-way set associative (128 pairs × 2 branches × 4 sequential entries/branch per guest/host)
- Fabric snoop interface:
  - Four sets of snoop and partial response interfaces
  - One 16-byte inbound data ramp
  - One 16-byte outbound data ramp
  - Primarily for agent communication, SLBI/TLBI-related commands (assuming no MMIOs required)
- Fabric master interface:
  - One master interface (single command bus, four combined response ports)
  - Used for in-memory table reads (for example, process table), STEG/PTEG lookups, and PTE updates
- Clocking/frequencies:
  - Nest clock frequency (2 GHz), 1:1 with Fabric

16.3 Window/Process Element Context

For the accelerators in the nest, a window or process element (PE) is the communication mechanism where trusted software provides the context required by the hardware to process a given packet or message. The context for a window/process element is located in main memory, as illustrated in the following diagram, and is maintained by the hypervisor. A window/process element is the portal that links system software and hardware together and provides the common structures that dictate how tasks through this window are processed. Ultimately, a window ID (or process element ID) is the index into the context stored in memory.

Figure 16-2. Window/Process Element Context



Any agent that requires use of the NMMU to translate an effective address (or guest virtual address) must provide the address translation context (see *Figure 16-2*) with the corresponding checkout request. For the format of the address translation context within a process element, see the *Coherent Accelerator Interface Architecture*.

16.4 Nest Translation Cache Pipeline

Figure 16-3 on page 242 gives a high-level view of the pipeline used to satisfy translation requests within the nest.

While processing a packet, an agent determines that it needs a translation for a given effective address and pulls its corresponding window/PE context. The agent first performs a lookup of its local cache (ERAT) to see if the translation has already been checked out. An agent can hold onto a translation within the processing engine or at the window level. If the translation is available locally, the agent services the translation request from its own cached copy. If the translation is not available (for example, an ERAT miss), the agent forwards a translation request to the NMMU.

The translation request travels over the Fabric via a 32-byte data-only tenure on its way to the agent's partner NMMU. After the translation request arrives in the NMMU, it collects in an agent input buffer. The depth of the queue equates to the total number of outstanding requests for a given source into the NMMU, which corresponds to eight checkout requests, eight check-in requests (for inclusive agents only), and one barrier/invalidate operation (for inclusive agents only). The requests are generally serviced in a round-robin fashion.

An arbiter selects which operation to send into the NMMU's SLB/TLB pipe. Highest priority are the invalidate (TLBIE/SLBIE) commands that arrive downstream from the Fabric. The other legs that feed into the arbiter are the requests from the agent input buffer and any internal operations that are needed (for example, directory updates into SLB/TLB). Because these internal operations release dependencies for previously requested operations, they are generally prioritized over new translation requests from the agent input buffer.

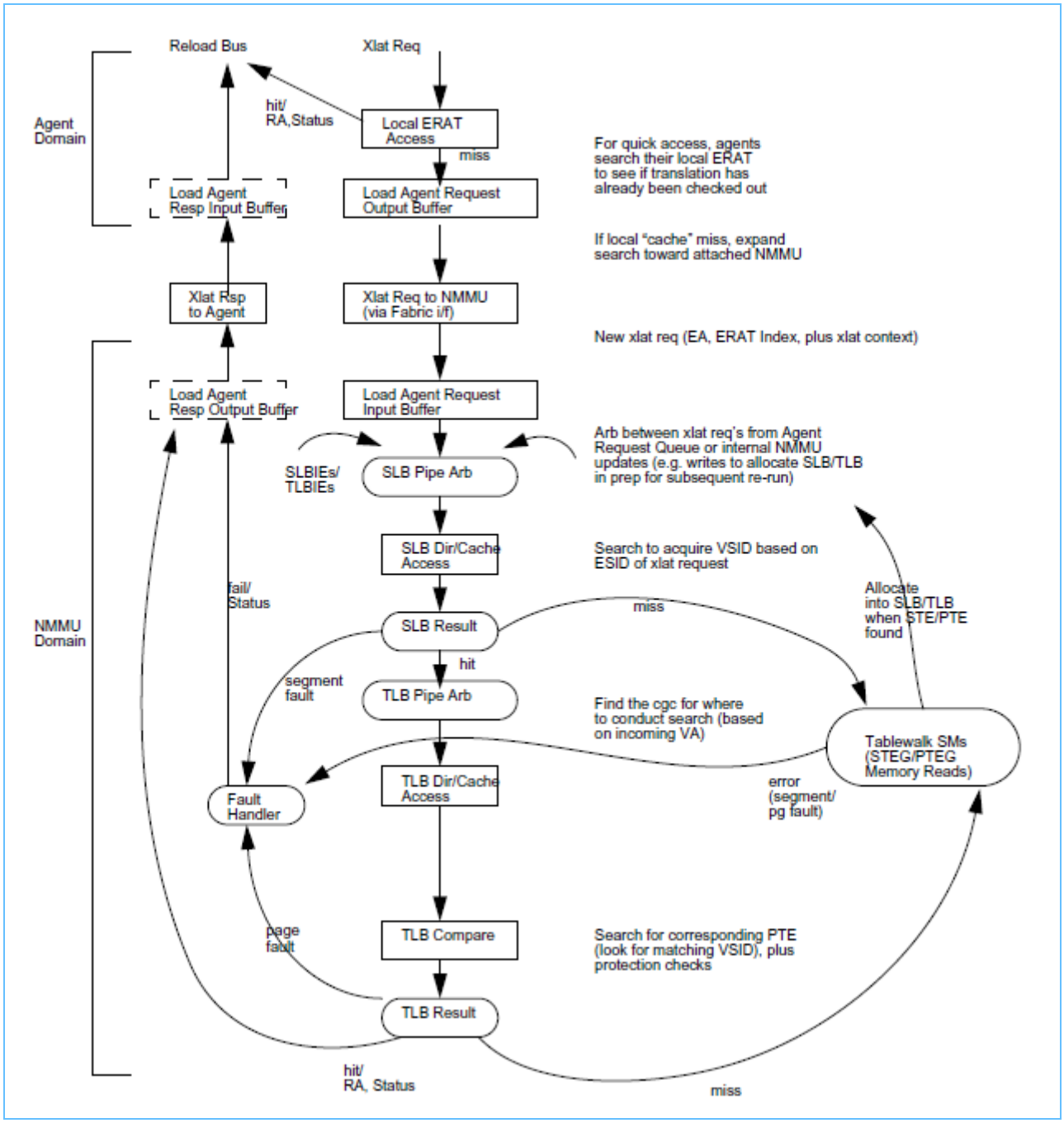
After a request is granted access to the NMMU's pipe, it performs a lookup of the SLB (for HPT translations) to find the corresponding virtual address. The NMMU searches (based upon a congruence class hash) the SLB directory for a matching ESID from the EA to derive the VSID from the SLB cache, which allows the VA to be formed. For Radix translations, the SLB is bypassed and the guest translation is forwarded to the TLB for resolution leading to a guest RA (host VA).

When the VA is determined, it is used to access the TLB. A hash of the VA is executed to isolate the congruence class (cgc) of the TLB to search the cache directory. Compares are executed to find the corresponding PTE for the translation request. The NMMU looks for a matching VPN, LPAR ID, PID, page size, hash type, and so on, among its criteria to find the appropriate PTE. The NMMU also performs protection checks in conjunction with key bits from the SLB and PP bits from the TLB.

If a matching PTE is found, the RA, page size, qualified C-bit, and status is returned to the agent immediately. For non-inclusive agents, some additional information is returned with the response to help with its subsequent slbie/tlbie management. If an error (for example, a segment fault or page fault) is discovered, fail status is returned to the agent and an error interrupt is sent by the agent to alert software of the problem.

If a matching PTE is not found, the miss is dispatched to a tablewalk state machine to resolve the translation request. The tablewalk machine drives read requests to memory to pull in the STE and/or PTE for the translation. After the STE/PTE data arrives, it is allocated into the SLB and/or TLB, respectively. After the cache and directory are updated, the tablewalk state machine recycles the translation request through the SLB/TLB pipe on behalf of the corresponding agent, which causes the entry to be re-run through the SLB/TLB pipe. This re-run should cause an SLB/TLB hit to occur, which allows the RA and status for the translation to be returned to the agent.

Figure 16-3. Nest Translation Pipeline



16.5 Nest Translation Protocol (for Fabric-Attached Agents)

The nest translation protocol is partitioned into three primary phases: the checkout of a translation, the check-in of a translation, and the invalidation of a translation due to a snoop TLB invalidation hit or due to the eviction of a NMMU cache entry. The following sections describe these various phases with a particular focus on agent communications being performed over the Fabric.

16.5.1 Translation Checkout

For an agent to obtain a translation, it must request that the translation be checked out. To do this, the accelerator agent initially captures the corresponding window/PE context for the access to pull in the relevant information pertaining to the address translation. If the translation has not already been checked out, the accelerator agent forwards a 32-byte, data-only operation on the Fabric, along with the agent's ERAT index, which is logged in the NMMU agent request-in buffer. The data tenure routes the operation to the destination NMMU using the RTag, which includes the payload (EA, plus translation context data optimal for an NMMU hit [for example, an LPID or PID]). After resolving the translation request, the NMMU returns a translation response to the agent that initiated the request. This is done by another data-only tenure on the Fabric with the corresponding real address, the associated status for the translation request, and the page size for the translation. After the agent acquires the translation, it is valid until the NMMU indicates that it must be invalidated, until the agent checks in the translation after it is done with its processing, or until the page boundary is crossed. See *Figure 16-4* on page 246 for more details.

Within the unit, the NMMU snoops the checkout request and performs a lookup of its SLB/TLB to see if the translation already exists. If there is a hit in the NMMU cache, the corresponding real address is provided to the accelerator agent immediately. If the request misses the SLB and/or TLB, the NMMU performs a table-walk to obtain the targeted RA. This is done by acquiring the corresponding segment table entry (STE) and/or by claiming the required page table entry (PTE) from main memory. The NMMU sets a flag in its TLB and/or SLB to represent that the translation is in use by an inclusive accelerator only. The inUse flag remains set until an agent check-in occurs or until the entry is invalidated.

16.5.2 Translation Check-in

When an inclusive agent is finished with a specific translation that it knows is no longer required,[1] it drives a request to the NMMU with the corresponding ERAT index to check-in the translation. If handled correctly, this can lessen the impact of an invalidation for a given translation. This is in contrast to allowing it to turn into an asynchronous forced invalidation at some point in the future when the NMMU snoops a tlbie/slbie (or when a NMMU TLB/SLB cache eviction occurs due to capacity reasons) and the corresponding NMMU TLB/SLB entry has the inUse flag set. The primary intent of an active eviction policy within an accelerator is to aid the NMMU TLBIE/SLBIE management. By clearing the corresponding inUse bit within a given NMMU TLB/SLB entry due to a check-in, the entry can be invalidated for a subsequent TLBIE/SLBIE without invoking a back-invalidation sequence with an agent. *Figure 16-4* on page 246 shows a high-level view of the check-in flow between an accelerator agent and the NMMU.

1.Translations can be held in the agent on a per window basis or within a common pool across windows.

16.5.3 Translate Invalidation Interface

At times, the NMMU initiates an invalidation sequence with an inclusive agent to indicate that a translation is no longer valid. An invalidate sequence can be triggered by either of the following two scenarios:

- A snoop invalidate (slbie/tlbie) that hits an entry in the SLB or TLB.
- When the NMMU evicts a valid SLB or TLB entry.

For both cases, the valid SLB or TLB entry must have its inUse flag set. which means that the translation is currently being used by an inclusive agent. In this scenario, the NMMU accepts the tlbie/slbie operation and protects the page/segment accordingly until the invalidation sequence with the agent is completed. The NMMU drives an invalidation request to any agent whose inUse bit is set for a matching SLB or TLB entry, along with the affected ERAT index, which is held within the NMMU inUse scoreboard. The sequence is completed when the agents drive a response back to the NMMU that indicates that the agent has quiesced by draining all pending operations for any outstanding translations for the targeted ERAT index (or transaction ID for non-inclusive agents) to the Fabric. After the barrier is detected, subsequent operations to the Fabric are halted within the agent until the drain is completed and new translations are obtained from the NMMU. Also, this invalidation sequence can additionally be initiated by the NMMU when its LRU algorithm leads to evicting a valid TLB/SLB entry with its inUse flag set.

To coordinate this event between an inclusive agent and the NMMU, the following sequence is executed:

1. When the NMMU detects an inUse flag that is set for a given cgc, it sends a raise-barrier request to the targeted agent to initiate a back-invalidate sequence.

   The NMMU asserts an internal retry window for subsequent in-flight checkout/check-in requests from the given agent.

2. Upon receiving the raise-barrier operation, the agent stops sending any new checkout/check-in requests to the NMMU until it detects a lower-barrier operation.

   Checkout/check-in operations that are already in the outbound request queue can be sent as the ERAT cannot pull these back.

3. The corresponding agent waits for a response from the NMMU for all outstanding checkout/check-in requests.

   If tablewalk or check-in machines have been started already, the NMMU allows those to finish before providing a response.

   Note that a tablewalk state machine is forced to abort if a castout of a cache member is detected with its inUse flag set. This is required to ensure that the NMMU can always provide an xlat response.

   The NMMU drives a "retry due to invalidate" status for all in-flight operations into its SLB/TLB pipe sourced by the targeted agent.

   Upon receiving a retry status, the agent can recycle the request after the barrier clears.

4. When the agent receives responses for all pending checkout/check-in requests, it sends an acknowledge for the raise barrier to establish the barrier and indicate that it is ready to receive the invalidate.
5. After it detects the raise barrier acknowledgment from the agent, the NMMU drives an invalidate request to the corresponding accelerator.
6. Upon receiving the invalidate request, the agent does the following actions before invalidating its local ERAT entry (or transaction ID for non-inclusive agents):

- Compares the ERAT index for the invalidate request with any pending check-in requests. If a match is found, the ERAT index is implicitly checked in (that is, removed from the queue as though the operation completed).
- If an ERAT index is in use by a DMA read/write machine, the agent waits for the corresponding index's access count to drain to zero (that is, no pending accesses to the Fabric for the given index).

7. The targeted agent drives a response to the NMMU for the invalidate request.
8. After detecting the agent's response, the NMMU invalidates its local SLB or TLB entry (dependent upon which cache entry is being invalidated) and updates its inUse flags accordingly.
9. For a given cgc, if the NMMU still has other members to invalidate with an inUse flag, it loops back to step 5 in the sequence to process the next cgc member under the current raised barrier.

   Note: The NMMU might amortize the barrier for processing subsequent cgc's if the invalidate is one in which the entire TLB is scrubbed (that is, for a tlbie for all entries or those matching an LPID/PID combo).
10. When the NMMU finishes cleaning the inUse flags for the cgc (or a larger working set), it sends the targeted agent a lower barrier request.
11. Upon receiving the lower-barrier request, the agent sends an acknowledge to the NMMU. This allows the agent to resume sending checkout/check-in requests to the NMMU.
12. The NMMU detects the lower-barrier response and drops its retry window for in-flight xlat operations.

16.5.4 Flow Diagrams of Agent/NMMU Translation Operations

This section summarizes the primary flows between an agent and the NMMU on behalf of address translation.

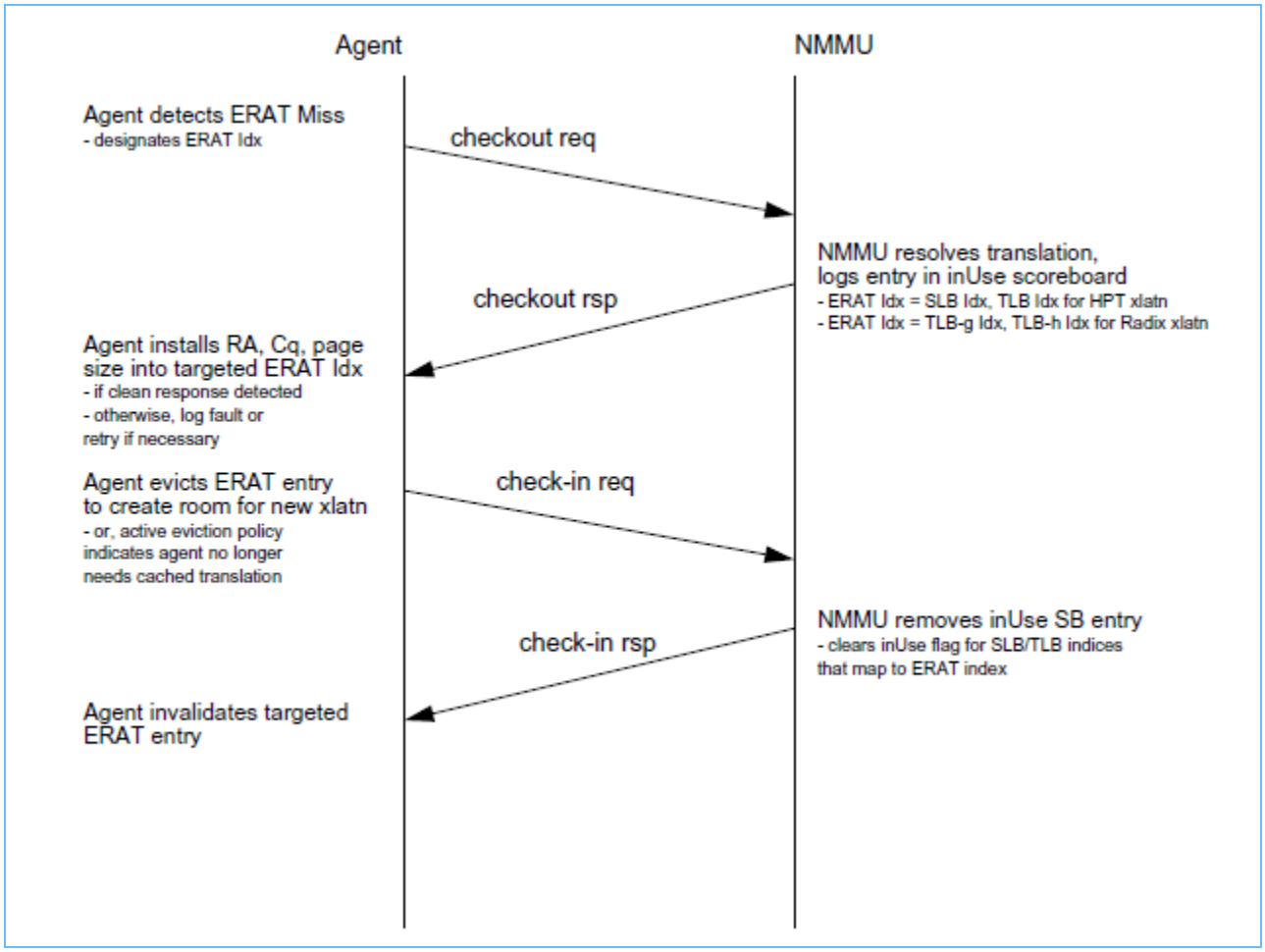
16.5.4.1 Checkout/Check-In Sequence

This section illustrates the sequence of events that an agent performs to checkout and check-in a translation from the NMMU. When an agent detects an ERAT miss (and there is an available ERAT entry), it forwards a checkout request to the NMMU. The NMMU services the translation and returns a response to the given agent for its checkout response. Coincidentally, the NMMU logs the ERAT entry in its inUse scoreboard as a part of its filtering mechanism for snoop invalidates and local castouts on behalf of inclusive agents.

When an inclusive agent is ready to evict a translation from its ERAT, it masters a check-in request to the NMMU. Upon receipt, the NMMU removes the entry from its inUse scoreboard and clears the corresponding inUse flags from the SLB and/or TLB.

After the inclusive agent receives a clean check-in response, it invalidates the targeted ERAT entry and is able to re-use the entry.

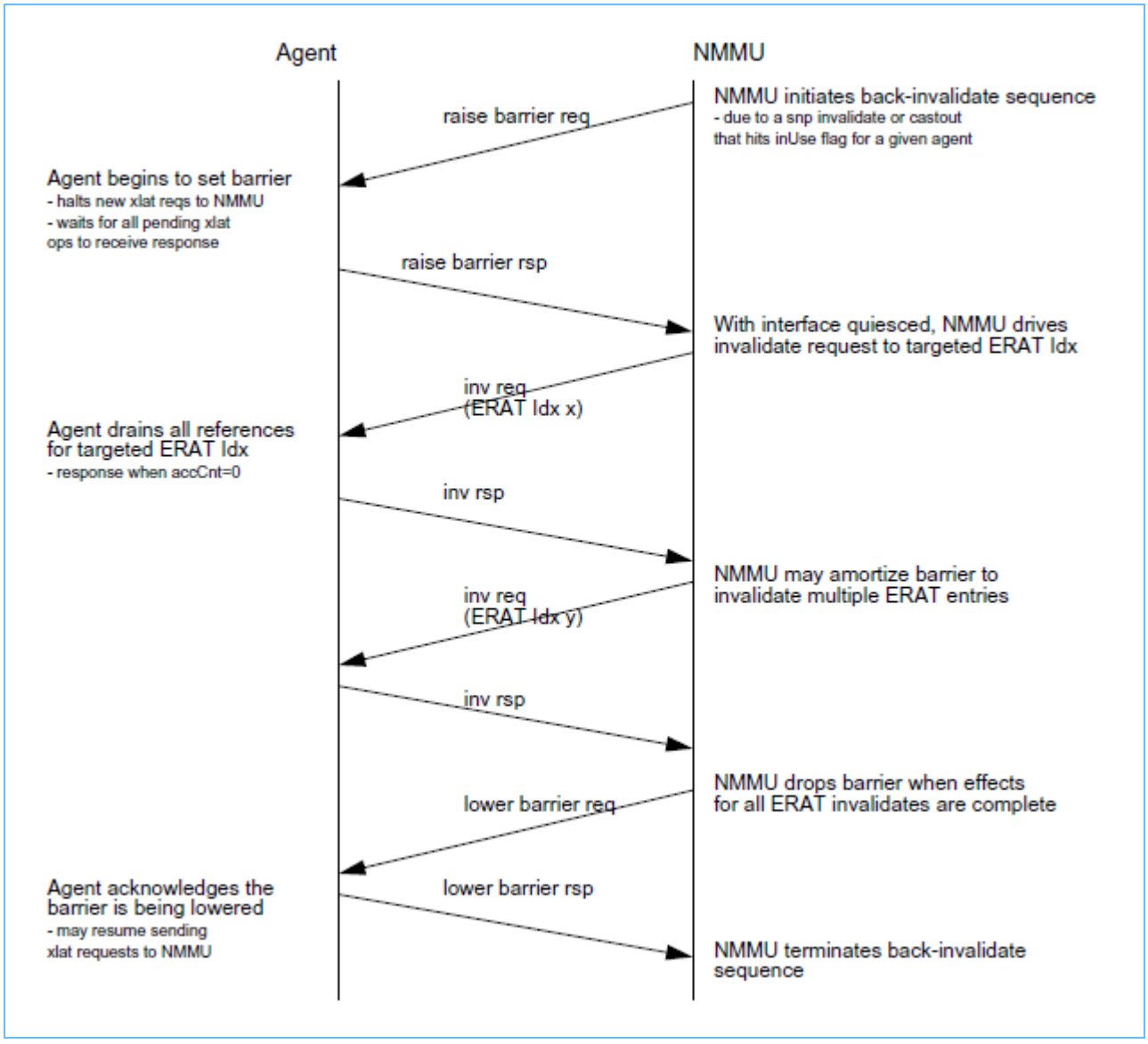
Figure 16-4. Agent/NMMU Flow Diagram (Checkout/Check-in)



16.5.4.2 Back-Invalidate Sequence

This section highlights the back-invalidate process between the NMMU and its inclusive agents. The trigger for this sequence is a snoop invalidate or local castout that hits an inUse flag (that is, translation has not been checked in). When this occurs, the NMMU initiates a raise barrier phase to warn the agent that an ERAT invalidate is coming. The agent withholds a response for the raise barrier until it receives responses for all pending xlat operations (that is, the interface has quiesced). When this phase completes, the NMMU cycles through targeted ERAT invalidates one at a time. An invalidate response from the agent indicates that it has drained all references for the translation. The NMMU lowers the barrier when it permits the agent to resume its normal traffic flow for translation operations.

Figure 16-5. Agent/NMMU Flow Diagram (Back-Invalidate Sequence)



16.5.5 NMMU Cache Pipeline

Figure 16-6 on page 249 illustrates a high-level view of the NMMU's cache pipeline after an accelerator agent issues a translation request to the NMMU. Note that this is primarily an HPT translation path, but a Radix translation has a similar flow.

The process begins as the NMMU arbitrates to select the next request to be serviced by the SLB pipe resource. Highest priority is given to SLBIE broadcast commands that are snooped via the Fabric bus. These commands require a unit partial response on the Fabric in a guaranteed time (TLAR), so that they are considered non-blocking operations. The next highest-priority level is aimed at completing updates for commands that have already been issued through the pipeline and are being processed by a tablewalk or invalidate state machine. These updates include directory writes and if needed, cache writes, internal tablewalk directory reads, and also recycled translations that should now hit in the SLB. By allowing these internal operations to complete, downstream resources are freed up to accept new translation requests. The third priority level is assigned to retry requests. These are requests that were sent through the pipe, but the NMMU could not resolve the translation at the time due to a dependency. Typical dependencies include congruence class collisions against pending queues in the NMMU that are processing misses or invalidations. The lowest priority for the SLB pipe arbitration point is designated for new translation requests that have not been issued to the pipeline yet.

After a request is selected to enter the SLB pipe, a lookup of the SLB is performed to obtain the virtual address for the corresponding effective address. If the virtual address is not cached within the SLB, the miss is dispatched to a tablewalk state machine to find a matching entry in the corresponding segment table located in main memory.

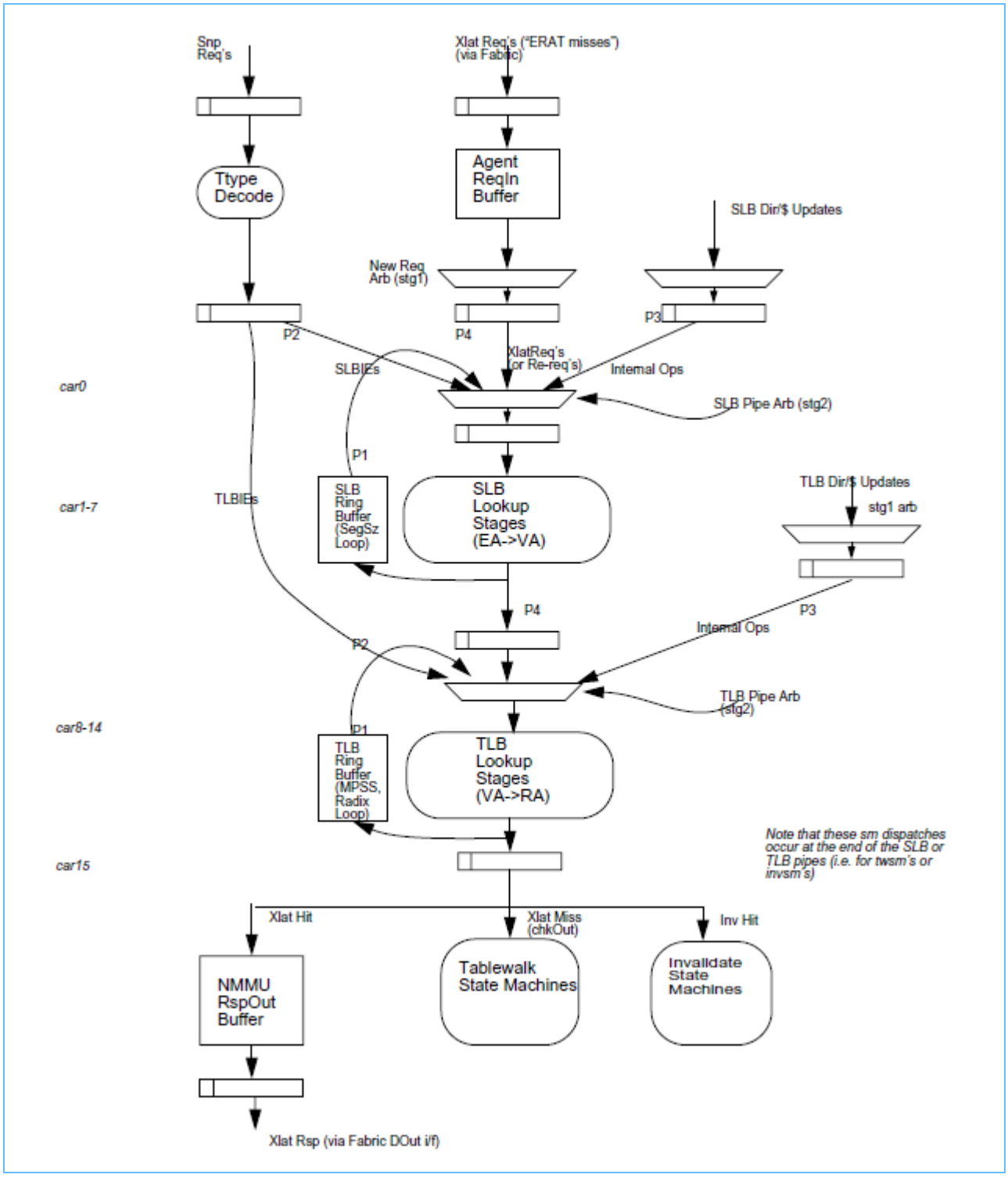
Once resolved in the SLB, the NMMU's pipe continues with arbitration for the TLB lookup phase. Similar to the SLB resource, the highest priority is assigned to TLBIE broadcast commands that are snooped via the Fabric. Likewise, these are considered nonblocking operations because they require a unit partial response on the Fabric within a TLAR interval. The next highest priority targets TLB directory/cache updates, internal tablewalk directory reads, and tablewalk recycles of newly installed translations for pending misses that the NMMU is processing. The third priority belongs to new translation requests, which would arrive downstream from the SLB pipe for HPT-based requests or from the SLB pipe's bypass path for Radix-based translations.

After the TLB pipe arbitration grants the next request, the pipeline continues with a lookup of the unit's TLB. The TLB lookup allows the NMMU to check its local cache to determine if a valid copy of the real address is present. If a matching page table entry is not found in the NMMU's TLB, a tablewalk machine is dispatched to search for one in the corresponding page table residing in main memory.

When the real address is successfully found at the end of the SLB/TLB pipe, the result is forwarded to the NMMU response out FIFO to be transferred through the Fabric to the accelerator agent that originally requested the translation.

If the operation sent through the SLB/TLB pipe is a snoop invalidate command (SLBIE/TLBIE), the NMMU looks up the corresponding SLB or TLB to determine if one or more matching translations are currently checked out by an accelerator. If the operation misses the NMMU's cache, a null partial response is returned to the Fabric indicating that the segment or page is not present within the given NMMU's scope. If the TLBIE or SLBIE operation hits in the NMMU's cache and a corresponding inUse flag is set, a retry partial response is returned to the Fabric until the NMMU can successfully drain all references for pending translations in the associated accelerators. If the SLBIE operation hits in the NMMU's SLB cache, no drain is required within the accelerators, but it is performed for simplicity and uniformity with TLBIE behavior with respect to the agents.

Figure 16-6. High-level NMMU Translation Pipeline



16.5.6 NMMU Control State Machines

There are several types of control state machines that manage the processing of the various translation requests that the NMMU receives. An engine ID is used to uniquely identify a given state machine.

16.5.6.1 Tablewalk State Machine

In the POWER9 NMMU, a tablewalk state machine is responsible for resolving a miss for a translation checkout request from an accelerator. It can service an SLB miss or a TLB miss. For the SLB miss case, the tablewalk engine searches for a valid matching segment table entry (STE) in memory that it can then install in the SLB cache. Likewise, for a TLB miss scenario, a tablewalk state machine searches memory for a valid matching page table entry (PTE) that it subsequently allocates in the TLB cache. After the respective cache miss is satisfied, the tablewalk state engine recycles the original checkout request through the SLB/TLB cache pipeline with the expectation of a subsequent hit within the NMMU. Upon a hit, the NMMU returns the RA and corresponding status to the targeted accelerator.

16.5.6.2 PTE Update State Machine

When a tablewalk state machine must update the PTE, it farms this task out to a PTE update state machine for this back-end processing. That is, R,C updates on behalf of a given PTE are all routed through the Fabric by these partner PTE update engines. Likewise, there is a 1:1 correspondence with the tablewalk engines, which means that there are 12 PTE update state machines for the POWER9 processor. The PTE update state machine's responsibility is simply to manage these updates to memory by mastering partial writes (ARMWF operations) on the Fabric. The PTE update state machine is allowed to go idle when it receives a clean cresp and it has evaluated the corresponding PTE valid bit from memory. There is a dedicated address queue slot per PTE update state machine to process these requests.

In the POWER9 processor, PTE updates are atomic updates of the PTE. They are treated just like atomic RMW with fetch operations on the Fabric with a special ttype (pteUpdt2). All updates of the PTE are considered to be atomic, including the R, C, and timestamp bits. The RMW is conditional in the targeted memory controller based upon the PTE valid bit being set for the appropriate PTE format. If the PTE valid is set, the RMW is allowed to occur. If the PTE valid bit is not set, the RMW is aborted in the memory controller. The memory controller returns the original PTE state to the NMMU with the fetch portion of this operation. The NMMU tablewalk state machine determines the success or failure of the atomic PTE update based on sampling the fetch data matching the expected PTE data. If the PTE update fails, a fail response status is returned to the corresponding agent. All data tenures are cache-line transfers with an address offset to target a given PTE. The address for a PTE update is the PTEG RA for HPT or the targeted PTE for a Radix translation.

Because the fetch data for a PTE update is returned to the NMMU and stored in the tablewalk's read buffer in the Fabric macro, the tablewalk state machine must ensure that it has no outstanding reads on the Fabric before executing an atomic PTE update.

16.5.6.3 Castout State Machine Overview

At times, the NMMU must evict a member from its SLB or TLB cache according to its LRU algorithm. When the member is clean (the entry is valid and its inUse flag is not set), the existing LRU member is simply overwritten by the new member data. However, if the member is dirty (entry is valid and its inUse flag is set), a castout state machine must process the eviction in much the same way as the snoop slbie/tlbie invalidate scenarios described earlier. The primary difference, though, is that the invalidate due to a castout is guaranteed to be for a single member (LRU member) for a given congruence class, whereas a snoop invalidate can impact multiple members within a cgc.

When a dirty checkout miss is detected, the SLB or TLB pipe simultaneously dispatches a tablewalk state machine for the new request and a castout state machine to clear out the old entry. Likewise, there is a 1:1 correspondence between a tablewalk state machine and its partner castout state machine, so that there are 12 castout engines for the POWER9 processor. The tablewalk state machine is allowed to search for a matching STE/PTE for the new request during the castout phase, but it cannot allocate the new STE/PTE until the castout state machine completes its eviction of the old STE/PTE.

Similar to an invalidate state machine, the castout state machine takes the following steps to evict a given SLB or TLB index (cgc/member):

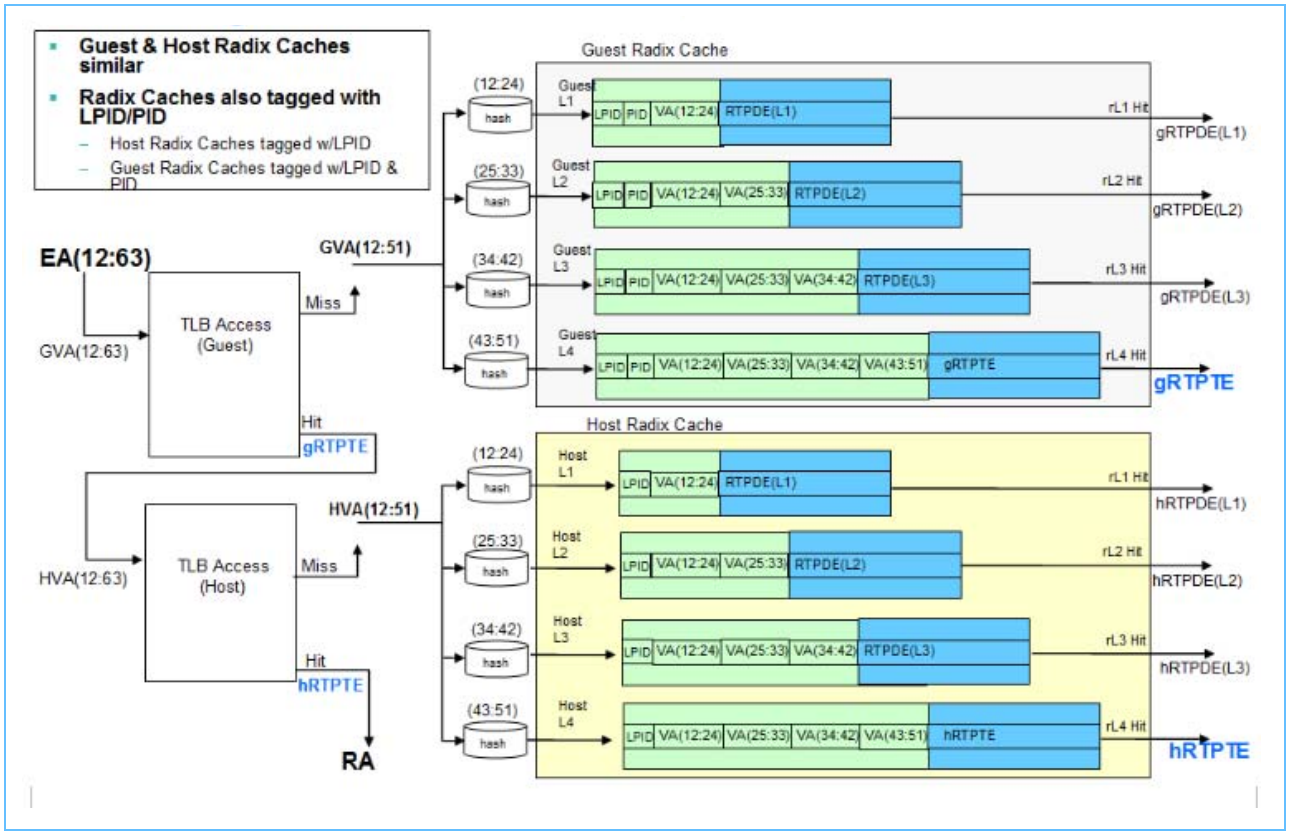
- At cache pipe dispatch, a castout engine is invoked because of a dirty checkout miss. The castout state machine captures the cgc and the LRU member for the SLB or TLB (the primary index).
- The castout state machine issues a request to the global inUse scoreboard to map the SLB or TLB index for the castout to the corresponding ERAT index, which correlates to the entry in the agent that must be torn down.
- After a response is provided by the inUse scoreboard, the castout state machine captures the ERAT index for the targeted SLB/TLB cache member and the SLB or TLB index (secondary index) that is not directly being invalidated.
- The castout engine drives invalidate requests to the corresponding inclusive agents for the given ERAT index.
- In the case of an SLB castout, there might be multiple invalidate requests in-flight under a barrier. Likewise, for Radix translations, there might be multiple invalidate requests to an agent for a TLB castout.
- When the castout state machine detects an invalidate response from the agent, the engine issues a directory write to the primary index (SLB or TLB) to clear the valid bit and to reset the inUse flag for an SLB/TLB castout. The castout state machine also issues a directory write to the counterpart cache to simply update its inUse flag.
- The castout state machine issues a clear command to the global inUse scoreboard to invalidate the corresponding ERAT index entry.
- While the castout state machine is processing a primary index eviction, it issues a query to the inUse scoreboard to see if any translations to the targeted primary index are still active. If there are still pending translations for the primary cgc member, the castout state machine wraps back to the step where it attempts to find the next ERAT index that must be processed.
- When agent shootdowns complete for the cgc member, the castout state machine sends a done pulse to its partner tablewalk state machine to give it clearance to allocate the new member in the cache. The castout state machine subsequently goes idle at this juncture.

16.5.6.4 Radix Page Walk Cache Overview

The Radix page-walk cache (PWC) is used by the tablewalk state machine to help it resolve a Radix translation. The tablewalk state machine stores guest and host page directory entries (PDEs) and PTEs for each level of the Radix tree (L1 - L4) in the cache.

Figure 16-7 shows an overview of the Radix PWC. A guest TLB miss can cause a tablewalk state machine to look in its PWC for a matching entry based upon the LPID/PID and the guest VA (GVA). Likewise, a host TLB miss can cause a tablewalk state machine to look in its PWC for a matching entry based upon the LPID and the host VA (HVA). When a matching PTE is found, the tablewalk state machine allocates it into the TLB for subsequent processing. L3 and L4 contain two to four consecutive PDE/PTEs to further help with prefetching for a sequential stream. Note that 1 GB pages map to a PTE found in the L2 of the PWC, 2 MB pages map to a PTE found in L3 of the PWC, and 4 KB/64 KB pages map to a PTE found in L4 of the PWC.

Figure 16-7. Radix Page-Walk Cache



16.5.6.5 Check-in State Machine Overview

A check-in request is initiated by an inclusive accelerator when it is finished using a translation that was previously checked out. From an accelerator's perspective, it essentially correlates to a castout of its local "cache" for a given translation. However, an accelerator can also implement more proactive measures to evict an entry out of its ERAT to check-in a translation. A check-in operation causes the NMMU to reset the corresponding inUse flag in its TLB and SLB to indicate that a given PTE or STE (alternatively, two PTEs with TLB-g, TLB-h for Radix translations) is no longer actively being used. This has the side effect of speeding up the resolution of tlbie/slbie commands on the Fabric.

An outline of the steps involved with the NMMU servicing a check-in request are as follows:

1. Check-in request is sent by an accelerator with a tag that indicates the agent ERAT index. The ERAT index is the globally unique identifier for a given translation between an agent and the NMMU.
2. The NMMU receives the check-in request into its agent input buffer.
3. When a downstream resource frees up, the NMMU pulls the request out of its agent input buffer and routes it to an available check-in state machine. A total of eight check-in engines are available for the POWER9 processor.
4. The check-in state machine issues a request to the global inUse scoreboard to map the ERAT index for the translation to the corresponding NMMU cache resources, which correlate to the cgc/member pair for the SLB and TLB.
5. When a response is provided by the inUse scoreboard with the corresponding SLB and TLB indices, the check-in engine issues SLB and TLB directory writes (or both to the TLB for guest and host indices for a Radix translation) to obtain congruence class locks for both halves of the nested translation.
6. When locks are secured, the check-in state machine issues a request to the inUse scoreboard to clear the valid bit for the entry of the targeted ERAT index.
7. At this point, the check-in state machine must verify whether the primary and secondary indices are still in use (that is, STE maps to multiple PTEs for HPT translations or multiple guest pages map to a single host page in Radix translations). This is done by another query to the inUse scoreboard for the primary and secondary indices, respectively.
8. If the corresponding primary or secondary index is no longer in use, the check-in state machine masters a directory write to the SLB or TLB to clear the inUse flag for the cache member and to reset the cgc lock for the respective index.
9. When the query operation completes within the inUse scoreboard, the check-in state machine drives a check-in response back to the agent to signal completion of the operation. This is arbitrated through the Fabric Dout path.
10. Upon receipt of the check-in response, the agent clears the valid bit of the corresponding ERAT index that was checked in.

16.5.6.6 NMMU Invalidate State Machine Overview

The high-level steps that the NMMU performs in executing an invalidation sequence are as follows:

1. The NMMU snoops an slbie/tlbie operation command and initiates an FBC invalidate state machine, which manages the command's protection window.
2. The NMMU snoops a corresponding slbie/tlbie set command, which is forwarded out of the Fabric macro and into the corresponding CTL invalidate state machine.

3. The CTL invalidate state machine forwards the set command to the respective SLB/TLB cache pipe to determine if it is a cache hit or miss.
4. At SLB or TLB dispatch time, the CTL invalidate engine receives hit/miss information for the snoop invalidate. It is initiated due to a snoop invalidate scenario. If the snoop hits in the NMMU, the invalidate engine captures the LRU member valid vector and the inUse vector for the targeted cgc. Note that there are a total of 12 invalidation state machines in the POWER9 processor to support up to four SLBIEs and eight TLBIEs.
5. The invalidate state machine issues a request to the global inUse scoreboard to map the SLB or TLB index for the invalidation to the corresponding ERAT index, which correlates to the entry in the agent that must be torn down.
6. After a response is provided by the inUse scoreboard, the invalidate engine captures the ERAT index for the targeted SLB/TLB cache member (primary index) and the secondary index (the SLB or TLB index that is not directly being invalidated).
7. The invalidate engine drives invalidate requests to the corresponding agent for the given ERAT index.
8. When the invalidate state machine detects a response from the inclusive agent, the engine progresses to the next member bit that is set in its inUse vector and repeats the previous process by identifying the corresponding ERAT index and by shooting down the required ERAT index in the agent. This process continues until all valid members have been drained for the respective congruence class.
9. When the agent shootdowns complete, the invalidate engine (similar to a check-in engine) issues SLB and TLB directory writes to clear the valid bit for the corresponding primary cgc members in the respective caches and to update the inUse flags for the secondary indices.
10. When clean dispatch results are detected for both the primary and secondary (SLB and/or TLB) directory writes, the invalidate engine issues a request to the inUse scoreboard to clear the valid bit for the targeted entries that map to an affected ERAT index.
11. When the clear operation completes within the inUse scoreboard, the CTL invalidate state machine drives an invalidate-complete pulse to the FBC invalidate state machine to terminate the command protection window and to end the sequence.

16.6 Unit RAS Overview

16.6.1 RAS Features

- No custom circuits required for arrays in the NMMU. Numerous arrays are all C8Ts and SSAs.
- All arrays have error detection and are recoverable.
  - Fabric data buses arrive and are sent with ECC. Associated buffers/arrays contain ECC. When data is manipulated, ECC errors are checked and reported with failing syndrome. Single-bit errors are corrected.
  - Other arrays (for example, SLB/TLB directory and cache) are primarily parity-protected.
  - Exception for unprotected arrays are those that cannot cause data integrity issues (for example, the SLB and TLB LRU).
- All ECC/parity generation includes error injection points to allow for single-bit and double-bit errors.
- Fabric command, snoop, and response buses are parity-protected.
- Control checkers:

  - Interface protocol checks for the various agents to verify correctness.
  - Timeout conditions detected, primarily through state machine non-idle scenarios.
  - As Fabric master/snooper, NMMU supports a hang-recovery mechanism.
  - Illegal state machine transitions are detected and verified through BugSpray checkers.
- Error Reporting:
  - Fault Isolation Register (FIR) is used to log various errors and failure conditions.

16.6.2 NMMU Error Handling Policies

The NMMU and its attached accelerator agents form a translation complex on POWER9 processor where all parties are notified if the other unit is down and unable to service requests normally. There are two broad categories of NMMU unit behavior in the presence of errors:

- Faults that occur during the process of translating an EA mastered by an accelerator:
  - In these scenarios, the NMMU returns the status to the corresponding accelerator with an encoding that provides the reason for the fault.
  - The corresponding agent can report the status to software in a variety of ways (for example, an interrupt or stamping the status into its packet header/control block).
  - The agent reports the original EA (or gVA for Radix translations) and the failing status code to software. Software is notified by using an interrupt that causes the code to touch the failing page. The touch causes the translation to be replayed out of the core MMU, which runs into the same error as detected by the NMMU. The core MMU provides its typical diagnostics to software for fault resolution (for example, DAR, HDAR, DSISR, HDSISR).
- Errors that cause the NMMU to drive a system checkstop, which is considered catastrophic.

17. Interrupt Controller

The POWER9 interrupt controller (INT) consists of three major units: the virtualization controller (P3VC), the presentation controller (P3PC), and the POWER9 fabric bus interface common queue (P3CQ). These units work together to take triggers from interrupt sources and deliver exceptions to the appropriate processor thread. This section provides an overview of the interrupt architecture, describes the INT units and their interfaces, and also describes how they operate with the interrupt sources and software in the POWER9 infrastructure.

Note: The "P3" prefix in the unit acronym name refers to version 3 of the interrupt architecture. The previous version was referred to as version 2.

17.1 External Interrupt Virtualization Engine

The POWER9 interrupt architecture significantly reduces the interrupt code overhead/path length and improves performance compared to the previous architecture. Other advantages of this architecture over previous versions are:

- Enables direct user-level I/O device drivers:
  - Direct I/O adapter interrupts to user-level event-based branches
  - Significantly simplifies CAPI models and path length
- Enables direct user-level virtual I/O signalling:
  - Significantly simplifies scalable inter-processor/partition signalling
  - Compliments scalable virtual super-sockets
- Combines all notification mechanisms into one architecture:
  - External interrupts, including inter-processor interrupts (IPI), targeting:
    - Operating dystem (OS)
    - Hypervisor
    - Event-based branch (EBB)
  - Enables authorized signalling by:
    - I/O device
    - Platform service
    - Program at any privilege level
  - Adds routing to dispatched logical server in addition to physical thread:
    - Desired combination of LPID, VP/VT, PID, TID
  - Removes hypervisor from the path, except:
    - When required to dispatch a logical server
    - To handle extreme scalability
    - To handle corner cases in page migration

17.2 High-Level Block Diagram

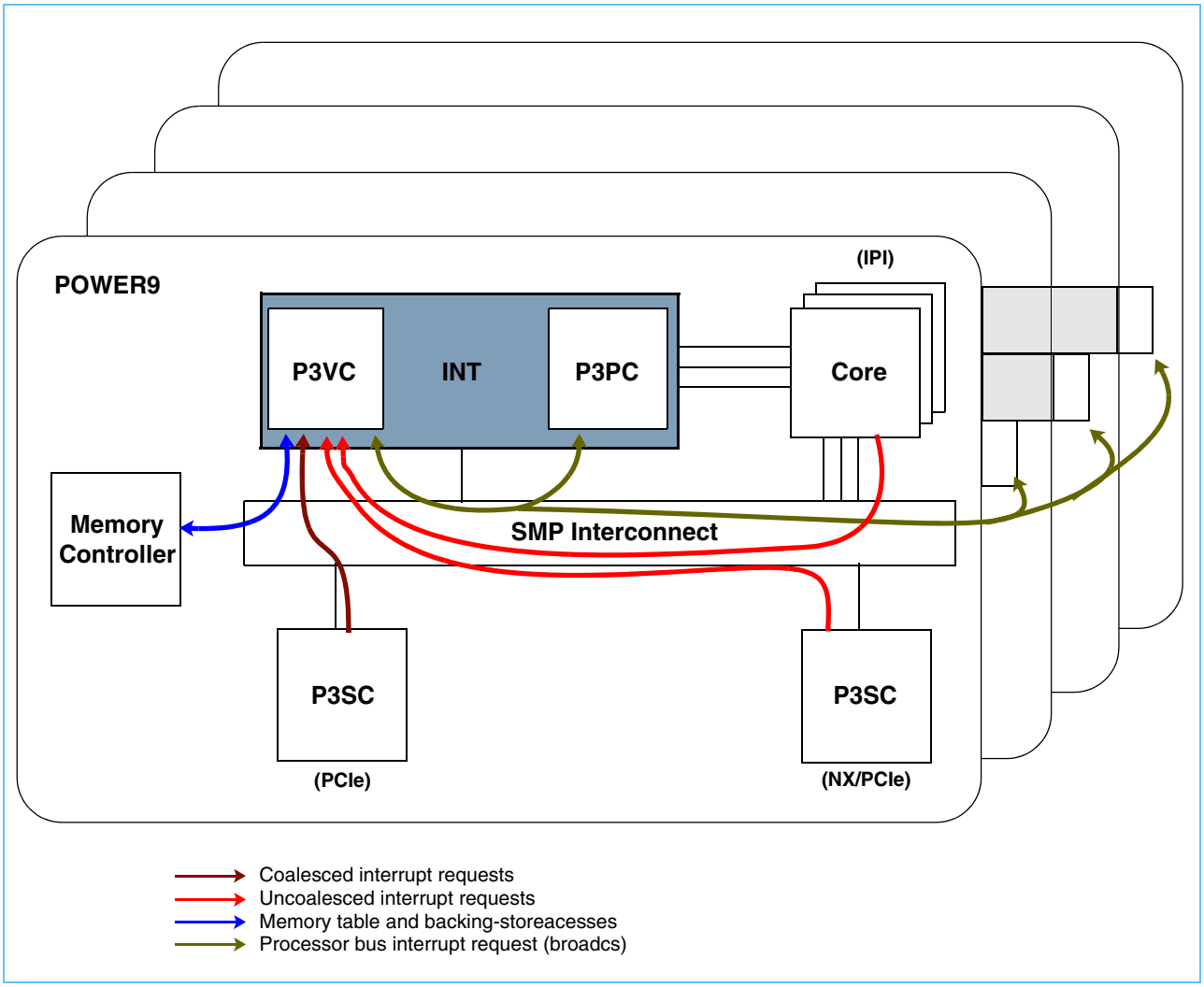
The high-level diagram, *Figure 17-1* on page 259, depicts the conceptual interaction among sources and the controller blocks in interrupt signalling and notification. The individual elements are interconnected and communicate via the POWER9 Fabric bus.

The P3VC receives notification triggers from interrupt source controllers (P3SCs) via a POWER9 Fabric bus store operation (for example, cache-inhibited write: ci_wr). It processes the notification using information contained in the event assignment entry (EAE) that is located in main memory and associated with the specific trigger. This processing might include updating an event queue entry, and then forwarding the notification to the P3PC, which signals an exception to one of the processor threads. The P3VC also handles notification redistribution if a state change to the assigned processor thread preclude it from handling the interrupt, or notification escalation if there is no processor thread that is currently capable of handling the interrupt.

The P3PC has exception notification wires connected to individual processor threads. Three wires exist for each thread. The processor thread uses one exception wire to generate hypervisor interrupts, another to generate operating-system interrupts, and a third wire to generate an event-based branch. Associated with each of the exception-notification wires in the P3PC is prioritization and exception-queueing logic that prevents less favored events from pre-empting more favored ones or from loss due to dropping an event. Associated with each of the exception notification wires is one or more logical server numbers stored in CAM-like lines. This structure is also referred to as the thread context (TCTXT). These logical server numbers identify which software entities are currently dispatched on the specific physical processor thread. When the P3VC issues Fabric bus operations to route an event notification, these CAM-like lines are searched to identify candidate processor threads. In addition to the CAM-like lines, priority and exception-queuing logic mentioned previously, each interrupt-generating exception has logic to track how much interrupt work has been handled by the associated processor thread. This information is used to evenly distribute interrupt processing load among the candidates.

The P3CQ serves as the POWER9 fabric bus interface controller between the interrupt logic and the rest of the POWER9 chip. This unit is responsible for sequencing the appropriate fabric bus protocol when the interrupt controller drives or receives commands. It performs compares to determine if the interrupt controller is the destination of a command (for example, a store operation used for an interrupt trigger). It is also responsible for driving the fabric bus histogram, poll, and assign commands to find the correct presentation controller for an interrupt trigger. Another key P3CQ function is sending and receiving the AIB interface to the virtualization and presentation controllers.

Figure 17-1. Interrupt Presentation Interaction

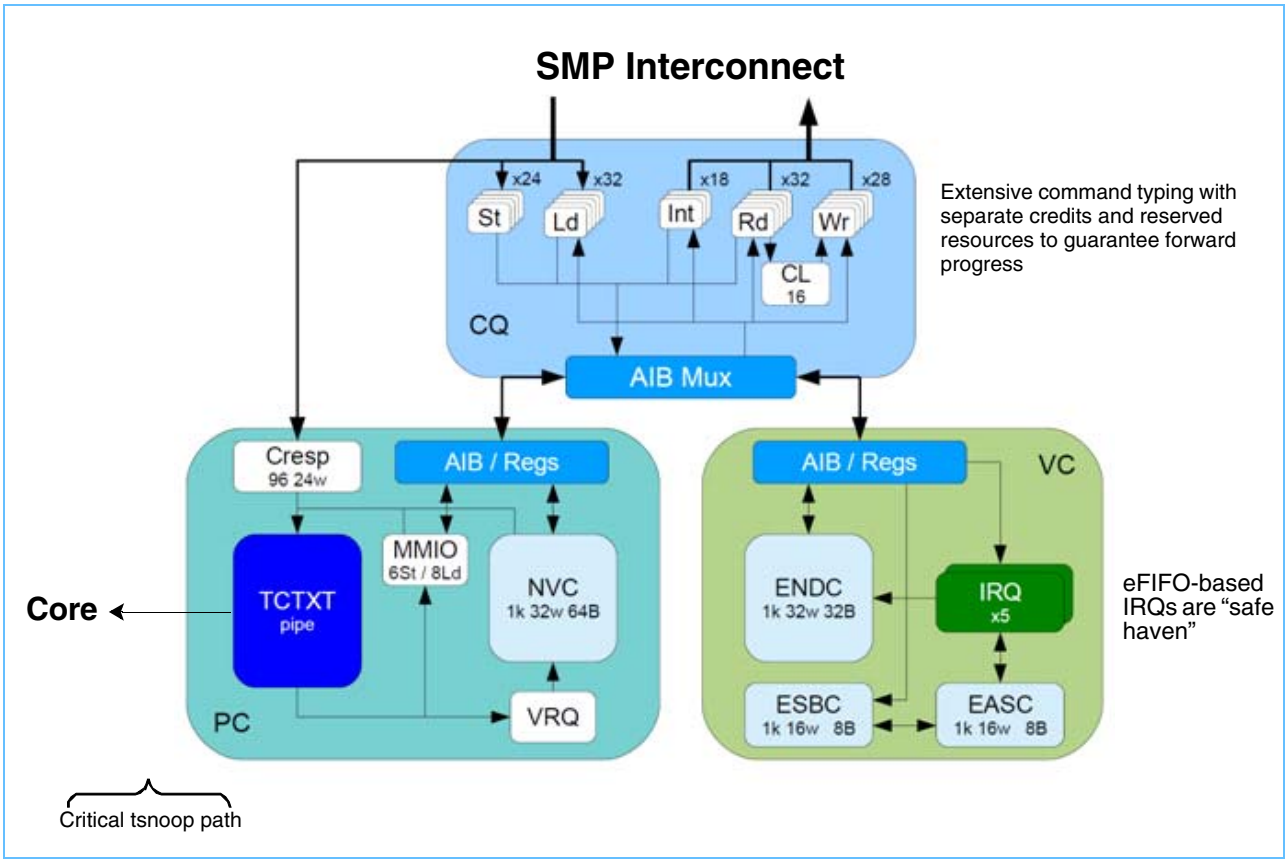


17.3 INT Unit Overview

Figure 17-2 shows the detailed structure and implementation of the three major units. Within each unit, the major sub-blocks and interfaces are outlined. This section provides more description and details on the three major units.

Note: The INT unit adheres to the POWER9 RAS requirements with parity on latches and ECC on SRAMs and major interfaces.

Figure 17-2. Interrupt Controller Microarchitecture



17.3.1 P3 Common Queue (P3CQ)

The interrupt controller common queue (CQ) is the bridge between the fabric bus and the presentation and virtualization controllers.

The CQ is connected to the PC and VC through two AIB ports. The AIB arbitration and muxing exists within the CQ logic. The AIB interface logic also allows PC-VC communication for local accesses.

The P3CQ features/functions are:

- CI load/store machines, each machine handles one processor bus operation at a time.
- Read/write/interrupt machines, drive DMA read/write, CI read/write, and interrupt commands.
- Block routing and tracking for storing target addresses for block CI operations and scope tracking for interrupt commands.
- BARs for scope generation based on the fixed address mapping scheme used for POWER9.
- Migration registers for memory migration secondary BAR information.
- One interrupt controller PB BAR register.
- One thread management PB BAR register.
- Four configurable PB BAR Registers (1 spare/reconfiguration)
- DMA scope generation using BAR registers
- Block-based CI address generation.
- Snoop directory with 16 entries for cache-line protection during pull-push mode.
- 16 cache-line data buffer slots for push-pull operation.

17.3.2 P3 Virtualization Controller (P3VC)

The virtualization controller is the main interrupt processing unit. It takes triggers from the fabric bus and processes them using information found in the corresponding event queue entry.

The P3VC main features/functions are:

- Five interrupt trigger queues (IPI, hardware, first-level escalation, second-level escalation, redistribution) that can be extended in main memory
- 16-way set-associative cache for 1K event assignment entries (EAEs)
- 16-way set-associative cache for 32K states (1K groups of states)
- 32-way set-associative cache for 1K event notification descriptors (ENDs)
- Support for cache-scrubbing and cache-watch commands
- Support for **sync** commands on all interrupt queues in the event notification descriptor cache (ENDC)
- Up to 16 blocks owned by P3VC

The P3VC main processing flow is:

- Interrupt triggers come in as CI store operations. Depending on the interrupt source, it is stored into one of the five interrupt queues.
- The queue is read and the interrupt request is either dropped if the mask bit is set, directly sent to the ENDC block if no event assignment structure cache (EASC) lookup and state bit check are required (a

case where the trigger is an event-queue trigger and therefore holds EQ information), or an EASC lookup is performed.

- The EASC does an EAE lookup to get EQ trigger information.
- State bit check is executed if the interrupt controller does the interrupt coalescing. If the current state is reset, the state is switched to pending. If the current state is Pending, the state is switched to queued.
- The EAE is returned to the IRQ subunit. The mask bit is asserted in the following case: EAE valid bit = '0', or EAE mask bit = '1', or SB check was required, and PQ state was not reset; which means the interrupt was already presented or the SB entry is disabled.
- If the EASC lookup response has the mask bit set, the interrupt process ends. Otherwise, EQ trigger is forwarded to the EQC block for processing. At this point, the interrupt trigger becomes an Event Queue trigger.
- The ENDC block checks that it owns the EQ block. If the current interrupt controller does not own the EQ block, the EQ trigger is forwarded to the owner using a CI store operation. The target queue is configurable. Otherwise, the ENDC does an END cache lookup to get the corresponding EQ descriptor.
- Depending on the END content, the ENDC can:
  - Do nothing
  - Post an event in the event queue and increment the EQ pointer
  - Execute an SB check and update
  - Generate an interrupt request
  - Increment per the priority backlog counter in the Notification Virtual Target (NVT)
  - Set per the priority pending bit in NVT
  - Issue an EOI command
- Depending on the interrupt response, the ENDC can:
  - End the interrupt process
  - Increment per the priority backlog counter in NVT
  - Set per the priority pending bit in NVT
  - Escalate
  - Issue an EOI command

In addition to EQ triggers processing, P3VC processes end-of-interrupts. EOI comes as a CI load operation. When EOI is received by either the event state buffer cache (ESBC) or the ENDC, the corresponding state bits are updated according to the PQ state bit state machine definition.

17.3.3 P3 Presentation Controller (P3PC)

The presentation controller unit holds state information regarding which software entity is dispatched on each processor thread. The P3PC is responsible for responding to the fabric bus interrupt histogram, poll, and assign commands and participating in selecting the best thread for the interrupt. The P3PC drives exception wires connected to the individual processor threads. It also maintains the logic server structure cache which contains backlog counters and virtual processor control and pending bits.

The P3PC main features/functions are:

- CAM pipeline with 96 thread contexts (24 cores × 4 threads)
- Local load/store machines (eight load and six store per type)
- LSI logic for creating and handling LSI interrupts

- Notification virtual descriptor cache (NVC) for caching notification virtual target entries (NVTs) and processing NVT requests from the CAM pipeline
  - Single snoop bus 3 used for interrupt commands
  - EBB, operating system, and hypervisor-level interrupt line per thread
  - Works on NVT data
  - Can own up to 16 blocks and the associated NVTs
  - Push-pull mode selectable on a per block basis for the NVC

The major subunits of P3PC are:

- Thread context (TCTXT) Pipe: Contains interrupt-related processor physical thread information (thread context) used for routing and processor bus responses. Has a 'direct' fabric bus interface through the P3CQ and drives exception wires to each processor thread.
- Notification virtual descriptor cache (NVC): Responsible for caching notification virtual descriptor entries used for escalation, backlog, and redistribution processing.
- AIB interface (Rx/Tx): Primary communication interface between P3PC and P3CQ/VC for register, CAM pipe, and NVC cache access.
- LSI: Responsible for handling level-sensitive interrupts from the pervasive. The LSI contains a direct interface with TP.

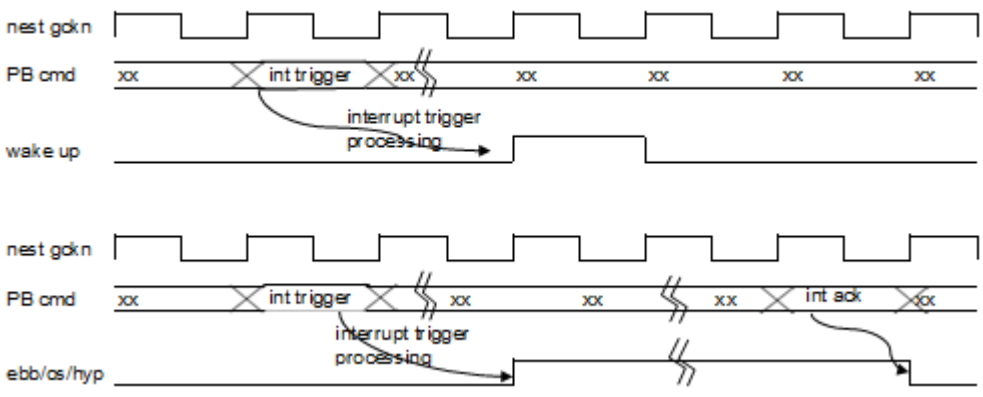
P3PC has three exception signals that it sends or presents to each processor thread:

- Hypervisor exception
- Operating system exception
- Event based branch (EBB)

The P3PC also sends the Msgsend signal to each processor thread. There are a total of 384 interrupt wires on a POWER9 chip (24 cores × 4 threads/core × 4 exception wires).

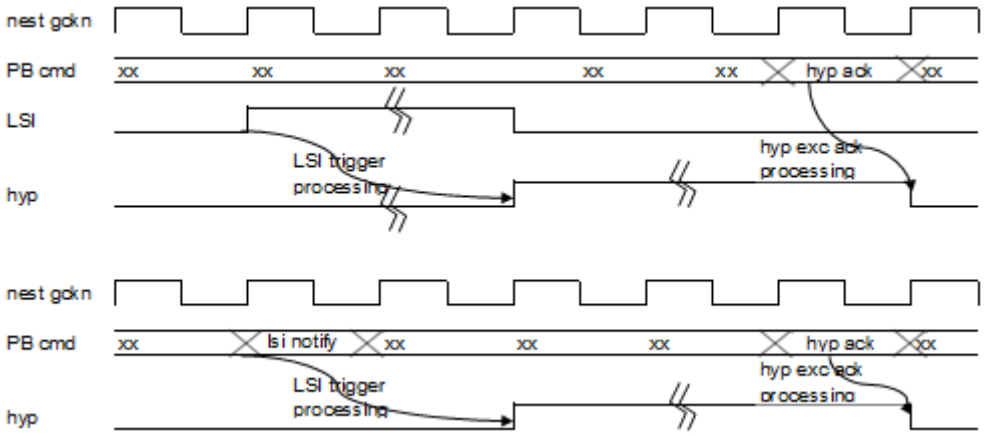
The msgsend exception line is only one pulse wide, the other exception lines are held HIGH until an interrupt acknowledge command is received from the core.

Figure 17-3. Exception Wire Activation Example



In addition to the processor bus interrupt commands: histogram, poll, and assign, the interrupts can also be triggered by a level sensitive interrupt (LSI) event trigger from the pervasive unit. The LSI trigger can be communicated to the presentation engine via the LSI hardwire or a CI store operation to the LSI notification register inside the presentation engine.

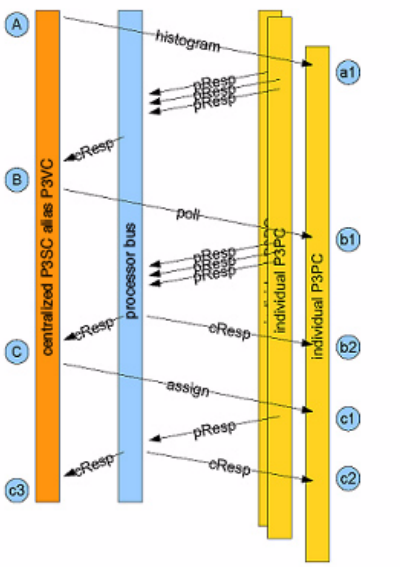
Figure 17-4. LSI Activation Example



17.4 Fabric Bus Interrupt Command

A P3PC receives interrupt fabric bus commands (histogram, poll, and assign) and generates responses based on the contents of the CAM lines and other state information in the unit. *Figure 17-5* through *Figure 17-7* on page 265 show the fabric bus command and response sequences for the P3PC.

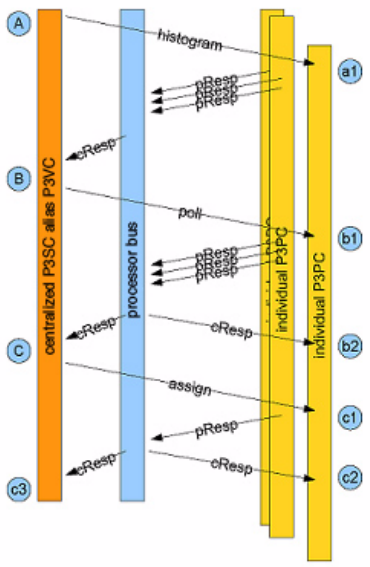
Figure 17-5. Transaction Diagram for Histogram, Poll, and Assign (Part 1 of 3)



The notification to the best server is done in three steps:

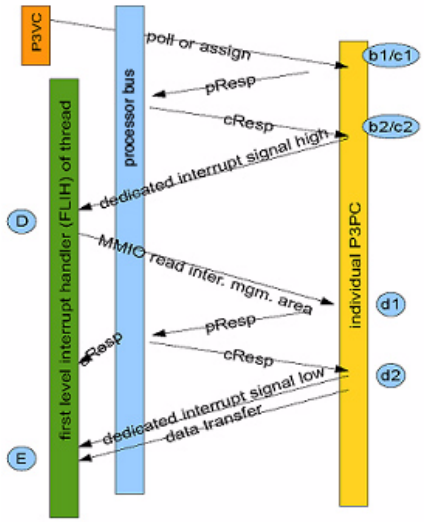
- A P3VC issues a Histogram command to all P3PC in the system or group.
- a1 Each P3PC sends a pResp (partial response) containing the largest age field of its matching servers.
- B The processor bus returns the cResp (combined response) indicating the largest age fields among all P3PCs.
  Based on that, P3VC issues a Poll command supplying the received age field as a parameter.
- b1 Each P3PC sends a pResp indicating a match (including age).
- b2 The processor bus returns a cResp, which indicates if note 1 or multiple P3PC reported a match in pResp and the cResp indicates one match. Then P3PC notifies its attached threads (see part 3).

Figure 17-6. Transaction Diagram for Histogram, Poll, and Assign (Part 2 of 3)



- C If the cResp indicates multiple matches, the P3VC issues an Assign command to a specific P3PC.
  If the cResp indicates no match, the P3VC repeats the whole process (starting with the histogram).
- c1 The addressed P3PC indicates in the pResp if there is a match.
- c2 If a P3PC reported a match in pResp, it will on arrival of the cResp notify its attached threads (see part 3).
- c3 If the cResp indicates no match, P3VC executes its escalation process.

Figure 17-7. Transaction Diagram for Histogram, Poll, and Assign (Part 3 of 3)



- b2/c2 Either if the poll is already indicating that only one P3PC has a match or after the dedicated poll with match, the thread attached to the CAM line is notified via assert of the dedicated interrupt signal.
- D As an acknowledgement, the first-level interrupt handler performs an MMIO read of the interrupt management area (4 CAM lines, that contribute to the interrupt signal).
- d2 Upon arrival of the cResp in the data phase is the the dedicated exception line de-asserted and the interrupt management area information (indicating the server to be interrupted) is transferred.
- E End of interrupt operation.

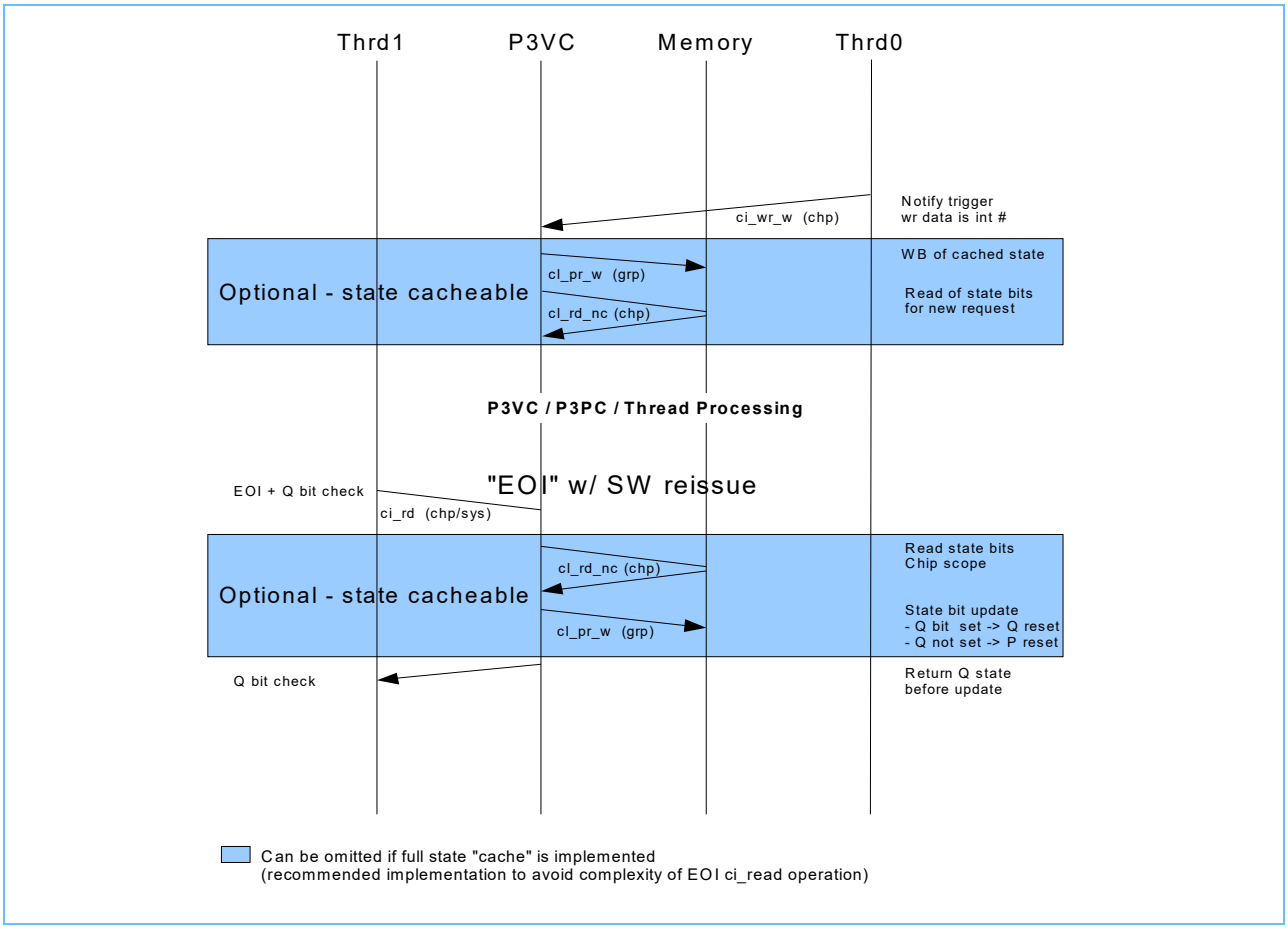
17.5 Interrupt Processing Flow Examples

Figure 17-8 through *Figure 17-11* on page 269 depict interrupt processing examples. *Figure 17-8*, *Figure 17-9* on page 267, and *Figure 17-10* on page 268 show trigger operations and reset for individual interrupt triggers. *Figure 17-11* shows event queue enqueues and presentation to the cores common to all *Figures 17-8 through 17-10*.

17.5.1 Inter-Processor Interrupts Example

Figure 17-8 is an example of the inter-processor interrupt (IPI).

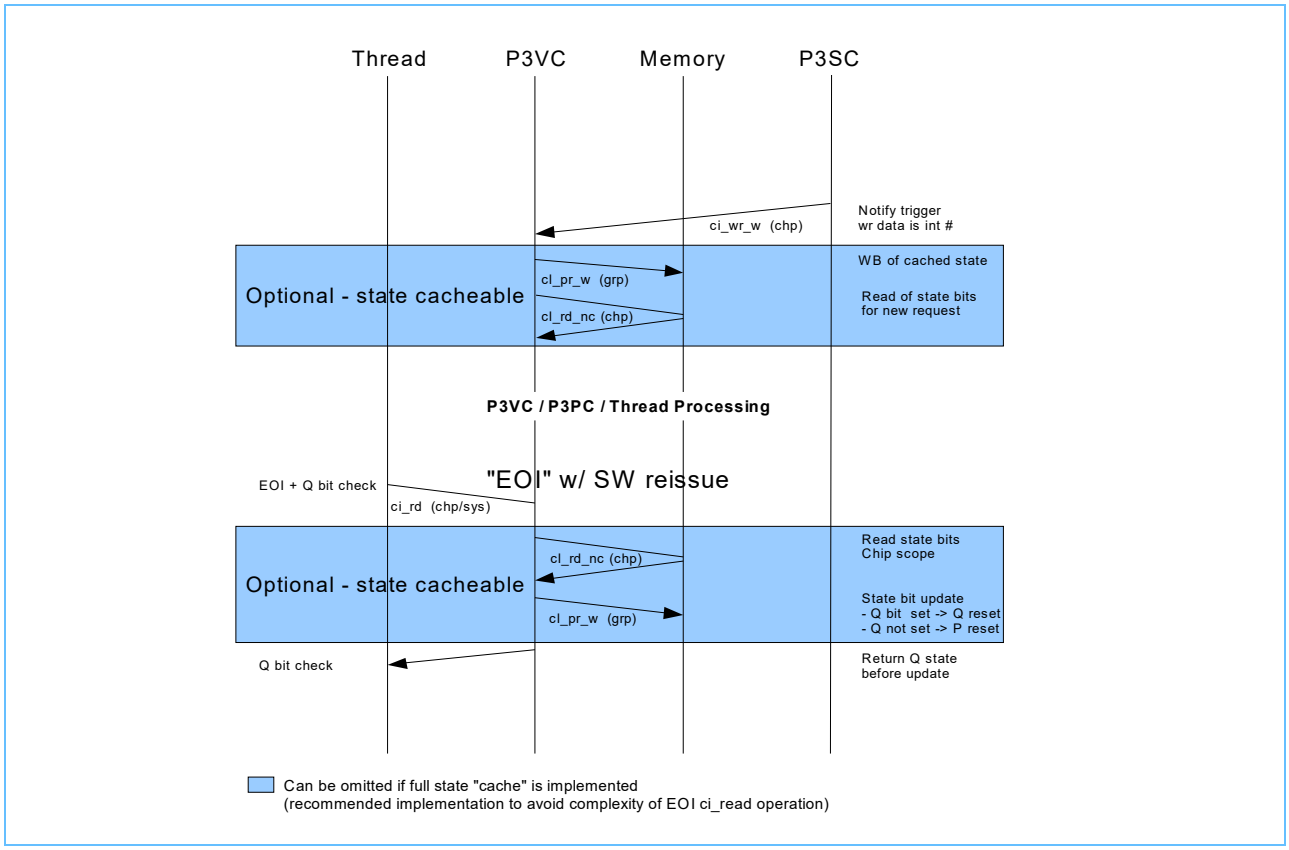
Figure 17-8. Inter-Processor Interrupts (IPI) Example



17.5.2 Hardware Interrupt with State Bit Check in P3VC

Figure 17-9 is an example of a hardware interrupt with the State bit check in P3VC.

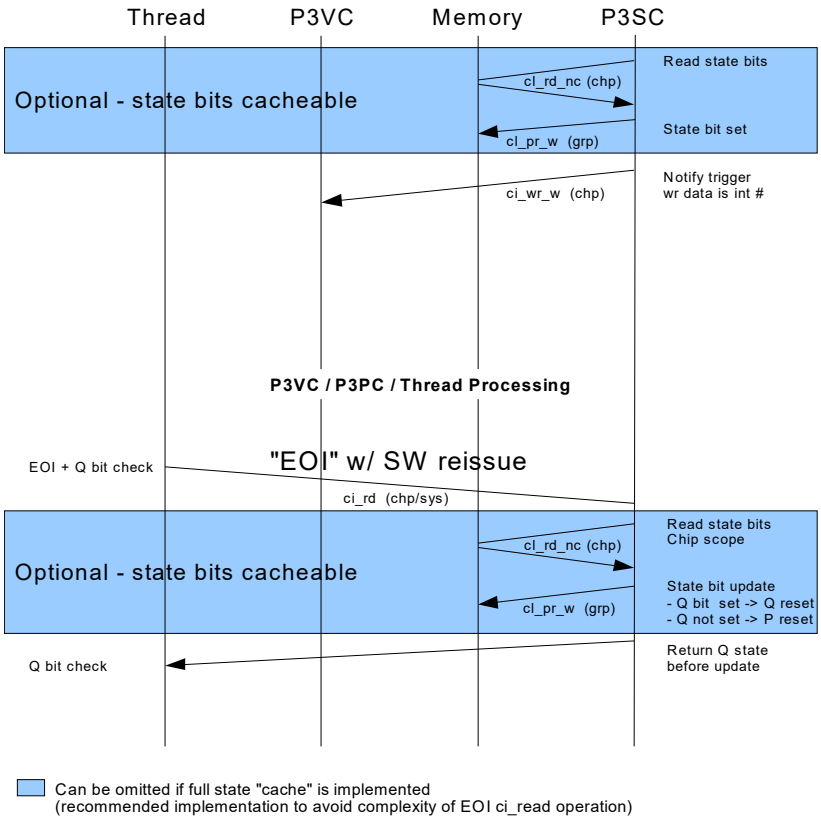
Figure 17-9. Hardware P3SC Interrupt Trigger and Completion (State Bit Check in VC)



17.5.3 Hardware Interrupt with State Bit Check in P3SC

Figure 17-10 is an example of a hardware P3SC interrupt trigger and completion.

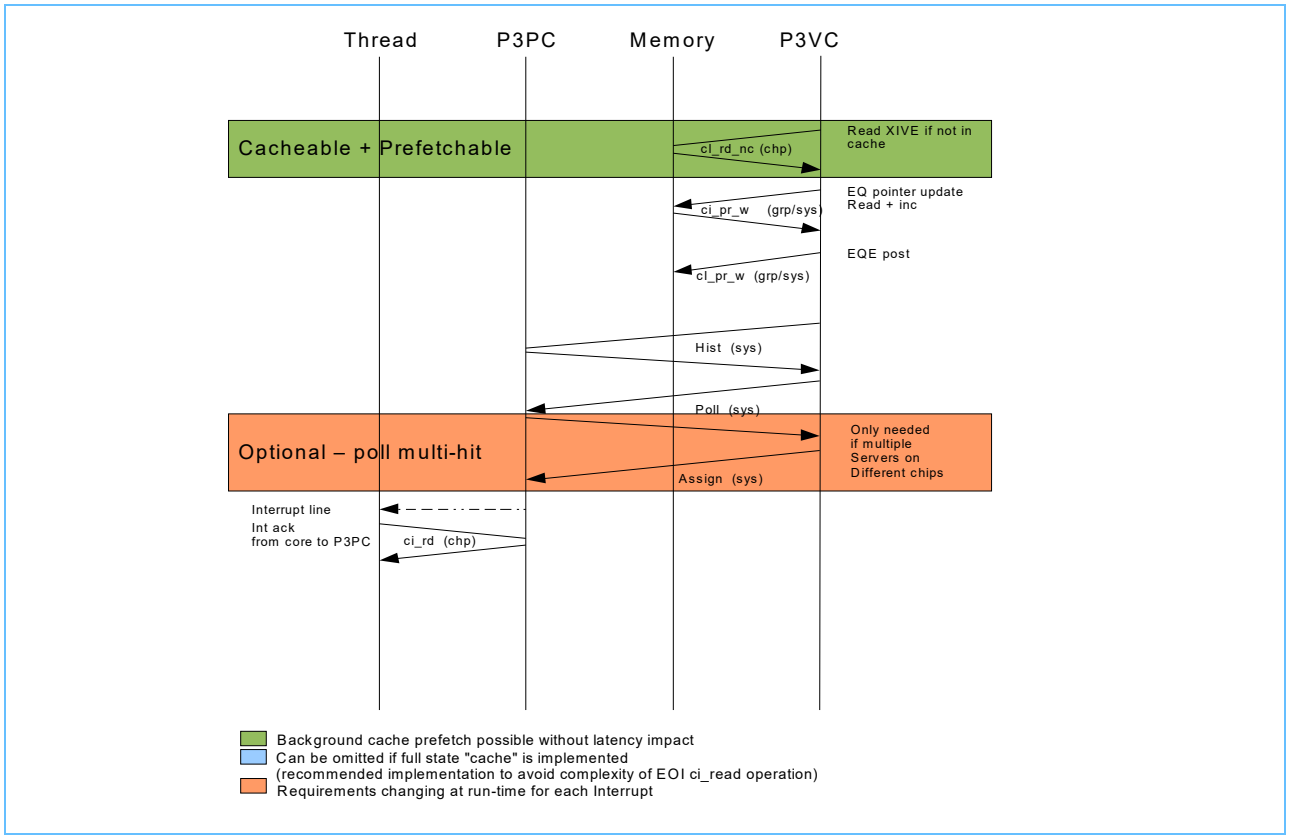
Figure 17-10. Hardware P3SC Interrupt Trigger and Completion (State Bit Check in SC)



17.5.4 P3VC and P3PC Basic Interrupt Handling

Figure 17-11 is an example of P3VC and P3PC basic interrupt handling.

Figure 17-11. Basic Interrupt Handling



17.5.5 Message Send (Msgsend) and Wakeup

The interrupt controller logic supports the internal Fabric bus msgsend command. The P3CQ snoops the SMP Fabric for msgsend commands. If it determines an address match, it asserts lpc_ack and passes the command on to the thread context (TCTXT) portion of the P3PC. This logic then decodes the appropriate threads and activates a wire to the appropriate threads of the POWER9 chip.

The interrupt controller logic does not support the wakeup function.

18. PCI Express Controller

The PCIe Express controller (PEC) provides PCIe Gen4 root-complex ports to connect to an adapter slot or as a link to a PCIe switch. It acts as a PCIe host bridge (PHB) from the internal, coherent SMP interconnect (also known as the processor bus) to the PCIe I/O.

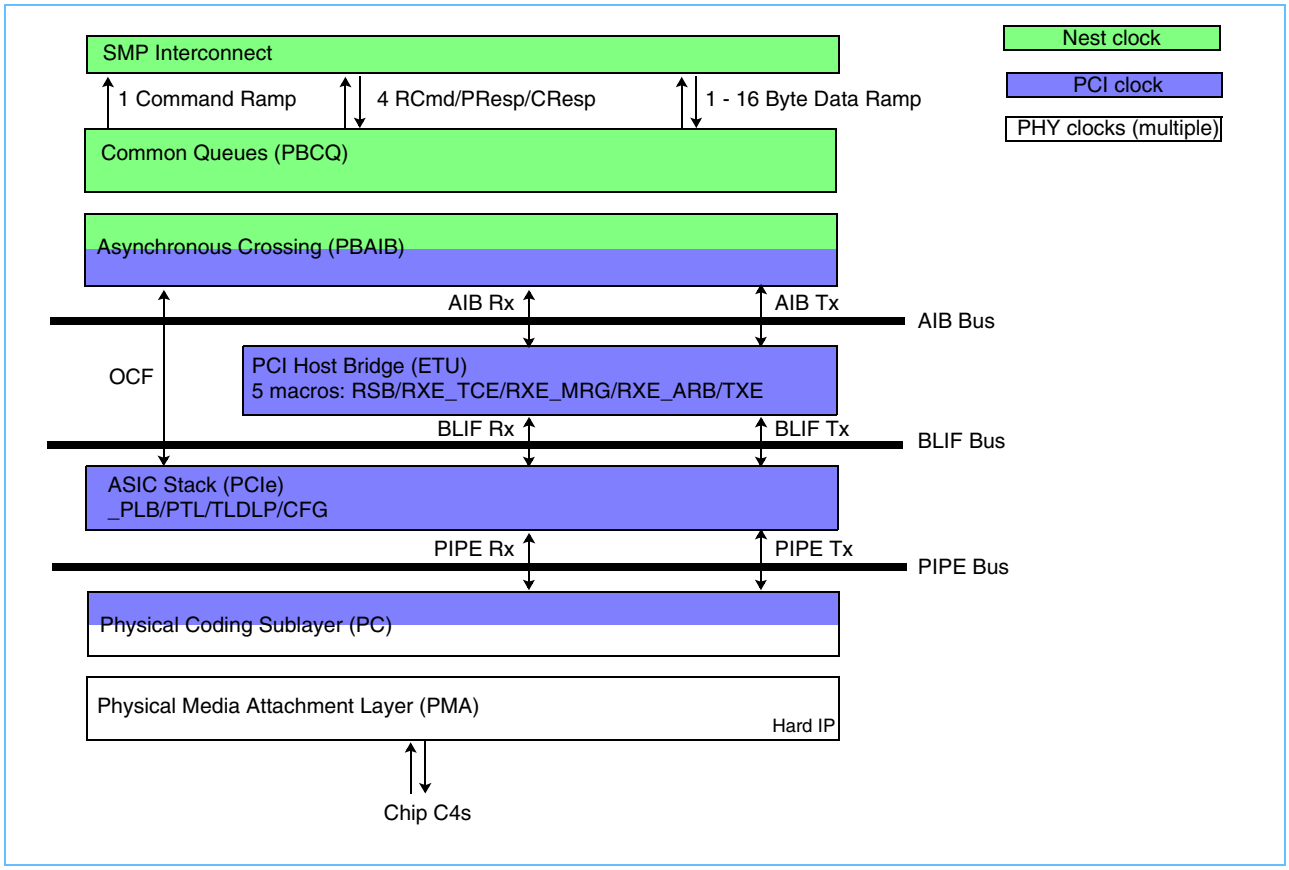
18.1 Overview

The PEC is composed of six major building blocks:

- Processor bus common queue (PBCQ) logic
- Processor bus to AIB interface (PBAIB)
- Express transaction unit (ETU)
- PCIe ASIC building blocks (PCIASIC)
- Physical coding sublayer (PCS)
- Physical media access (PMA)

Figure 18-1 shows an overview of the major blocks and defined interfaces.

Figure 18-1. High-Level Block Diagram



18.1.1 Processor Bus Common Queues

The processor bus common queue (PBCQ) logic is responsible for managing the transactions on the coherent processor/cache fabric, the SMP interconnect.

Key features of the PBCQ are as follows:

- CAPI support
- Tunnelled operations
  - Atomics
  - AS_Notify support
- Inbound DMA capability
  - Supports 64 DMA read transactions (128 on PEC0) on the SMP interconnect. DMA read transactions are sourced from non-posted read transactions from the PCIe.
  - Supports 32 DMA write transactions on the SMP interconnect. DMA write transactions are sourced from write transactions posted from the PCIe.
  - Peer-to-peer write capability.
- Tunnelled operations
  - Atomics: Atomic transactions are sourced from posted write transactions on the PCIe and, if necessary, return data back to the PCIe using an MMIO store.
  - AS_Notify: Quick method to communicate with the core.
  - CAPI support.
- Outbound MMIO capability
  - Two Base Address Registers (BARs) for external MMIO address ranges
  - 16 MMIO stores
  - 16 MMIO loads
- The ability to share resources with more than one PHB stack

18.1.2 Processor Bus AIB Interface

The PBAIB logic provides an asynchronous boundary crossing between the PBCQ and the AIB 2.0 interface.

18.1.3 Express Transaction Unit

The ETU is responsible for address translation, interrupt management, and error isolation.

Key features of the ETU (×8 or ×4 lane versions) are as follows:

- 512 KB (256 KB) partitionable endpoints
- 1 KB (512 KB) 4-way set-associative translation cache
- 4K (2K) MSI interrupts supported
- Eight LSI interrupts supported

18.1.4 PCIe ASIC Intellectual Property

The PCIe ASIC building block is composed of the packet buffer layer (PBL), the packet transaction layer (PTL), the transaction and data link layer (TLDLP), and the PCIe Configuration Register core (CFG). These blocks implement the PCIe transaction and data link layers.

18.1.5 Physical Coding Sublayer

The PCS manages the low-level networking protocol and signaling between the physical media and the higher-level link protocol layer across the PIPE interface. The 16 lanes of the PCS can be bifurcated into two ×8 lanes or trifurcated into one ×8 and two ×4 lanes.

18.1.6 Physical Media Access

The PMA provides the SERDES and analog protocols necessary to connect to the chip C4s. It also provides the PLLs used to drive the PCI clock grid.

18.2 POWER9 Configurations

The POWER9 chip has three PCIe controllers of 16 lanes each for a total of 48 lanes of PCIe Gen4 I/O. The three PECs can support 4 - 6 PCIe stacks and can be configured as follows:

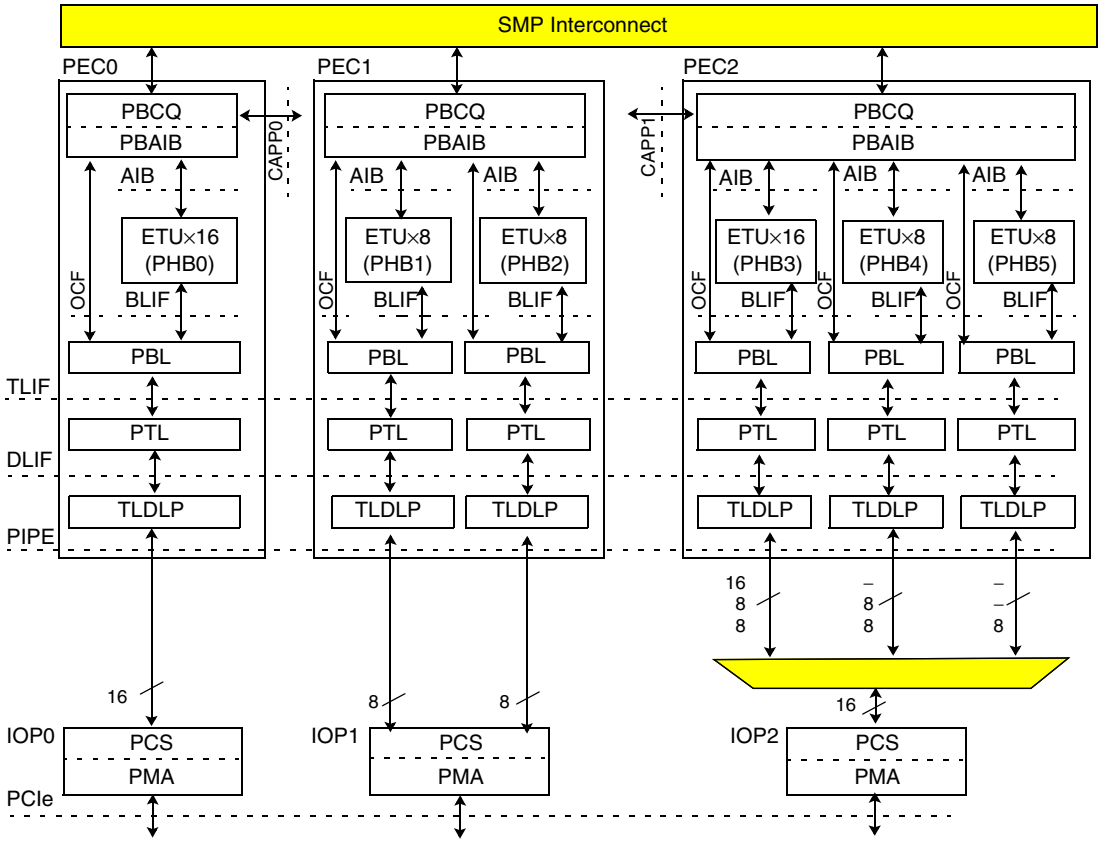
- PEC0: One ×16 lanes
- PEC1: Two ×8 lanes (bifurcation)
- PEC2: One ×16 lanes, two ×8 lanes (bifurcation), or one ×8 and two ×4 lanes (trifurcation)

Each grouping of lanes is called a stack and each stack has dedicated ETU and PCIe blocks. Each set of 16 lanes have only one PBAIB and PBCQ pair to interface to the SMP interconnect. The resources of the PBCQ are shared between the stacks that it services.

The logic in the nest clock domain is designed to run at the frequency of 2 GHz. The PCI clock domain runs at a frequency asynchronous to the nest and also at 2 GHz, with some logic running slower based on the PCI link training(1 GHz Gen3, 500 MHz Gen 2, 250 MHz Gen 1).

Within a stack grouping, the lanes can be swapped to facilitate board wiring.

Figure 18-2. POWER9 PCIe High-Level Diagram



18.3 Reliability, availability, and serviceability (RAS)

18.3.1 Bit-Level RAS

- End-to-end data protection from the processor bus ECC to the PCI packet LCRC/ECRC
- Arrays have SEC/DED ECC
- Register files have parity (some have SEC/DED)
- Support all processor bus parity/ECC
- Major control registers have parity protection

18.3.2 Enhanced Error Handling (EEH)

If an error can be isolated to an endpoint, this endpoint is blocked from introducing new transactions until the error can be resolved.

18.3.3 Freeze Mode

An error that requires a reset of a stack enters freeze mode. Freeze mode blocks all new transactions to and from the stack. Outstanding operations on the SMP interconnect run to completion, marking data as bad if required. Reset and initialization can be performed on the stack without a checkstop of the chip. A freeze on a stack does not affect the actions of another stack even if they share a PEC.

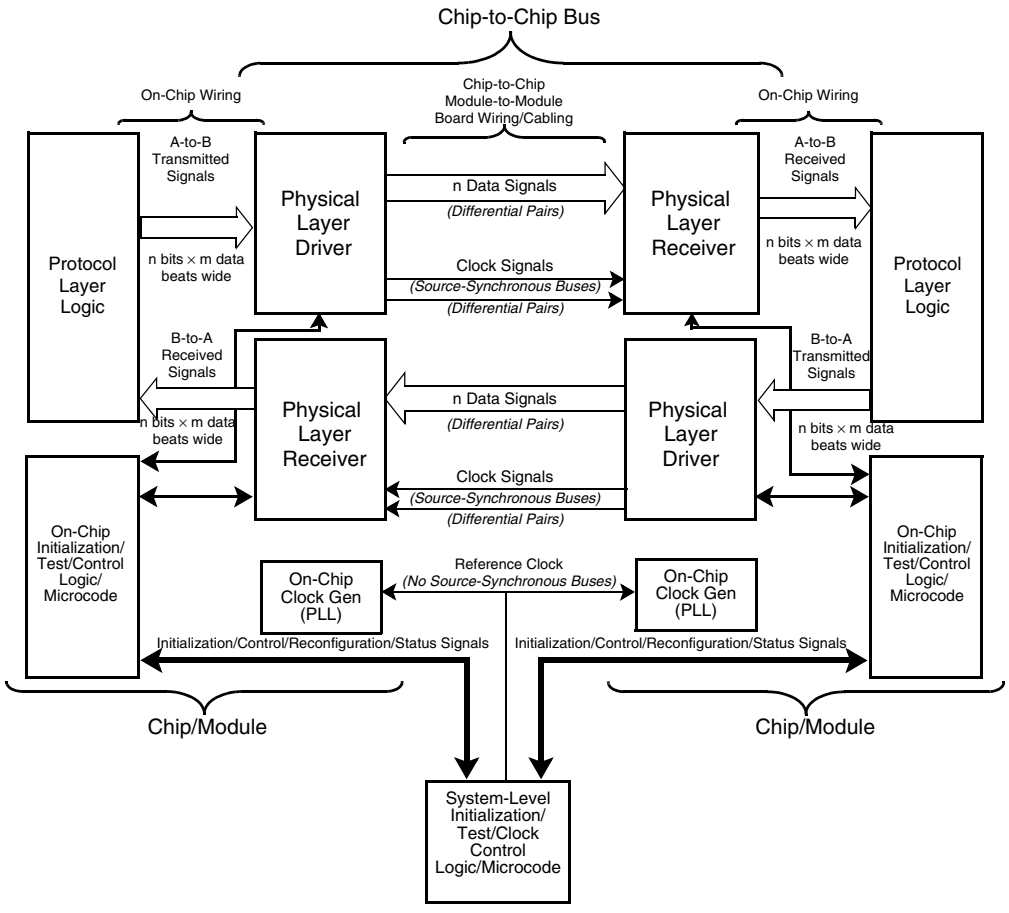
19. Elastic Differential Interface Plus

This section describes the Elastic Differential Interface Plus at 16 Gbps. *Figure 19-1* shows a system context-level diagram of a general bus system that connects two chips "A" and "B" using two uni-directional buses. Also shown is a separation between the core functions of the chip, the bus interface protocol layer, and the bus physical layer. This section gives an overview of the physical layer and on-chip initialization provided by the POWER9 processor.

The POWER9 Elastic Differential Interface Plus supports the following types of driver and receivers:

- X-bus interface: high-speed differential at 16 Gbps for chip-to-chip interconnect

Figure 19-1. System-Level I/O Interface



19.1 Elastic Interface Features

The supported features are summarized in *Table 19-1*. Requirements are defined relative to the operational-mode definitions.

Table 19-1. Interface Operational Mode Definitions

Mode Name	Definition
Initialization	The act of aligning and locking the data eye and bit lanes plus additional deltas relative to re-alignment and re-locking.
Functional	Passing workload data and maintaining signal integrity post-initialization.
Power Saving	All related capabilities for minimizing unused and idle lane power consumption.
Test	Capabilities related to hardware manufacturability.
Diagnostic	Bringup lab characterization of interface performance capabilities.

Initialization mode specifications are as follows:

- Protocols: 5-stage custom training states
- Frame alignment: Within clock group lane alignment only via initialization deskew
- Frequency: 16 Gbps
- Software: FSP1 service processor via GFW
- Spare-lane detect: Data failover (two signals total per bus/port)
- Workaround methods: Mostly-to-all initialization steps are software controllable
- Supported analog calibration methods: See *Section 19.2 Driver Features* on page 281 and *Section 19.3 Receiver Features* on page 281.

In addition to the initialization specifications, the functional mode specifications are as follows:

- Frequency: 16 Gbps
- Serialization ratio: 8:1
- Scrambling: Full scrambling is enabled during initialization
- Dynamic lane repair during run-time

Power-saving mode specifications are as follows:

- Power-saving mode is supported (light power down with fast wakeup).
- Frequencies: No special frequencies are required.
- Software: FSP1 service processor or host code via GFW; software maintains system status of spares and modes.
- Diagnostic/unused and spare lane logic: Lane control via clock off at LCB, and/or software controlled.
- IDDQ following MPG design rules.
- Analog control: Supported driver amplitudes and receiver channel-level detection

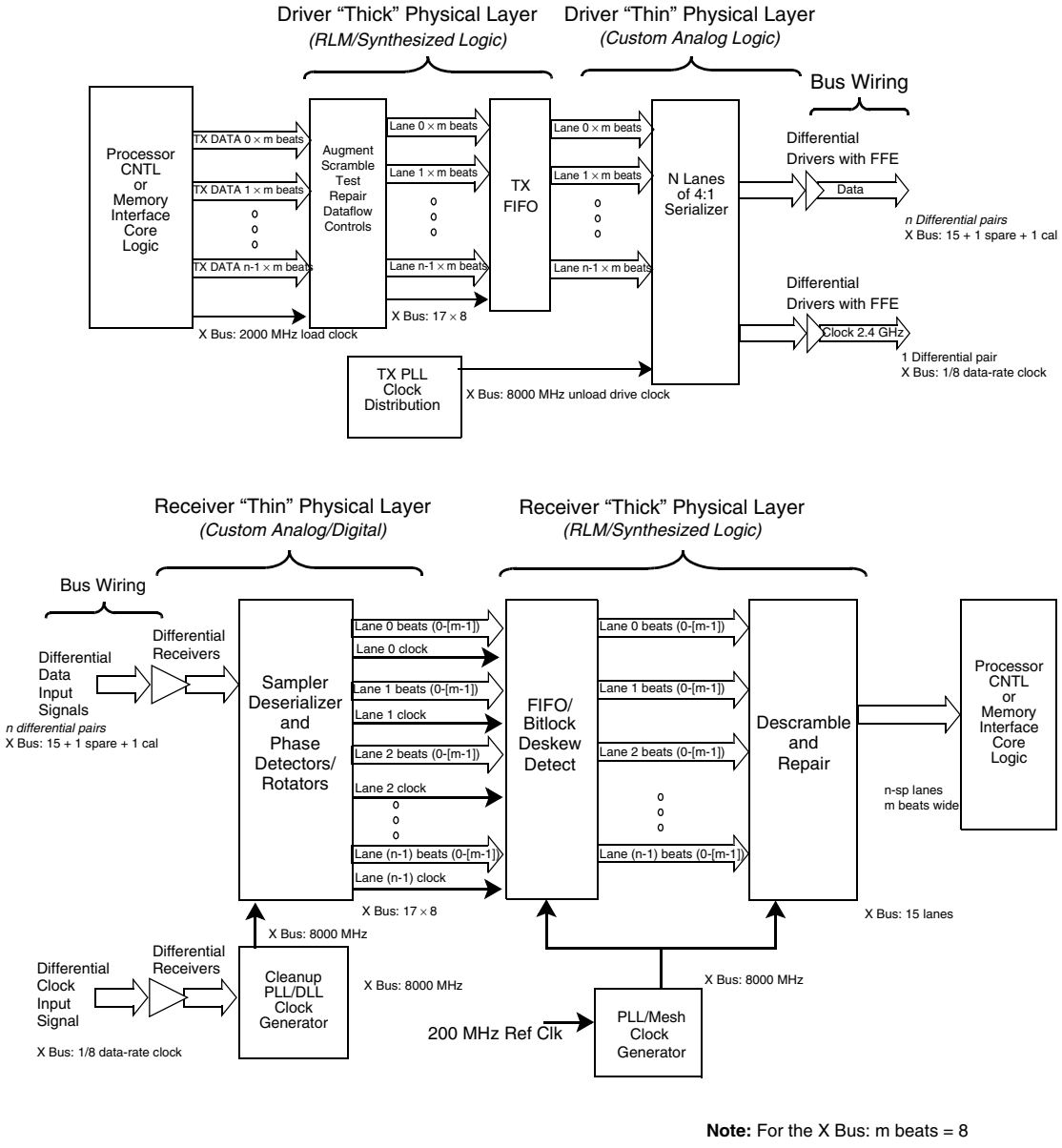
Test mode specifications are as follows:

- Frequency: 16 Gbps subject to PLL range limitations and a slow test frequency of 200 MHz.
- LBIST: AC and DC.
- Hardware-BIST: Built-in and independent TX and RX PHYBIST modes (see diagnostics list).
- PHYBIST: Real-time pattern generation test based on reused logic at product frequencies + margin.
- Wiretest: PHYs require support of 1149.x features for ASICs commonality. Also quick power-on diagnostics are supported through initialization.
- LSSD: Scan testing per MPG methodology.
- Software and test controls.

Diagnostics mode specifications are as follows:

- TX and RX internal PHYBIST
- Data sample observability
- SCOM control of all hardware functionality
- Per-bit receiver/sampler offset margining
- Phase rotator values read/write/bias
- Programmable I_{REF}
- Programmable phase detector loop sensitivity
- Programmable DFE parameter loop sensitivity
- Analog net sniffer
- Per-bit power down
- Per-bit quiesce
- Scramble disable
- TX amplitude adjustment

Figure 19-2. Top-Level Interface Block Diagram



Note: For the X Bus: m beats = 8

19.2 Driver Features

The driver features are summarized as follows:

- 16 Gbps with 8:1 serializer.
- Full-rate SST driver.
- Selectable 8:1 serializer with pre-cursor FFE.
- Rpre up to 1.30.
- Selectable AC boost: analog post-cursor FFE.
- Set and forget impedance calibrator.
- Drive amplitude reduction (margining) up to 50%. For characterization only, not mission mode.
- BIST error detector for at-speed loopback testing.
- Shared test pin mode. Differential driver output only.
- Time domain reflectometer (TDR).

19.3 Receiver Features

The receiver features are summarized as follows:

- Rx clock macro with PLL
  - Same I/O specifications as the POWER8 processor: 2.0 - 2.4 GHz bus clock range
  - Programmable feedback divider for POWER8 Memory Buffer backward compatibility
- Rx data mac
  - Each data bit with a single data path (single bank) using shadow lane protocol for calibration
  - Long-tail equalizer (LTE) for improved eye margins on lossiest channels
  - Continuous time linear equalizer (CTLE) with 12 dB of peaking range, 6 dB of gain range
  - CTLE applies common mode (differential zero) for DAC calibrations
  - 12-Tap DFE with current integrating summer. Modes: no-DFE, DFE1, DFE12)
  - 16 Gbps with 1:8 deserialization mode
  - Cross coupled PRBS streams for RX BIST testing

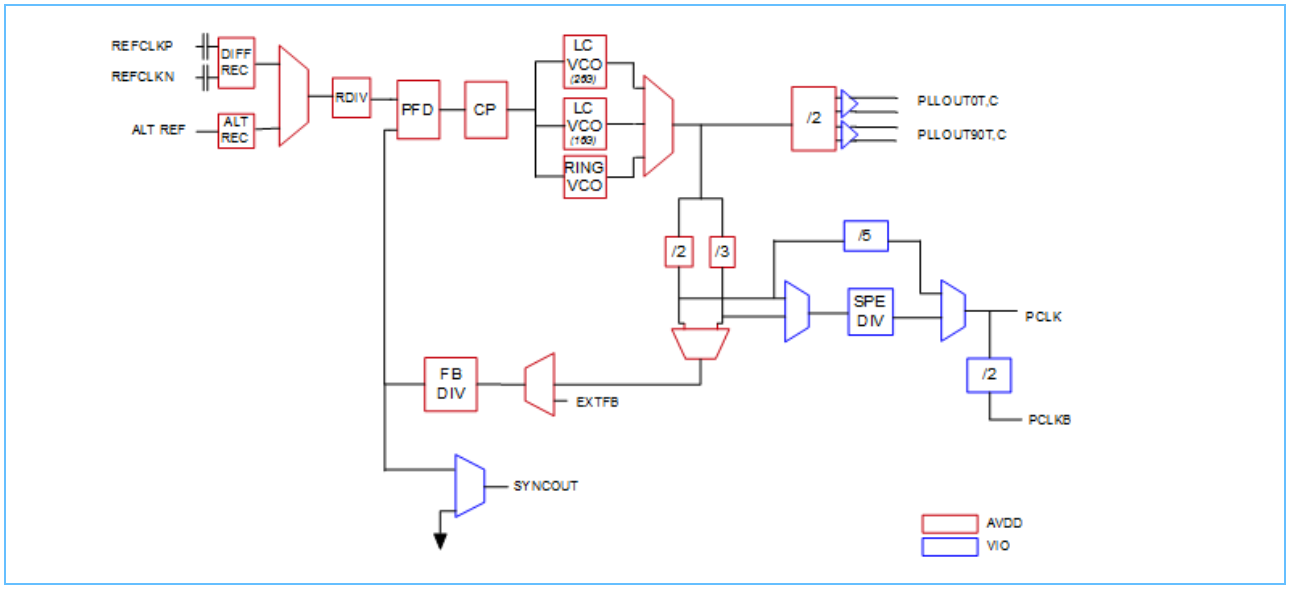
There are also some auxiliary test and characterization features, and individual tuning aspects of the DFE control loop. All of these features have dependencies on external logic blocks.

19.4 PLL Features

The LC PLL takes a reference clock (60 MHz - 1 GHz; 200 MHz POR) and produces a half-rate clock (8.0 GHz or 4.8 GHz) for both the transmit and receive slices. The PLL architecture is based on standard Torrent /HSS topology with PFD and charge pump. In the PLL, two full-rate LC VCOs are used for tuning the 16 GHz mode with a range of 14.4 - 17.6 GHz. Also included is a ring VCO that is used for testing and schmooing (4.3 GHz - 8.6 GHZ). The output frequency range is ½ the VCO range (I-only) phase. The LC VCO implements a band-switched feature to achieve low gain across a wide range. The half-rate clock is generated by a divide-by-2 I/Q clock generation circuit. The VCO bands are selected using a logic algorithm run during initialization that uses the internal FMIN, FMAX, and CVHOLD bits to place the VCO into its minimum, maximum, and central frequencies for each band, respectively.

The mode bit selects the ring VCO and loop filter components so that the PLL can attempt to compensate for the noise of various applications.

Figure 19-3. Block Diagram of PLL



20. OpenPOWER Interface at 25.78125 Gbps

Figure 19-2 on page 280 shows a system context-level diagram of a general bus system that connects two chips "A" and "B" with two uni-directional buses. Also shown is a separation between the core functions of the chip, the bus interface protocol layer, and the bus physical layer. This section gives an overview of the physical layer and on-chip initialization provided by the POWER9 processor.

The POWER9 OpenPOWER interface supports the following types of driver and receivers:

- SMP interconnect
- NVLink 1.0 and NVLink 2.0
- SMP A-bus link
- OpenCAPI is available over the 25G Link (SMP A-bus link). There are 32 lanes available, each supporting a 25 Gbps transfer rate.

20.1 Interface Features

The supported features as summarized in *Table 19-1* on page 278. Requirements are defined relative to the operational-mode definitions.

Initialization mode specifications are as follows:

- Protocols: 5-stage custom training states
- Frequencies: 25.78125 Gbps, 19.2 Gbps, half rates using 2:1 gear ratio
- Software:
  - FSP1 service processor via Global firmware (GFW)
  - MMIO
- Workaround methods: mostly-to-all initialization steps are software controllable
- Supported analog calibration methods: See *Section 20.2 Driver Features* on page 284 and *Section 20.3 Receiver Features* on page 285.

In addition to the initialization specifications, the functional-mode specifications are as follows:

- Frequencies: 25.78125 Gbps, 19.2 Gbps, half rates using 2:1 gear ratio
- Serialization ratios: 16:1
- Scrambling: Full scrambling is enabled during initialization.

Power-saving mode specifications are as follows:

- Frequencies: No special frequencies are required. Also see prior bullet.
- Ability to power down unused bricks to consume 10% of active power.
- Software:
  - FSP1 service processor or host code via GFW; software maintains system status of spares and modes.
  - MMIO

Test mode specifications are as follows:

- Frequencies: 25.78125 Gbps, 19.2 Gbps, half rates
- LBIST: AC and DC
- Hardware-BIST: Built-in and independent TX and RX PHYBIST modes (See diagnostics mode list)
- PHYBIST: Real-time pattern generation test based on reused logic at product frequencies + margin
- Wiretest: PHYs require support of 1149.x features for ASICs commonality. Also quick power-on diagnostics are supported through initialization.
- LSSD: Scan testing per MPG methodology
- Software and test controls

Diagnostics mode specifications are as follows:

- TX and RX internal PHYBIST
- Data sample observability
- RX FIFO pointer collision observability
- SCOM control of all hardware functionality
- MMIO
- Per-bit receiver/sampler offset margining
- Phase rotator values read/write/bias
- Programmable Iref
- Programmable phase detector loop sensitivity
- Programmable DFE parameter loop sensitivity
- Analog net sniffer
- Per-bit power down
- Per-bit quiesce
- Scramble disable. No scrambling in 25 Gbit PHY
- TX amplitude adjustment

20.2 Driver Features

Driver features are as follows:

- 25.78125 Gbps with 16:1 serializer.
- 19.2 Gbps with 16:1 serializer.
- Half-rate series source terminated (SST) with precursor FFE, amplitude margin function, impedance calibration, and postcursor FFE for IOO driver.
- Selectable AC boost: precursor FFE.
- Set and forget impedance calibrator.
- Drive amplitude reduction (margining) up to 50%. For characterization only, not mission mode.
- Full TX power-down mode when port is not required.
- Individual TX lane power-down mode when lanes are not required.

- BIST error detector for at speed loopback testing.
- Shared test pin mode. Differential driver output only.
- Time domain reflectometer.

20.3 Receiver Features

Receiver features are as follows:

- CTLE peaking
- Gain calibration
- 1-tap speculative DFE
- Local offset calibration compatible with floating body devices
- Common mode calibration
- Recovered clock
  - NVLink: 1/16 baud rate CDR clock
  - POWER9 optics: 1/16 baud rate CDR clock
  - Loop bandwidth greater than 3 MHz for good low-frequency jitter correlation with good crosstalk jitter rejection
  - Tolerates at least 100 ppm drift, 100 ppm fixed-frequency offset, and 1000 ppm spread-spectrum support
- SCOM support
- JTAG wire test support
- Eye metrics available on spare lanes with full vertical and horizontal eye scan capability
- Full RX power-down mode when group is not needed
- Individual RX lane power-down mode when lanes are not needed
- CDR must run continuously
- Other parameters are calibrated every 50 ms

There are also some auxiliary test and characterization features, and individual tuning aspects of the DFE control loop. All of these features have dependencies on external logic blocks that are described in the detailed design workbook.

20.4 PLL Features

The dual LC PLL takes a reference clock between 60 MHz - 1 GHz ; with the plan-of-record being 133 MHz . The dual LC PLL produces a half-rate clock of 9.6 GHz or 12.890625 GHz for both the transmit and receive slices. The PLL architecture is based on standard topology with PFD, charge pump, and loop filter. In the PLL, dual full-rate LC VCOs are used for two tuning ranges, the 25.78125 GHz mode with a 28.36 - 23.20 GHz range and a 19.6 GHz mode with a 21.12 - 17.28 GHz range. A ring VCO is also included for testing and schmooing. The output frequency range is ½ the VCO range (I only) phase at the ranges of 14.18 - 11.60 or 10.56 - 8.64 GHz. The LC VCO implements a band-switched feature to achieve low gain across a wide range. The half-rate clock is generated by a divide-by-two clock generation circuit. The VCO bands are selected using a logic algorithm that is run during initialization. The algorithm uses the internal calibration to obtain the optimum VCO band.

The mode bit selects the ring VCO and loop filter components so that the PLL can attempt to compensate for the noise of various applications. (See *Figure 19-3 Block Diagram of PLL* on page 282.)

21. DDR4 Interfaces

21.1 Overview

The POWER9 processor incorporates DDR PHY memory interface physical units capable of supporting several memory topologies. It is optimized for DDR4 memories as defined by the JEDEC, and incorporates all of the required features and many optional ones.

At a high level the DDR unit is responsible for:

- Transporting and mapping command, control, address, and data signals presented from the embedded memory controller.
- Providing all necessary configuration registers, state machines, control logic, and status monitoring to execute all required DDR calibration functions (that is, read calibration, fine and coarse write leveling, ZQ calibration, and so on).
- Providing elastic interface style FIFOs (PHYs) for purposes of sampling, de-skewing, bit aligning incoming data, buffering, and launching outgoing data. These FIFOs also assist in crossing clock domains.

Each DDR unit is self-contained and consists of four independent ports that connect to DIMM slots. This unit is replicated twice on the POWER9 processor to provide a maximum of eight ports.

The DDR PHY supports the following memory devices on each port.

- DDR4 RDIMMs and DDR4 LRDIMMs, including 3D stacks up to eight high
- DRAM data widths of ×4, ×8
- DRAM densities of 4 Gb, 8 Gb, 12 Gb, 16 Gb
- One or two DIMMs per port
- DRAM speeds of 1866, 2133, 2400, and 2667 Mbps

To accommodate DRAM timing variability, and POWER9 process, voltage, and temperature corners, the DDR PHY implements the following calibration sequences:

- Write leveling
- DQS alignment
- Read clock alignment
- Read centering
- Write centering
- Coarse write alignment
- Coarse read alignment
- Tx output impedance calibration
- Read voltage reference (V_{REF}) calibration
- Write voltage reference (V_{REF}) calibration

To accommodate voltage and temperature drifts, DQS alignment, read clock alignment, and read centering can be run periodically after the initial calibrations.

The DDR PHY on the POWER9 processor supports two ranks per DIMM and rank-switching in a minimum of three memory clock cycles. The DDR PHY maximum read latency is eight memory cycles.

To support DDR4 JEDEC specifications above speeds of 2400 Mbps, the following features are supported:

- Programmable preamble
- CRC support
- Rx Vref training

Other features include:

- Per buffer addressability mode (PBA)
- Per DRAM addressability mode (PDA)
- DDR4 maximum power-saving mode
- Per-bit tuning on all address, command, control, clock, data, and strobe signals
- Programmable output impedance and slew rates
- Rank grouping feature
- Extensive RAS support
- Power-down modes
- Custom calibration modes to support custom calibration patterns

21.2 Mainline Operation

The DDR unit must support all mainline functions initiated by the POWER9 memory controller. This includes:

- CKE controls for powering down and powering up ranks and entering and exiting self-refresh mode
- Bank activate commands
- Burst length 8 and burst chop 4 read and write operations
- Periodic refreshes

The memory controller (MC) ensures proper spacing and timing of all command, control, data signals, and adherence to the JEDEC specifications. The primary responsibility of the DDR unit is to propagate all command, address, data, and control signals from the MC unit to and from the DRAM devices. Communication between the MBA and DDR units is done by using an internal bus. The command, control, and address bits flow through a unidirectional ADR43 unit in each DDR PHY port that can drive up to 43 interface pins. Data is transceived through four bidirectional DP16 units and one DP8 unit in each DDR PHY port that can accommodate 72 bits (9 bytes) per port.

22. PCIe Interface

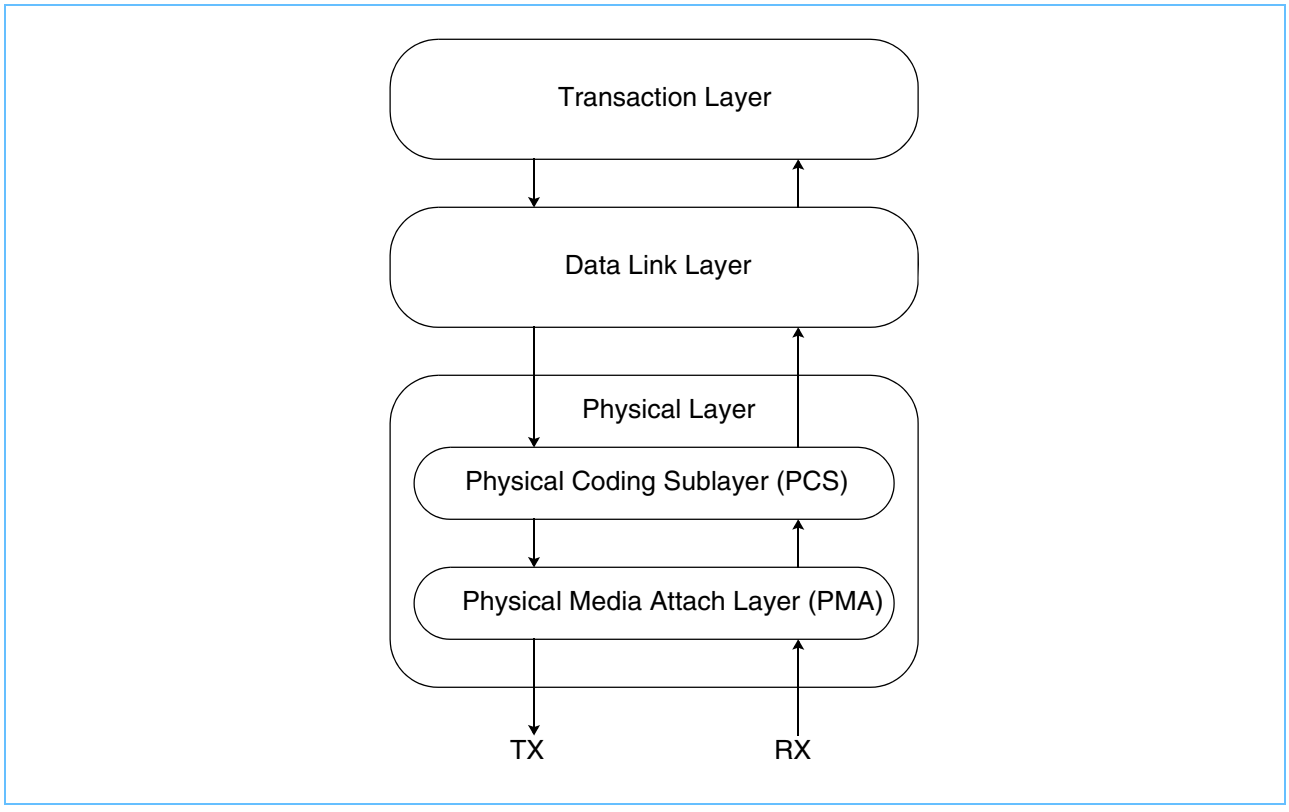
22.1 Overview

The PCIe interface macro, IOP_X844_TOP, contains the physical coding sublayer (PCS) and physical media attach (PMA) hardware layers for implementing the PCI Express GEN4 standard.

The PCS layer is responsible for interfacing the transaction and data link layers with the physical layer. This layer has two main sections. The first is a transmit section that prepares outgoing information passed from the data link layer for transmission by the physical media layer. The other main section is a receiver section that identifies and prepares information received by the physical media layer for consumption by the data link layer.

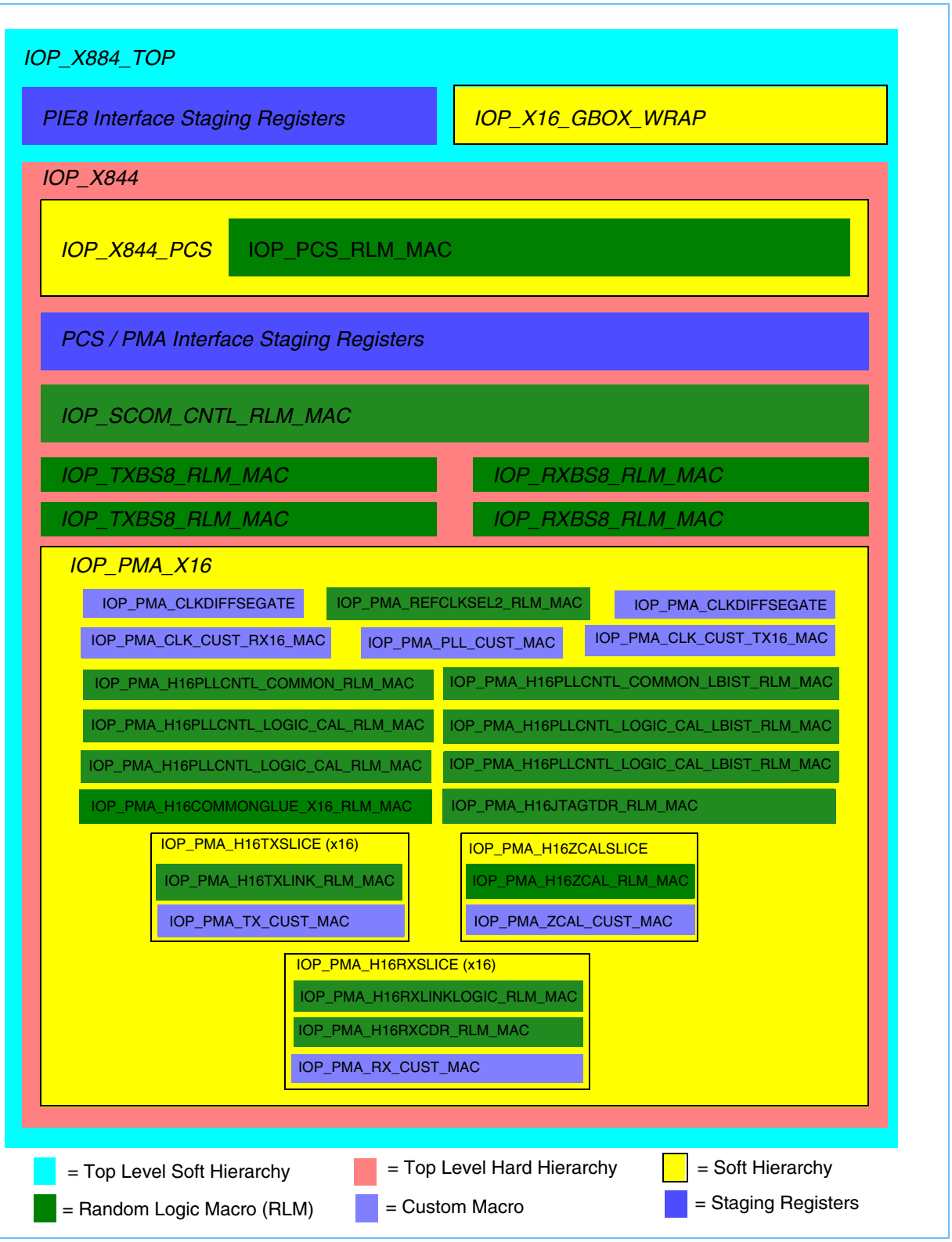
The PMA layer is responsible for serializing data provided by the PCS layer and transmitting on to the link per PCIe electrical specifications. It is also responsible for receiving serial data from the link and provided deserialized data to the PCS.

Figure 22-1. PCI Express Functional Layers Diagram



The IOP_X844_TOP supports a number of PCI Express Gen4 link options. These link options include 1 - 16 lane link, 2 - 8 lane links, 1 - 8 lane link, and 2 - 4 lane links. All link options support 2.5 GTps (GEN1), 5.0 GTps (GEN2), 8.0 GTps (GEN3) and 16.0 GTps (GEN4) rates of data transfer.

Figure 22-2. IOP_X844_TOP Hierarchy Diagram



The IOP_X844_TOP design hierarchy is depicted in *Figure 22-2* on page 290 and briefly described as follows:

- IOP_X844_TOP: Top-level soft hierarchy
  - PIE8 Interface Staging Registers: 2-deep pipeline staging registers on PIE8 interface signals between IOP and ES units
  - IOP_X16_GBOX_WRAP: LBIST stump mux Logic
  - IOP_X844: Top-level hard hierarchy
    - IOP_X844_PCS: Physical coding sub-layer soft hierarchy wrapper
      - IOP_PCS_RLM_MAC: PCS logic RLM
    - PCS / PMA Interface Staging Registers: 1-deep pipeline staging register between PCS and PMA interface signals
    - IOP_SCOM_CNTL_RLM_MAC: Serial communications port logic RLM
    - IOP_TXBS8_RLM_MAC: 8-lane transmitter boundary scan logic RLM
    - IOP_RXBS8_RLM_MAC: 8-lane receiver boundary scan logic RLM
    - IOP_PMA_X16: 16-lane physical media attach layer soft hierarchy
      - IOP_PMA_CLKDIFFSEGATE: Custom differential clock buffers for reference clock
      - IOP_PMA_REFCLKSEL2_RLM_MAC: Power management token logic RLM
      - IOP_PMA_CLK_CUST_RX16_MAC: Custom 16-lane receiver C1 clock distribution
      - IOP_PLL_CUST_MAC: Custom dual LC tank VCO PLL with current reference generators
      - IOP_PMA_CLK_CUT_TX16_MAC: Custom 16-lane transmitter C1 clock distribution
      - IOP_PMA_H16PLLCNTL_COMMON_RLM_MAC: PLL IREF and VREG control logic RLM
      - IOP_PMA_H16PLLCNTL_COMMON_LBIST_RLM_MAC: PLL common logic LBIST RLM
      - IOP_PMA_H16PLLCNTL_LOGIC_CAL_RLM_MAC: PLL VCO calibration logic RLM
      - IOP_PMA_H16PLLCNTL_LOGIC_CAL_LBIST_RLM_MAC: PLL VCO calibration logic LBIS RLM
      - IOP_PMA_H16COMMONGLUE_X16_RLM_MAC: Unit level glue logic RLM
      - IOP_PMA_H16JTAGTDR_RLM_MAC: JTAG and test data register logic RLM
      - IOP_PMA_H16TXSLICE: Single lane transmitter soft hierarchy
        - IOP_PMA_H16TXLINK_RLM_MAC: Transmitter control logic RLM
        - IOP_PMA_TX_CUST_MAC: Transmitter custom analog macro
      - IOP_PMA_H16ZCALSLICE: Transmitter impedance calibration soft hierarchy
        - IOP_PMA_H16ZCAL_RLM_MAC: Transmitter impedance calibration logic RLM
        - IOP_PMA_ZCAL_CUST_MAC: Transmitter impedance calibration custom analog Macro
      - IOP_PMA_H16RXSLICE: Single lane receiver soft hierarchy
        - IOP_PMA_H16RXLINKLOGIC_RLM_MAC: Receiver link control logic RLM
        - IOP_PMA_H16RXCDR_RLM_MAC: Receiver clock and data recovery logic RLM

– IOP_PMA_RX_CUST_MAC: Receiver custom analog macro

The IOP_X844_TOP unit is constructed using the FX14HP technology and provides a total of 16 full duplex lanes of communication support. Each link is independently capable of operating at 2.5 GTps (Gen1), 5.0 GTps (Gen2), 8.0 GTps (Gen3), or 16.0 GTps (Gen4) data rates. The IOP_X844_TOP contains a phase lock loop (PLL) macro that was two independent oscillators. One oscillator is used to produce a 10 GHz clock from a 100 MHz reference clock source. This clock is used to produce 2.5 GTps signaling by dividing the PLL oscillator clock by 4 (¼ rate mode) and to produce 5.0 GTps signalling by dividing the PLL oscillator clock by 2 (½ rate mode). The second PLL oscillator produces a 16 GHz clock from a 100 MHz reference clock source. This clock is used to produce 8.0 GTps signaling by dividing the PLL oscillator by 2 (½ rate mode) and to produce 16 GTps signaling by dividing the PLL oscillator by '1' (full-rate mode).

Table 22-1. Data Rates and Receiver Modes Supported by the IOP_X844_TOP Unit

Data Rate	High-Speed PLL A	High-Speed PLL B
2.5 GTps (Gen1)	[Non-DFE]	N/A
5.0 GTps (Gen2)	[Non-DFE]	N/A
8. 0 GTps (Gen3)	N/A	[DFE12], DFE5, DFE1, Non-DFE
16.0 GTps (Gen4)	N/A	[DFE12], DFE5, DFE1, Non-DFE

DFE12 = 12-tap DFE
DFE5 = 5-tap DFE
DFE1 = 1-tap DFE
Non-DFE = Standard receiver equalization (high-frequency peaking)

The IOP_X844_TOP unit provides point-to-point data transmission over media with a differential characteristic impedance of 100 Ω. This transmission media can be a combination of printed circuit board, connectors, backplane wiring, fiber, or cable. The length of the transmission path is maximized in applications where impedance characteristics are well-matched and the frequency response of the media does not create excessive distortion of the transmitted signal.

The IOP_X844_TOP unit employs numerous equalization schemes to address media losses and crosstalk challenges. The transmitter implements feed forward equalization (FFE) by using a programmable 3-tap, baud-spaced, finite impulse response (FIR) driver with the following equation:

$$H(Z) = K\ (C_0\ z^{+1} + C_1\ z^{0} + C_2\ z^{-1})$$

The driver amplitude (K) is adjusted in the range 200 - 1200 mVppd in 47 power settings. The relative weights of C_0 to C_2 are user-configurable to create a wide variety of transmitter FIR pulse-shaping filters. Reducing the driver amplitude, in general, increases the power consumed by the driver.

The receiver provides a combination of an automatic gain control (AGC) amplifier with dynamic peaking control (DPC) plus a baud-spaced decision feedback equalizer (DFE) circuit that complements the transmitter equalization capability. Twelve taps of DFE equalization are available at data rates 5.0 GTps, 8.0 GTps, and 16.0 GTps. The DFE adaptation circuit examines the incoming serial stream and dynamically adjusts coefficients to maximize the internal eye opening. Two distinct modes of operation for the receive clock and data recovery (CDR) are possible: traditional non return to zero with high frequency-peaking equalization and automatic gain control (non-DFE mode) and DFE mode, which also includes automatic gain control and some high-frequency peaking. The non-DFE mode, combined with transmitter side pre-emphasis filtering and the receiver AGC amplifier, provides a powerful set of equalization capabilities for channel applications where the

losses are low to moderate. When the channel losses are substantial, the PCIe interface can be operated in a DFE mode. The main advantage of DFE mode is that the receiver CDR can correctly deserialize received eyes that are closed with improved bit error ratio (BER). Three selectable DFE modes are available: 1 tap (DFE1), 5 taps (DFE5), and 12 taps (DFE12) providing optimal user performance trade-off capability.

Using the equalization capability of the IOP_X844_TOP, losses up to -28 dB at the fundamental frequency (that is, 8.0 GHz at 16.0 Gbps operation) can be recovered at BER < 10^{-12}.

22.2 Key Features

The IOP_X844_TOP unit in FX14HP includes the following features:

- Supports 2.5 GTps (Gen1), 5.0 GTps (Gen2), 8.0 GTps (Gen3), and 16.0 GTps (Gen4) data rates.
- Contains two independent LC tank-based VCOs operating from a common differential 100 MHz reference clock source that produce 10.0 GHz and 16.0 GHz differential clocks to support internal unit operation. Additionally a 2 GHz single-ended clock is provided to support nest grid clocking.
- Four preprogrammable transmitter and receiver configurations selectable by port using hardware pins or registers. Facilitates fast speed switching during speed negotiation routines.
- Support for spread spectrum clocking of up to ±6000 ppm difference between TX and RX sections at up to 33 KHz modulation. Supports PCIe separate reference clock with independent spread spectrum (SRIS) support.
- Integration of greater than 80 channels per chip.
- Aggressive equalization capability to enable legacy system upgrades.
  - 3-tap FFE driver equalization, baud-spaced
  - Dual-mode CDR: non-DFE or DFE
  - 12-tap Decision Feedback Equalizer (DFE) for use in 8.0 GTps and 16 GTps modes.
    - Programmable DFE length: 1, 5, or 12 taps
  - Variable AGC amplifier
  - Dynamic peaking control
  - Programmable driver launch levels
  - On-chip 100 Ω termination
- Power supply: VDN (nominal): 0.8 V, HSSAVDD1/2 (nominal): 1.5 V, VIO (nominal): 1.1 V
- Multiple Link configurations
  - 1:16 lane bidirectional link
  - 2:8 lane bidirectional links
  - 1:8 lane bidirectional link with additional 2:4 lane bidirectional links
- Asynchronous clock-data recovery.
- Parallel data path width support
  - 2.5 GTps (Gen1) and 5.0 GTps (Gen2): 10-bit width
  - 5.0 GTps (Gen3) and 16.0 GTps (Gen4): 8-bit width
- Driver impedance calibration by using an off-chip precision resistor for accuracy
- Integrated receiver AC-coupling capacitors
- Compatible with PCI Express Base Specification 4.0 supporting 2.5 Gbps, 5.0 Gbps, 8.0 Gbps, and 16 Gbps data rates

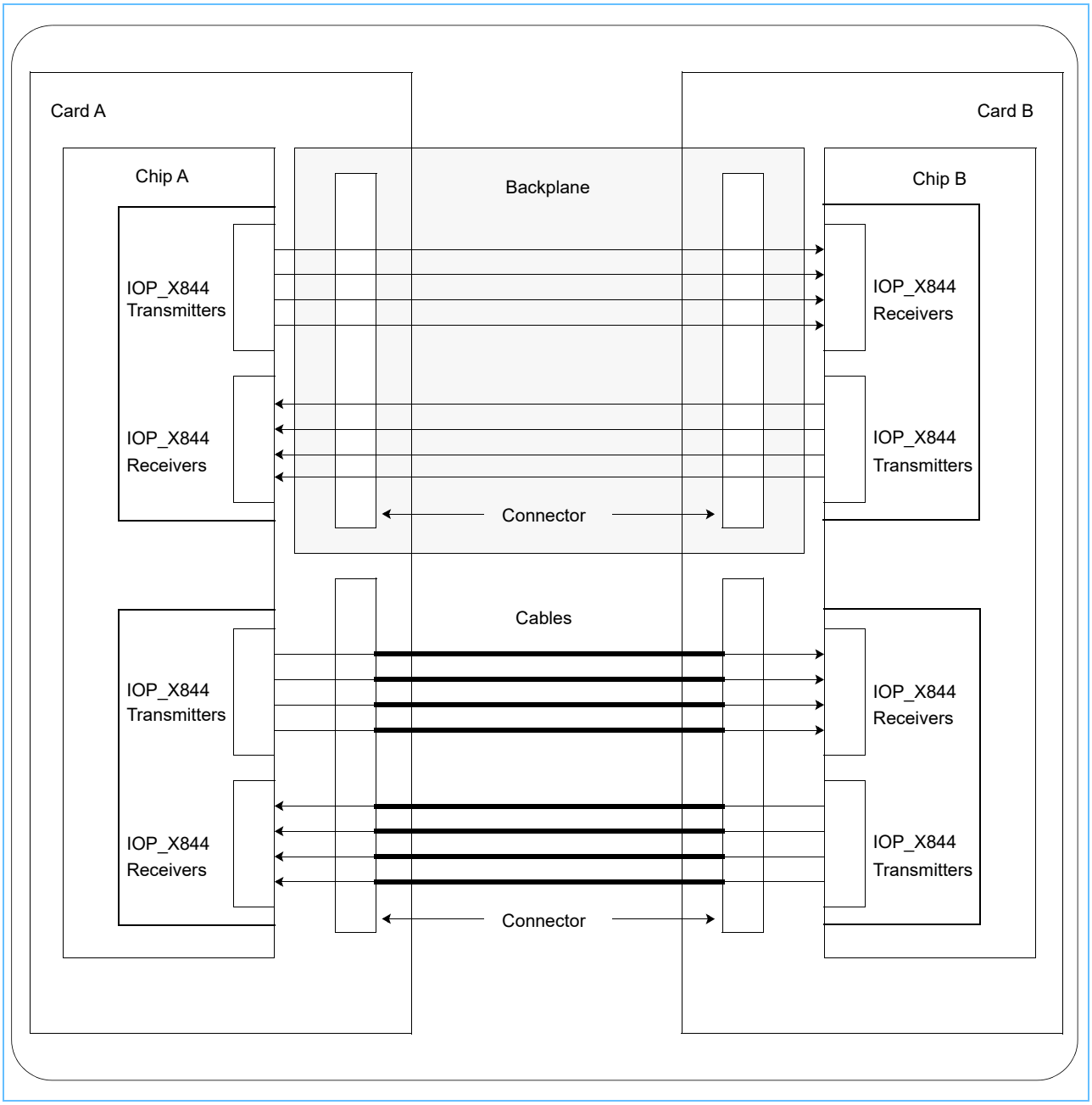
- Support for manufacturing and system test
  - Generalized scan design (GSD) compliant with manufacturing functional (macro) tests
  - Full-rate PRBS built-in self-test (BIST)
  - Compatible with the IBM at-speed structure-test (ASST) at the high-speed data interfaces
  - Compatible with IEEE 1149.6-2003 AC JTAG
  - TDR hardware support
  - Insitu receiver eye monitoring hardware support

22.3 Typical Application

The IOP_X844_TOP unit is used to provide PCI Express Gen4 communication links to systems requiring this communications standard. Multiple IOP_X844_TOP unit instances can be employed to provide greater communications bandwidth. *Figure 22-3* on page 295 shows an example configuration of the IOP_X-844_TOP units in a system environment. The IOP_X844_TOP unit can support an integration count of more than 80 transmit/receive lanes, depending on chip periphery, I/O count, power dissipation, and other packaging constraints.

The high-speed PLLs in the IOP_X844_TOP unit require a reference clock, HSSREFCLK. These clocks are distributed to all of the IOP_X844_TOP units by using a differential clock-tree topology. The reference clocks are provided by using a high-quality off-chip clock source operating at 100 MHz.

Figure 22-3. Typical IOP_X844_TOP Unit Application



Each IOP_X844_TOP unit PMA block consists of sub functional blocks known as slices. Slices are internal subdivisions in the core and are not visible to the core user. These slices are the PLL slice, the TX slice, the RX slice, and the ZCAL slice. The PLL slice implements the two HS PLLs and related logic. The TX slice implements one transmitter, and the RX slice implements one receiver. The IOP_X844_TOP unit PMA block contains one PLL slice, 16 TX slices, and 16 RX slices.

23. Power Management

Like the last several POWER processor generations, the POWER9 processor uses a number of traditional dynamic power-savings techniques. For example, clock gating[1] latches and arrays when they are not required to reduce peak power and, therefore, the thermal design point, also referred to as total design power (TDP). The POWER9 processor can also dynamically clock gate or power gate[2] individual processor cores, or collections of cores and their associated caches when they are not being used.

Additionally, the POWER9 processor continues to support "adaptive power management" techniques to reduce average power and to proactively take advantage of variations in workload, environmental conditions, and overall system usage. The EnergyScale™ firmware, coupled with the policy direction from both the customer and feedback from the hypervisor and operating system that is running on the machine, determine the modes of operation and the best power and performance trade-off to implement during runtime to meet customer goals and achieve the best possible performance. Like the POWER8 processor, the POWER9 processor contains an on-chip controller (OCC) to run the EnergyScale firmware, which supports software-requested performance states (Pstates), Idle (Stop) states, chip and system thermal management and protection, and power-supply current over-limit protection.

Managing the power and performance trade-off is a complex problem. There are many ways to control the behavior of the hardware, but these also have a number of side effects that vary based on the workload being processed. Because there is no single policy that can be implemented, the POWER9 processor, like its predecessors, supports an adaptive approach to the problem in the form of a joint hardware, firmware, and software solution, collectively known as EnergyScale. EnergyScale provides the mechanisms that enable the customer to observe the power, performance, and usage of the processors and other components of the system.

23.1 Policies and Modes of Operation

EnergyScale removes the "ugly" details of a low-level hardware implementation to provide policies (operational modes) that allow the customer to achieve the required level of power and performance efficiency within specified bounds. Implementations differ depending on which hypervisor is running on the system: Power KVM or PowerVM.

The POWER9 chip supports multiple power management choices for system operation, which can be selected by the customer depending on the situation at any given time or for a particular datacenter's constraints.

The POWER9 power management supports IBM's second generation of Workload Optimized Frequency (WOF), a mechanism where the OCC can take advantage of the available socket power to increase processor core frequency. The OCC power-management firmware components consider, in "real time", factors such as workload intensity, Pstates in use across the chip, and powered-off cores to boost the maximum allowed frequency within the socket power, electrical current, and thermal budgets.

1. Clock gating involves deactivating clocks for portions of a circuit that are not in use.
2. Power gating involves turning off the current to portions of a circuit that are not in use.

23.1.1 Power Management in Linux-Based Systems (Power KVM)

OpenPOWER systems run a Linux-based hypervisor. The primary interface to EnergyScale in Linux-based systems is from the Linux Governor[1]. The following links describe the available capabilities:

- *CPU frequency scaling*
- *The Ondemand Governor*

23.1.2 Power Management in PowerVM-Based Systems

On IBM-branded PowerVM-based systems, the modes and policies provided to the customer are similar to previous Power systems and are described in greater detail in the following white-papers:

- *IBM EnergyScale for POWER8 Processor-Based Systems*
- *Manual for Using WBEMCLI Tool to Fetch Flexible Service Processor CIM Data*

23.2 Base Enablement Summary

Power Management consists of four major elements, each requiring a set of functions available in the hardware to accomplish these elements. EnergyScale firmware requires a processor on which to run decision-making code, a way to measure what the processor and other system elements are doing, a way to actuate (control) the runtime operation of the system, and a way to shut off components that are not currently being used.

23.2.1 On-Chip EnergyScale Microcontroller

Real-time monitoring and decisions must be made to optimize the power and performance of the system while maintaining safe operational parameters for the chips and the other system components. To accomplish this, a dedicated on-chip microcontroller (OCC) is included on the POWER9 chip. This is an embedded PPC-405 core that runs at ¼ of the nest frequency (typically between 400 - 600 MHz). The OCC complex also contains local SRAM, access to system DRAM memory, and access to on-chip SCOM (clocks-running scan communication) registers via the on-chip pervasive control bus (PCB) network.

23.2.2 Measurement Capability

To make intelligent decisions, the OCC must be able to measure the state of the system during runtime. Sensors embedded in the system components (processor chips, memory chips, power supplies, and so on) enable access to various aspects of the system components:

- Temperature
- Power (voltage and current)
  - Analog sampling on the various voltage rails
  - Note that the POWER9 processor does not include digital power proxy (per-core power estimation circuitry)
- Activity metrics
  - Utilization, usage, and performance

1. The Governor is the component of Linux software responsible for managing the work allocation, power, and performance of the microprocessor cores.

These are outlined in more detail in *Section 23.5.2 Sensors* on page 305.

23.2.3 Dynamic Voltage and Frequency Scaling (DVFS)

On PowerVM-based systems, EnergyScale firmware chooses the optimal frequency and voltage during runtime in response to the workload running and the policy selected by the customer.

Linux-based systems use power/performance states (Pstates) as an abstract representation of frequency and voltage on a per-thread basis. EnergyScale firmware must then combine the "votes" in an "auction" process to choose the optimal operating point.

23.2.3.1 Pstates

Pstate requests are represented as an 8-bit unsigned value, where Pstate0 is F_{MAX} at V_{MAX} supported for the core on this particular system and chip sort point. Each increase in Pstate request value represents a drop in frequency of 16.67 MHz assuming a 133 MHz reference clock input, meaning 4.2 GHz of Pstate space can be regulated in theory. For each of the discrete Pstates, EnergyScale calculates values derived from manufacturing test characterization data.

23.2.3.2 Actuation

DVFS involves more than changing the frequency and voltage to a processor. To change Pstates, EnergyScale firmware must control analog iVRM tune settings for V_{DD} (based on maximum current ratios) and manage resonant clocking mode and sector buffer strength settings for the clock grid and voltage droop monitor compare and threshold settings.

To accomplish DVFS, the following actuation is required:

- Processor core frequency control
- External voltage regulation module (VRM) control
- Internal voltage regulation module (iVRM) control
- Pstate clipping function to enforce a power cap and support deterministic workload optimization

23.2.3.3 Instrumentation

To perform DVFS intelligently and safely requires the following:

- Thermal measurement to know if thermal constraints are being maintained
- Power measurement to know how power supplies (either current or power) are performing
- Power measurement distribution to allow all chips in a system to have access to measurements
- Activity counters to gain insight in the workload characteristics to make good frequency choices

Coordination of all the measurements within socket constraints of power delivery and thermal limits requires the use of a highly programmable OCC complex.

23.2.4 Processor Idle (Stop States)

To enter the processor idle state, the following elements are required:

- Stop instruction
  - Executed by the software running on each thread of the processor core
  - Requests entry into a specified Stop level via the PSSCR
  - Stop level specifies the allowed wakeup latency and dictates the amount of power saved
- Coherently disconnect cores and also their caches from the SMP Fabric
- Clock and power gating in response to executing the Stop instruction on all threads of the core
- Restoration of cores and caches to operational mode in response to wakeup events (interrupts)

These elements with their underlying functions are further broken down to elements that are core centric and chip centric.

23.3 Feature Summary

A few strategic design changes were made in the POWER9 chip to provide additional high-value power management capability. These features include:

- Workload optimized frequency support:
  - Deterministically increase the frequency (and performance) of the processor based on the combined power consumption of the workloads running on the cores of each processor chip.
  - Hardware includes higher-priority communication channels between on-chip microcontrollers.
- Stop states for idle cores (similar to the x86 notion of C-states):
  - Replace the doze, nap, sleep, winkle instructions, and their fast and deep variants, which were present in previous POWER processor chip architectures.
  - Support a set of numbered stop levels, each with increasing power savings and exit latency.
  - Enable a core instant-on mode, supporting a rapid exit from a core powered-off state (for example, about 250 ms).
  - Enable lighter guest or OS-level stop states, without incurring the latency of state loss and restore.
- Dynamic I/O bus width modes:
  - Adapt to periods of lower activity or phases of low-system usage, especially in a cloud environment.
  - Adapt to periods of lower link activity or phases of low-system utilization:
    - Fabric links dynamic 2-byte mode for systems with 4-byte links
    - NVLink 1/8 mode
    - PCIe ×1 mode capability (requires device driver and system software support)
- Fully programmable PowerPC-lite Engine (PPE) instances:
  - This new design for the POWER9 chip is an embedded PowerPC core based on a subset of the PPC405.
  - Distributed throughout the chip in proximity to localized power and performance management functions:

— Off-loads the lower-level power management functions, replacing previously fixed-state machines.
— Enables more flexibility to work around unforeseen problems without requiring a hardware change.
— Implements improved mechanisms and algorithms.

23.4 Power Management Infrastructure

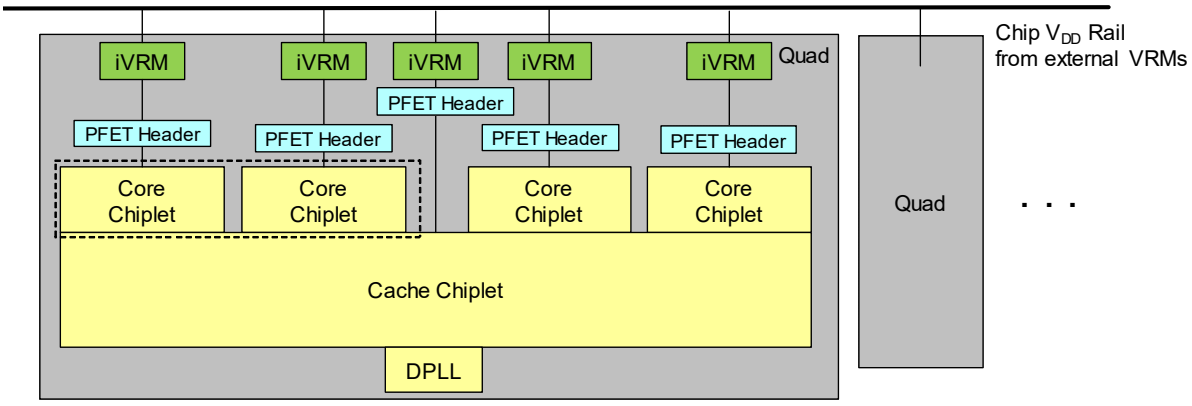
The POWER9 processor does not support per-core frequency control because of physical chip layout and wiring constraints. Instead, SMT4 processor cores are arranged in groups of four, along with their respective L2 and L3 caches, into a structure known as a Quad. The frequency is managed at the Quad level. This means DVFS must be managed at a Quad level based on the fastest Pstate request of the four cores that occupy the same Quad. Independent per-core voltage control is therefore only possible when a core enters a Stop state and is clocked off. Otherwise, the voltage and frequency of all cores in the Quad must track in unison. When a core enters a Stop state and is powered off, its L3 cache remains active until all four cores in the Quad enter into a sufficiently deep Stop state at the same time.

Likewise, within a quad, pairs of SMT4 cores (or each SMT8 core) share an L2 and L3 cache. The L2 cache can only be clock gated if both cores using that L2 cache enter a Stop state that will allow it.

23.4.1 Quad Voltage and Clock Domains

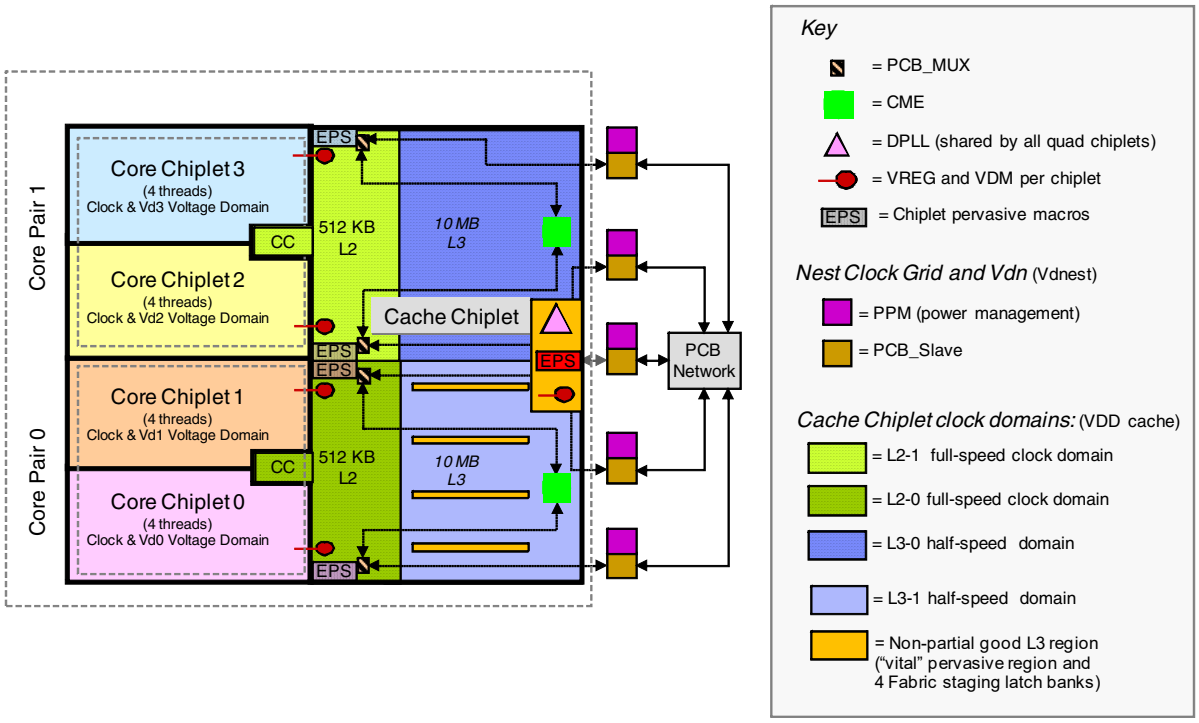
The Quad voltage control can be visualized as shown in *Figure 23-1*.

Figure 23-1. Quad Voltage Control



A more detailed visualization of the Quad with power management related infrastructure, as well as clock and voltage domains is shown in *Figure 23-2* on page 302. Each Quad is replicated six times on the POWER9 chip and contains five synchronous chiplets (four core chiplets plus one cache chiplet). Each core chiplet contains four SMT threads. The cache chiplet contains three clock grids, one full-speed grid for each L2 cache unit, and one half-speed clock grid shared by both L3 cache domains. Each of the two instances of L2 and L3 cache domains are functionally dedicated to a core pair. Each pair of physical core chiplets can appear to software as either a single SMT8 core or a pair of SMT4 cores depending on which POWER9 chip part was manufactured. In all chip configurations, the physical SMT4 core chiplets are clocked and powered off independently.

Figure 23-2. Detailed Quad Voltage and Clock Domains



The POWER9 processor implements a hierarchical power-management solution. Power Management controls are applied at the lowest level possible to allow the greatest flexibility and to reduce overall complexity of the hardware design.The hypervisor runs Energy Management algorithms controlling power and performance of the cores at the partition and micropartition level by requesting Pstates and Stop states. The OCC manages controls at the processor chip and memory level. In general, power management hooks exist inside the processor core itself, inside the Quad, in the chip-level "nest" unit level, and at the chip level. This hierarchy affects how the features are implemented and therefore, laid out in the SCOM registers. For example, internal voltage and power gating can be controlled at both the core and Quad level (the external voltage level continues to be shared at the chip level), frequency at the Quad level, and software-directed modes and instruction throttling controls inside the core itself.

23.4.2 On-Chip Microcontrollers

Figure 23-3 is an abstract visualization of the power management-related hardware on the POWER9 chip. Shown are the twenty-one instances of the PPE plus the OCC microprocessor, that together enable the power management functions. The IO-PPE and SMP Fabric PPE are also tasked to do other non power-management related functions; as well as the self-boot engine (SBE), which is responsible for IPL of the chip and for chip access security during runtime.

Figure 23-3. High-Level Diagram of POWER9 PPE Instances

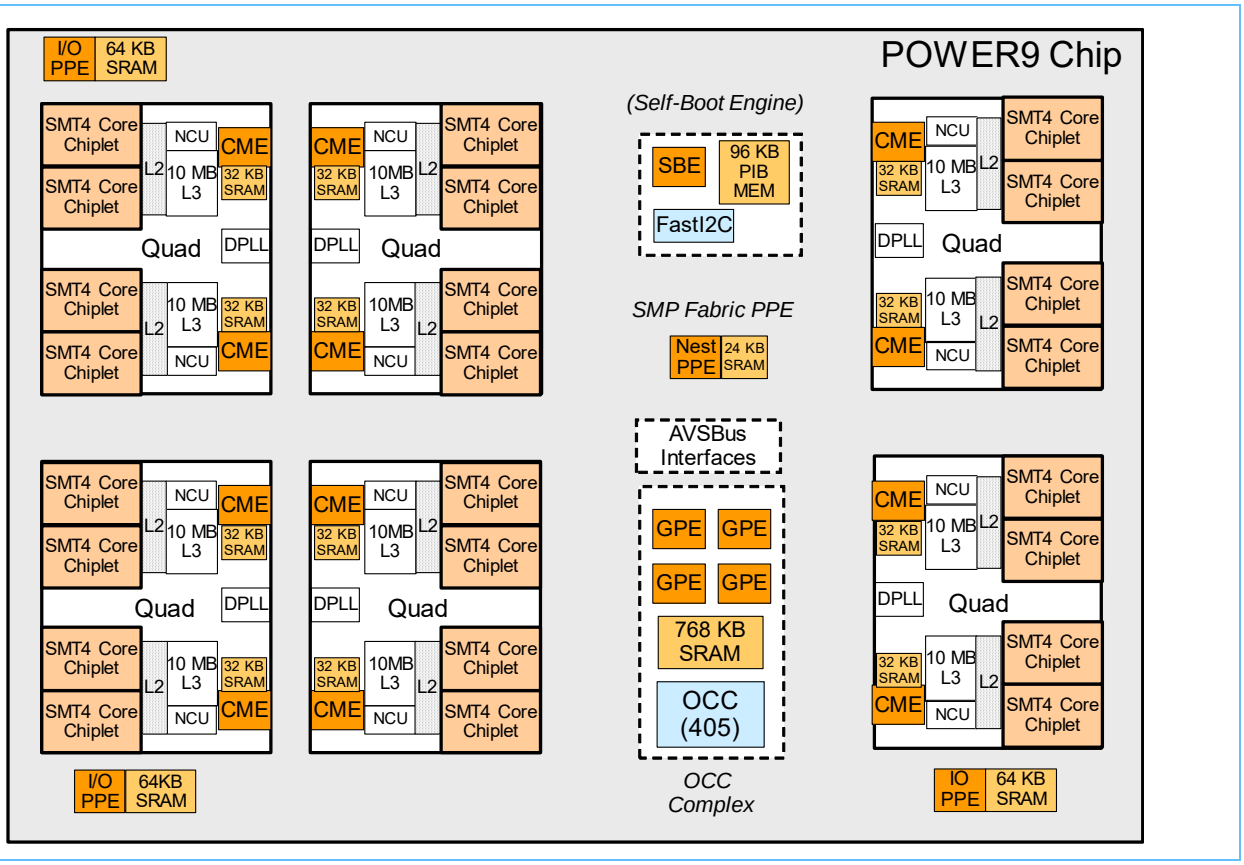
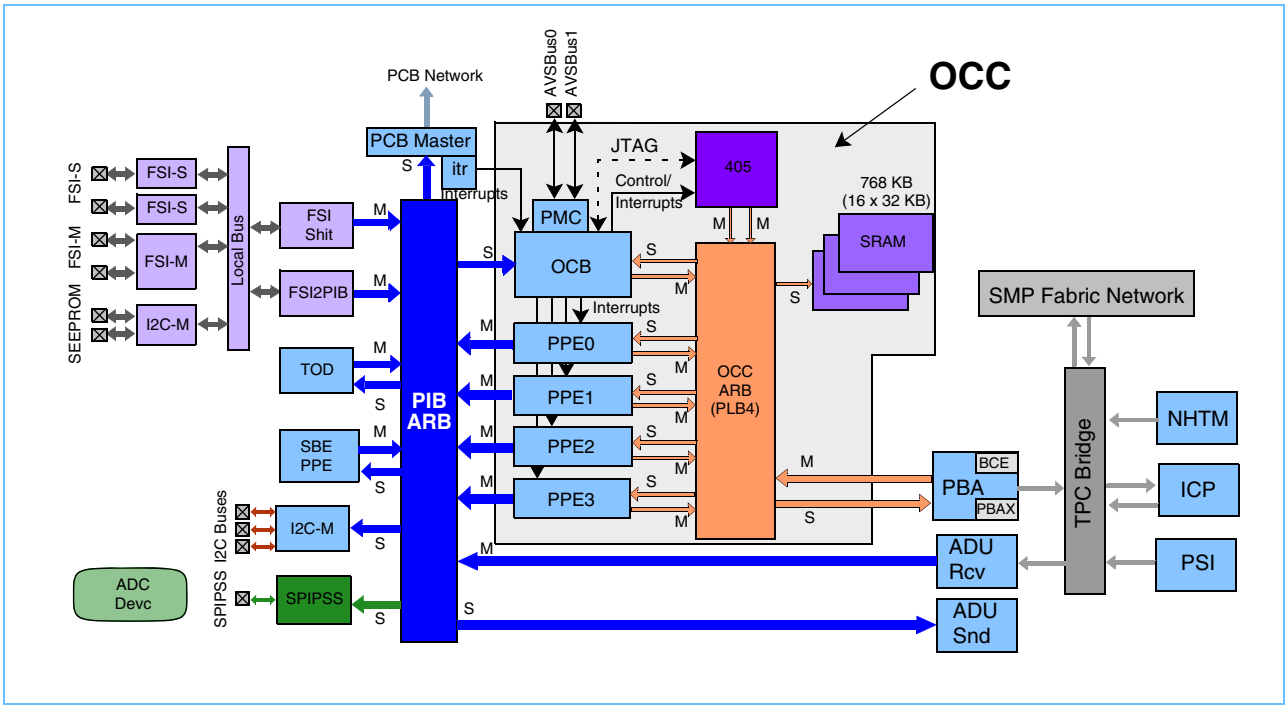


Figure 23-4. OCC Complex



23.5 Chip Hardware Features

23.5.1 Communication Paths for Firmware

- Dedicated special wakeup bit per core and cache chiplet for four different firmware components
- EnergyScale firmware communication
  - Four independent messaging queues (up to 64 bytes each) in the OCC SRAM, intended to be used by the hypervisor and service processor firmware to communicate with the OCC via the respective SCOM operations.
  - OCC is attached to the on-chip pervasive PCB network, enabling the OCC firmware to access the various functional units on the chip.
  - OCC firmware accesses the controlled regions of system memory via the on-chip fabric.
  - CME firmware communicates with the hypervisor by writing the Power Management Status Register (PMSR), which is available to the hypervisor as a special purpose register (SPR). See *Section 23.5.8.3 Power Management Status Register (PMSR)* on page 312 for more information.
  - The hypervisor communicates to the EnergyScale firmware by writing the Power Management Control Register (PMCR) SPR (for example, Pstate requests). See *Section 23.5.8.1 Power Management Control Register (PMCR)* on page 312 for more information.

23.5.2 Sensors

This section describes various power management sensors.

23.5.2.1 Analog On-Chip Thermal Sensor (OCTS)

- Reduced set of analog thermal diodes available as on-chip thermal sensors (OCTS), available only via C4s.
- Only used for calibration of the DTS during manufacturing test.

23.5.2.2 Digital Thermal Sensor (DTS)

- Diode bandgap design with built-in A/D converter to provide digital readout.
- Sensor collection macro converts to °C on read (using calibration settings from manufacturing test that is loaded during IPL).
- Two implemented per core chiplet; two implemented per cache chiplet; three in the nest region at the chip level.
- Available via SCOM during runtime, used by the OCC to safely implement DVFS and to protect the chip from over-temperature conditions. Thirty-two SCOMs are required to access all the chiplets (24 core + 6 cache + 2 nest chiplets).
- Not available to CME; only available to OCC, SBE, or the service element
- No automated hardware thermal over-temperature protection.

23.5.2.3 Voltage Droop Monitor

- The voltage droop monitor (VDM) includes a detection circuit that monitors deviation of the local power grid from a target voltage. The VDM is configurable to indicate one level above and three levels below that voltage.
- The VDM is implemented on the core V_{DD} and cache V_{DD} power grids: one per core chiplet and one per cache chiplet; five instances per Quad region.
- The VDM allows for guardband reduction via DPLL frequency feedback and as a protection mechanism from sudden voltage droop excursions.

23.5.3 Dedicated Activity/Event Counters

23.5.3.1 Processor Core EMPATH Counters

- Dedicated to power management firmware, not shared with the performance monitor unit (PMU).
- Used for processor and memory usage measurement to direct power and performance trade-off decisions and the selection of appropriate power-management techniques.
- Addressable by the hypervisor as SPRs or to EnergyScale firmware as SCOM registers.
- Free-running (not cleared on a read) and wraps (roll over) upon reaching the maximum value, to support noncritical timed access and use by multiple firmware entities. 32-bits wide, with pre-counters where necessary for high-frequency or multiple events, to ensure roll-over occurs in more than one second.
  - Firmware reads these registers, and therefore must sample these registers twice and subtract the values to determine the elapsed count after detecting and accounting for the wrapping in the calculation.
- The following counters are important inside the processor core:
  - For per-core accounting:
    - Raw cycle count
    - Stall counters (workrate busy and finish)
    - Two programmable memory subsystem hierarchy counters
    - System memory (DRAM) access counter
  - For per-thread accounting:
    - Active run cycles (how often the operating system sets the CTRL run latch, indicating that active work is being processed off the run queue)
    - Instruction dispatch
    - Instruction completion
    - System memory (DRAM) access counter

23.5.3.2 Nest SMP Fabric Usage Counters

- Implemented as SCOM registers that can be accessed by the OCC
- Over-commit rate/retry counters in the OCC SMP Fabric attach macro
- Other counters embedded in the SMP Fabric
- Fabric PPE can summarize the SMP Interconnect activity for sampling by the OCC complex

23.5.4 On-Chip Microcontroller Complex

The OCC complex provides on-chip microcontrollers along with a communication support infrastructure that enables them to communicate to other components on the chip and in the system.

23.5.4.1 On-Chip Microcontroller (OCC)

The OCC includes:

- An embedded PowerPC 405 with 16 KB instruction and data caches.
- On-chip 768 KB SRAM tank.
- Runs at ¼ frequency of the Nest interconnect.
- Access to system DRAM memory via the Nest Fabric for instruction and data area overflow (firmware managed).
- Access to full chip pervasive infrastructure (PIB/PCB) via a bridge from the native 405 processor local bus (PLB). The PLB is termed the on-chip controller interconnect (OCI) on the POWER9 chip.

23.5.4.2 General Purpose Engines (GPEs) for OCC Function Off-Load

- Two dedicated instances of the PPE are used to off-load mundane tasks. They are necessary to free up the OCC to run its real-time operating system and to properly protect and optimize system functionality.
- One GPE is targeted to run operations as scheduled by OCC firmware as part of the real-time control loop (for example, sensor collection from each chiplet or selected actuation functions).
- One GPE is targeted to run operations as scheduled by OCC firmware as part of the real-time control loop or for background tasks for data collection and other OCC support functions (such as, performance monitor PMU-let collection and atomic memory counter updates, memory thermal-sensor collection, and so on).
- GPE programs are OCC SRAM resident. Data areas can be in either SRAM or system memory.

23.5.4.3 GPEs for Chip-Level Function Management

Two GPEs that are also part of the OCC complex are dedicated to support particular chip-level power management functions.

S-GPE: This third PPE instance in the OCC complex primarily manages entry and exit from Quad-level Stop states. When the Stop state powers off an entire Quad chiplet, that state must be restored in the form of a mini-IPL of the cache chiplet, including restoration of the CME code image.

P-GPE: This fourth PPE instance in the OCC complex manages Pstate transitions (Quad DVFS) in response to processor core Pstate requests, protection functions (system power capping, thermal over-temperature limit, power supply over-current limit), and to support OCC-directed WOF direction. This PPE also manages the sequencing of external VRMs and other devices by driving high-level SPI interface protocols. Other auxiliary functions can share this PPE as additional threads of execution (for example, performance monitoring support).

23.5.5 Dedicated Core Management Engines (CME)

The dedicated core management engine has the following features:

- One PPE instance per core chiplet pair (two per Quad) to manage local power management functions and algorithms.
- 32 KB local SRAM with block-copy engine access to memory via the NCU and L3 cache unit, which provides access to the SMP interconnect.
- Runs at ¼ frequency of the L3 cache (1/8 speed of the processor cores).
- Necessary for "core instant-on" to minimize Stop state transition latency for levels below Level 8.
- Responds to changes to PMCR and PSSCR SPRs and can update PMSR SPR back to the cores.
- Off-loads intensive chores from the centralized system power controller:
  - Core chiplet Stop state sequencing
  - Localized quad (core and cache chiplet) Pstate management

23.5.6 On-Chip Accelerators

23.5.6.1 Chiplet Pervasive-Power Management (PPM) Extension

Macros are associated with each core and cache chiplet to enable the CME and OCC Complex to communicate and control the Quads. In addition, it provides access to the chiplet pervasive network control registers in the neighboring PCB-slave macros associated with the Quad.

PFET Power Gate Control State Machine:

- Necessary for di/dt management of core and cache chiplet power-on and off transitions
- Used for entry and exit of iVRM regulation mode
- Programmable 8-step on and 8-step off sequences for full on/off
- Subset used for entering and exiting iVRM enablement (for example, enter and exit internal regulation)

PCB-Network Interrupt Generation Mechanism:

- Provides communication mechanism to chip-level OCC-complex, either:
  - Used by code running on CME, or
  - Automatically generated by hardware for the unlikely case of CME errors or for Quad wakeup events that require Stop-GPE assistance

23.5.7 Actuator and Control Features

23.5.7.1 On-Chip Frequency Control

- Frequency can be varied at the Quad level (four cores plus the cache chiplet) and not at the core level because of the POWER9 chip physical infrastructure.
- The DPLL has a requested dynamic frequency range from 0.5 GHz to greater than 5 GHz. Note that the full range is not supported or achievable by every chip or every system.
- Reference clock mode saves clock grid power when the core or cache chiplets are in a clocked-off but still powered-on idle state.

23.5.7.2 External (Off-Chip) VRM Voltage Control

Connectivity from the POWER9 chip to the external VRMs via off-module C4s is supported.

- Access to an I^2C master is provided.
- Two industry-standard PMBus (AVSBus) interfaces are available: http://pmbus.org/Specifications/CurrentSpecifications
  - Low-level AVSBus communication protocol is handled by the P-GPE or OCC firmware.
  - VRMs can return a status frame for command confirmation of VID write validity.

These connections are used to send VID codes to VRMs associated with a given chip. Actual connectivity is dependent on system implementation.

  - V_{DD} (core and cache logic) voltage targets (required for DVFS)
  - V_{DN} (nest logic, optional control that is based on the service element structure)
  - V_{CS} (SRAM and array in the cache and nest domains) voltage targets (optional control that is based on the service element structure)

External VRMs that use load-line sensing, automatically ramp each voltage rail to the given target without stepping assistance from the POWER9 chip or firmware.

23.5.7.3 External Sampling

When connected, the AVSBus or I^2C interfaces can also be used to read voltage, current, and VRM status.

An industry-standard SPI bus (via off-module C4s) can be connected to a system-level analog-to-digital converter that enables OCC firmware to read the current and voltage (such as, power) sampled on various voltage rails in the system. Low-level SPI communication protocol Is handled by hardware.

23.5.7.4 On-Chip Voltage (iVRM) Control

Per-Quad DVFS is enabled by on-chip voltage regulation of the V_{DD} rail. Note that V_{CS} is not power gated. Each of the five chiplets in the Quad has an internal VRM (iVRM) with controls to enable their proper operation. Because all chiplets in the Quad share the same frequency, during runtime operation the iVRMs must be set to the same value. The iVRMs can potentially be used for a Stop state to drop a clocked-off core chiplet to V_{MIN}. The iVRMs enable Quads requesting a lower-performance Pstate to have a voltage reduction and the accompanying drop in power, despite the external rail being at a higher voltage because other Quads on the chip are requesting a higher-performance Pstate.

23.5.7.5 Core and Cache Chiplet Power-Down

P-FET devices are used as switches to independently power off core and cache chiplets during runtime in response to all threads on a core or in the Quad executing the Stop instruction. The power-down and power-up sequences on both V_{DD} and V_{CS} follow an 8-step ramp process to avoid injecting droop or noise on the neighboring power grids.

23.5.7.6 Resonant Clocking Mode Support

For the high-speed core and L2-cache clock grids, a resonance mode is provided to save active clock grid power. In this mode, on-chip inductance on the grid is tuned to minimize the power consumed. Power-management firmware must tune these controls across certain DVFS ranges. Pulsed clock mode and clock sector strengths are also controlled.

23.5.7.7 Voltage Droop Protection

A voltage droop monitor (VDM) is instantiated per core and in the cache region of the Quad. To protect circuit margin in the case of an unexpected instantaneous drop in voltage, the DPLL can be configured to reduce the operating frequency to preserve guardband. In response to detection of a small droop event, the DPLL can be configured to reduce a small percentage of the operating frequency, which also serves to mitigate via a more timely reaction should a future larger droop also occur. In response to a large droop event, the DPLL can be configured to reduce a larger percentage of frequency. Lastly, if a droop event begins to approach operational V_{MIN} in an extreme case, the core can be configured to temporarily throttle back instruction fetch and issue in the processor cores as an emergency protection measure.

With this detection and protection, the POWER9 processor can ship at higher operational frequencies while still preserving guardband for safe, reliable operation. The definition of small, large, and extreme droop are programmable in the power management logic. Droop events are expected to be rare and the protection mechanisms do not measurably affect performance because they impact performance by only a fraction of one percent.

23.5.7.8 OCC Hang Detection Hardware

An OCC heartbeat register is implemented per quad, as a timer that must be reset periodically. If the OCC becomes nonresponsive and the hardware timer overflows:

- A malfunction alert can be generated to the processor cores to communicate lack of OCC health to the hypervisor firmware running on the POWER9 cores.
- A local external interrupt is generated to the P-GPE so that it will enter safe mode (reduced frequency with a potential reduction in voltage, and if necessary, some amount of instruction throttling) while the OCC is unavailable to manage the chip's health.

23.5.7.9 Active Power-Down of Unused I/O PHYs

The POWER9 processor supports static disablement of active power for unused PHYs and I/O clocks in systems where an interface is not populated.

Elastic-interface buses support hardware driven dynamic spare lane power down. If another lane on the interface fails, these spare lanes are immediately powered up and ready for use.

23.5.7.10 Partial Good and Runtime Deallocation

The POWER9 processor supports static partial-good capability. This means that during IPL, deconfigure bad cores or their associated caches that have manufacturing defects and prevent their enablement. The POWER9 processor supports dynamic deconfiguration of cores and quads should they not all be licensed for use during runtime. In PowerVM-based systems, the POWER9 processor supports runtime de-allocation of bad cores and quads, which permanently deconfigures them in response to a GARD operation that is caused by a detected hardware error. Bad or deconfigured cores are considered to be in the deepest Stop state (Stop level 15), such that the other SMT4 or SMT8 core in the pair or the entire Quad can enter deeper states if required. As many as possible of the deconfigured cores and quads are turned off to save power:

- Deconfigured single cores are completely power gated.
- Deconfigured core pairs drop to slightly lower power stopping the associated L2 clock grid and powering off both cores.
- Deconfigured quads (including four-core chiplets, their cache chiplet, and the SMP fabric interface) are completely power gated.

23.5.8 Architected Control Registers

23.5.8.1 Power Management Control Register (PMCR)

The PMCR is the mechanism used by the hypervisor firmware (for example, PowerKVM) to request Pstate changes. The 8-byte register is implemented as a generic special purpose register per core. The format and content of the register is defined by firmware and is intended to contain Pstate requests along with other performance and power optimization hints, such as workload priority and quality of service expectation, dependent on the version of firmware using it.

Note: The layout of this register differs from the POWER8 definition. Except for Pstate0, perform a two's complement of the POWER8 Pstate value and multiply it by '2' to convert it to the equivalent POWER9 Pstate value.

Table 23-1 describes the PMCR.

Table 23-1. PMCR Description (Version 0x1)

Bits	Field	Description
0:7	UpperPS	Upper Pstate request (for future enhanced Pstate support). This field is ignored in Version 0x1of this register format. **Note:** Unlike in the POWER8 core, separate Global Pstate requests are not supported in the POWER9 core.
8:15	LowerPS	Lower Pstate request. In this field, x'00' represents the fastest frequency supported and subsequent increasingly positive values (x'00', x'01', x'02', …) are decreasing frequency in 16.667 MHz steps from the fastest frequency as the base. In Version 0x1 of this register format, this field provides both the Local and Global Pstate request. **Note:** For comparison, the POWER8 processor used increasingly negative values (x'00', x'FF', x'FE', …), in 33.333 MHz steps in this field and in the previous field.
16:59	Reserved	Reserved for future enhancements. **Note:** In version 0x1, these bits are ignored and must be all 0's.
60:63	Version	Indicates the format of the fields in this register expected by firmware. x'0' POWER8 format, not supported but treated the same as version x'1' by the POWER9 core. x'1' POWER9 format, using only bits 8:15. x'2' Future extensions for enhanced Pstate support.

23.5.8.2 Power Management Idle Control Register (PMICR)

The PMICR is not implemented on the POWER9 chip. The function is subsumed by the new per-thread PSSCR Register, which is defined in the Stop instruction architecture.

23.5.8.3 Power Management Status Register (PMSR)

The PMSR enables Energy Management firmware to communicate to the hypervisor running on the POWER9 core. The 8-byte register is implemented as a generic special purpose register with only one instance per core. The format and content of the register is defined by firmware to communicate metrics for energy efficiency and Quality of Service, such as the actual Local and Global Pstates that are achieved.

Note: The format of the upper word of this register is unchanged from the POWER8 definition and the lower word is defined for the POWER9 core.

Table 23-2 describes the PMSR.

Table 23-2. PMSR Description (Sheet 1 of 2)

Bits	Field	Description
0:7	Global Actual Pstate	Global Actual Pstate. Represents the Pstate value that pertains across all cores on the chip and represents the maximum value currently possible. The value x'00' represents the fastest frequency supported and subsequent positive values are decreasing frequency in 16.667 MHz steps from the fastest frequency as a base.
8:15	Local Actual Pstate	Local Actual Pstate. Represents the presently operating Pstate value for this core. This value is less than or equal to the Global Actual Pstate. The value x'00' represents the fastest frequency supported and subsequent positive values are decreasing frequency in 16.667 MHz steps from the fastest frequency as a base. **Engineering Note:** For the POWER9 core, the Local Actual Pstate present in the core where the PMSR is accessed, also pertains to the three other cores associated with the Quad where the core exists.
16:23	Pmin	Pstate Minimum. Reads from this field return the presently established minimum Pstate for this core as set by the platform. This value can change autonomously based on the current policy in place and the physical constraints of the platform. The value x'00' represents the fastest frequency supported and subsequent positive values are decreasing frequency in 16.667 MHz steps from the fastest frequency as a base.
24:31	Pmax	Pstate Maximum. Reads from this field return the presently established maximum Pstate for this core as set by the platform. This value can change autonomously based on the current policy in place and the physical constraints of the platform. The value x'00' represents the fastest frequency supported and subsequent positive values are decreasing frequency in 16.667 MHz steps from the fastest frequency as a base.
32	PMCR Disabled	SPR-Based Energy Management Disabled. Reads from this field indicate whether the platform has disabled the PMCR SPR to control the core PState. 0 PMCR enabled. 1 PMCR disabled.
33	Safe Mode	Safe Mode. Reads from this field indicates whether the chiplet has been put into a fixed safe mode (frequency and voltage setting), where Pstate requests have been suspended due to errors or for externally forced reasons (such as, firmware updates). 0 Not in safe mode. 1 Safe mode engaged.
34	IVRM Allowed	Internal Voltage Management Allowed. Reads from this field indicate whether this system allows the chiplet internal voltage regulation to be enabled for localized voltage control. 0 iVRMs are never used for regulation on this system. 1 iVRMs are allowed to regulate the voltage on this system. **Engineering Note:** Generally, this bit reflects the enablement by platform firmware upon IPL. However, in PowerVM systems, this bit can change due to errors with the iVRM during run time (loss of reference voltage and internal errors). If these types of errors occur and checkstops are not produced, the platform disables the iVRMs and continues running. If checkstops are produced and error analysis indicates that the iVRMs were the cause, the platform can re-IPL with the iVRMs disabled as the means of recovery action.

Table 23-2. PMSR Description (Sheet 2 of 2)

Bits	Field	Description
35	IVRM Enabled	Internal Voltage Regulation Enabled. Reads from this field indicate whether the chiplet internal voltage regulation is active for localized voltage control. Note this is a snapshot taken when the PMSR was last written. 0 iVRMs are not currently regulating voltage for this core. 1 iVRM regulation is enabled for this core.
36:59	Reserved	Reserved for future enhancements. Set to all 0's.
60:63	Version	Version of PMCR (and PMSR) format supported by the current version of EnergyScale Firmware running on this system.

23.5.8.4 Power Management Memory Activity Register (PMMAR)

The PMMAR is not implemented on the POWER9 processor.

23.5.9 Architected Idle Modes (Stop States)

A new Stop instruction has been added to the POWER ISA for the POWER9 processor. The Stop instruction replaces the previously architected nap, sleep, and winkle (and unimplemented doze) instructions that are not supported by the POWER9 processor. This instruction works in conjunction with the per thread Power Save Status and Control Register (PSSCR). See the *Power ISA (Version 3.0B)* for details.

With this instruction, the operating system or hypervisor can stop a nonrequired thread from occupying the core pipeline. The hypervisor can disable the thread to free up core resources to the remaining threads or enable the core to switch SMT modes. The chiplet powered-off states can also be used in PowerVM-based systems to apply concurrent patches and to perform runtime array repair when required.

When all threads on a core enter the Stop state, the entire core enters into a Stop state; thereby, saving additional power. This enables varying levels of power savings, each with an increasing amount of power saved but higher latency to resume operation. Sixteen Stop levels can be requested, defined as four ranges of state loss each, with four levels of "deepness" encoded in the requested Stop level:

- Level 0 - 3: Lowest latency. There is no state loss when PSSCR[ESL] = '0'. This can be executed by the operating system. The SMT thread switch is selectable (via PSSCR[ESL]) when executed by the hypervisor. These levels can be configured to wake up to the next instruction or to SRESET when executed by the hypervisor (like previous generation idle states). An operating-system request does not involve the hypervisor. Timing facilities are maintained. The hypervisor can convert an operating-system request into an interrupt back to the hypervisor. Level 1 is equivalent to the legacy nap mode if executed by the hypervisor and doze mode if executed by the operating system. Level 2 clocks off the entire core, thus it is equivalent to the legacy fast-sleep mode.

  The POWER9 processor does not support level 3. Level 3 is reserved as a placeholder to additionally drop the core to V_{MIN} (minimum operational voltage) if iVRMs are enabled in future system designs.
- Level 4 - 7: Some hypervisor state loss is possible; therefore, its execution is only supported by the hypervisor. Timing facilities are maintained. Because PSSCR[EC] must be set to '1', these levels (like legacy stop states) always wake up to an SRESET vector and not the address of the exception that caused the wakeup. Level 4 is similar to the legacy deep-sleep state that powers off the POWER9 core for the chip, with the exception that timing facilities are preserved and the L2 cache also remains running.

- Level 8 - 11: Quad-level states. All hypervisor states, including timing facilities might be lost. Level 8 clocks off the powered-off cores' L2 cache so that it has no state equivalent to previous POWER processors. Level 11 powers off the entire Quad, which is comparable to the legacy deep-winkle state. Level 9 or 10 clocks off but does not power down the entire Quad, which is comparable to the legacy fast-winkle state and is not supported because the power and latency tradeoff does not warrant its use.
- Level 12 - 15: Reserved for future chip or system idle states.

Not all possible Stop levels are supported by EnergyScale firmware. If a nonimplemented Stop level is requested, the POWER9 processor enters the next lower implemented state.

Software requests a Stop state per thread. If any thread is still running, the core cannot enter a Stop state. The core only requests a Stop state based on the lowest Stop state of all its threads (for example, on an SMT4 normal core, if three threads request Stop state 15 and one thread requests Stop state 2, the core enters Stop state 2). The CME receives the Stop request and performs the entry (turn off the requested clocks and/or power). The CME only controls the core chiplet Stop transitions. In Stop states greater than seven, the CME must ask Stop-GPE (in the OCC complex) for assistance to perform Quad-level Stop states.

Idle state sequencing is almost completely managed by PPE microcode on the POWER9 core instead of hardware state machines.

23.5.9.1 Wake-Up Events

When PSSCR[EC] = '0', any interrupt causes an exit from the STOP state.

When PSSCR[EC] = '1', only interrupts enabled by PECE fields in the LPCR causes a STOP exit.

When PSSCR[ESL] = '1', any hypervisor state loss that is required for the core to resume execution is first restored by the hardware before an SRESET is seen by the hypervisor.

23.5.9.2 State Loss and Restoration

Only the hypervisor can initiate state loss with PSSCR[ESL] = '1'. The hypervisor is responsible for saving any nonhypervisor thread context (such as, GPRs, VSRs, FPRs) that must not be lost upon execution of the Stop instruction and then restore those values after wakeup. On the POWER9 core, the only state that can be lost for Stop levels less than four, when PSSCR[ESL] = '1' are the following SPRs: CR, FPSCR, VSCR, XER, DSCR, AMR, IAMR, UAMOR, AMOR, DAWR, DAWRX.

Note: Although some minimal hypervisor state (AMOR, DAWR, DAWRX) might be lost, the POWER9 core reports SRR1[46:47] as '10'; otherwise, the hypervisor cannot distinguish between states when the Timebase is preserved (Stop levels 4 - 7) versus lost and requiring restoration. This deviates from the description in the *Power ISA (Version 3.0B)*, which states that any hypervisor state loss should be reported as '11'.

Before executing the Stop instruction for levels greater than three, the hypervisor is responsible for calling a Stop-API supported by the power-management firmware for saving a subset of hypervisor SPR values necessary to properly execute the SRESET interrupt vector immediately upon wakeup. These values are restored during wakeup by the power-management infrastructure before the SRESET interrupt is taken by the hypervisor, appearing as if they were not lost. These registers typically include SPRs important to the initial processor context such as the HRMOR, LPCR, and HSPRG0. The PSSCR and LPCR are required to be saved by the hypervisor via the Stop API for Stop levels greater than three. Otherwise, after a special wakeup the following scenarios might occur:

- An unintended Stop11 can result while in the Stop state because PSSCR scan flushes to Stop level 15 by default.
- The wakeup conditions might no longer be honored.

Although not necessarily required for the hypervisor to resume operation, other registers that are supported by this mechanism include the HMEER, PMCR, HID, MSR, and DAWR. These SPRs, handled by the StopAPI, are only restored for Stop levels greater than three (for example, calling the StopAPI to save DAWR does not restore it for Core Stop levels less than or equal to three).

For Stop states less than eight, the timing facilites (TB, VTB, DEC, PURR, SPURR, HDEC) are preserved, including any state required to maintain operation of those facilities, (such as, RWMR, TFMR, CTRL). Additionally, the following POWER9 registers are also preserved for states less than eleven: DPDES, SPRC, SPRD, HMER, HMEER, PSSCR, PMSR, PMSCR, PMCR, L2QOSR.

For Stop states greater than seven, the timer facilities might be lost. Therefore, the hypervisor is responsible for restoring the timebase and associated timer-based registers after wakeup, as well as all other SPR states.

The CME is responsible for restoring the core and cache scan state and SCOM values, and for causing the core to restore the previously listed subset of hypervisor SPRs during the wakeup (via a special "self-restore" step after the core is powered on). System and host firmware is also required to call an SCOM restore API for any SCOM registers changed during runtime that must be restored on a core or cache power-on during Stop wakeup.

Table 23-3 summarizes by unit which functions in the Quad are clocked and powered off by the various Stop levels.

Table 23-3. Stop Instruction to Unit Mapping

Stop Level	Stop 0	Stop 1	Stop 2	Stop 4 and 5	Stop 8[1]	Stop 11
VSU, ISU	Instruction Stop	Clock off	Clock off each core ≥ Stop 2	Power off ≥ Stop Level 4		
IFU, LSU	–	–				
PC, Core EPS	–	–				
L2-EX0	–	–	–	–	Clock off if both corresponding cores in EX0 are ≥ Stop Level 8	Powered off if both core pairs (all four cores) are ≥ Stop Level 11
L2-EX1	–	–	–	–	Clock off if both corresponding cores in EX1 are ≥ Stop Level 8	
NCU-EX0, NCU-EX1 L3-EX0, L3-EX1 CME-0, CME-1 Quad EPS, DPLL	–	–	–	–	–	
SRR1[46:47]	'01' if PSSCR[ESL] = '0', else '10'			'10'	'11'	'11'

1. Currently, Stop 8 is not supported. However, it might be supported in future POWER9 designs.

Any Stop levels not listed in *Table 23-3* are rounded down to the the next smallest numbered level (for example, a Stop 9 request is rounded down to Stop 8).

As per the ISA, on wakeup from a core Stop state, SRR1[46:47] on every thread indicates the amount of architected state lost.

If Stop level ≥ 8, all processor states might be lost and SRR1[46:47] = '11'.

Else, if Stop level ≥ 4, or if PSSCR[ESL] = '1', SRR1[46:46] = '10' because the timing facilities are preserved, but other hypervisor state loss is allowed.

Otherwise, SRR1[46:47] = '01' because the system wokeup from Stop levels < 4 and PSSCR[ESL] was not set to enable state loss.

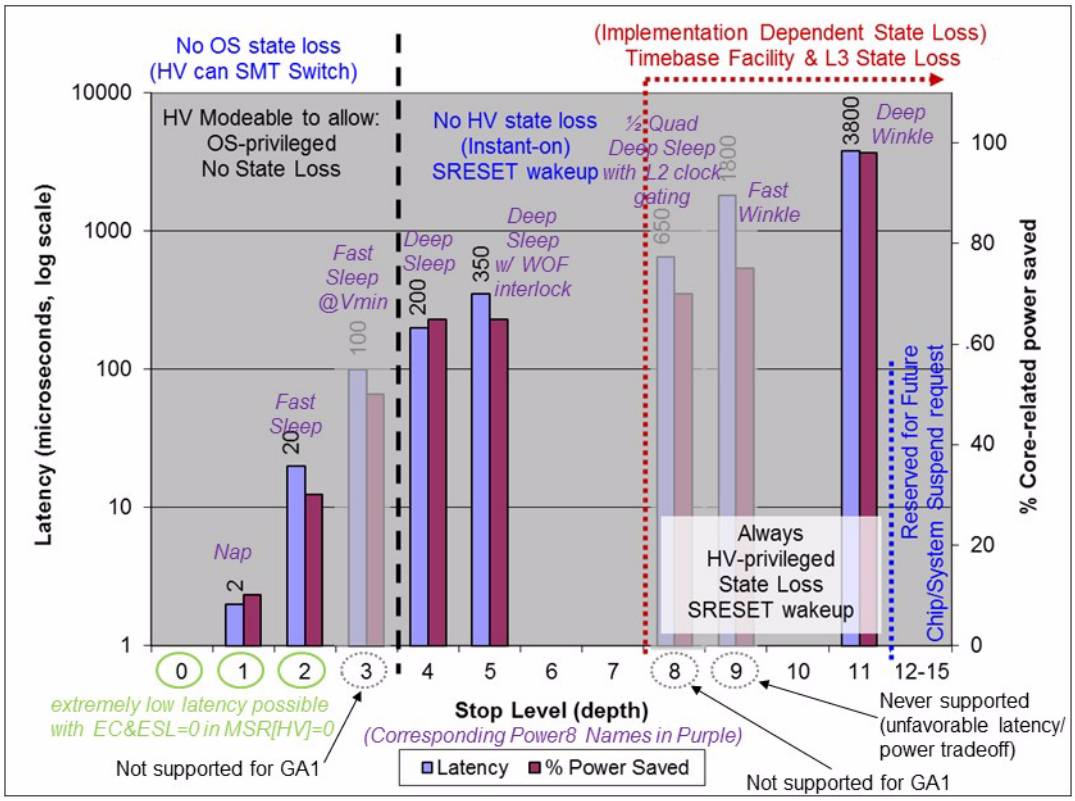
23.5.9.3 Auto-Promote of Stop Levels

The POWER9 cores does not honor the optional auto-promote feature provided in the PSSCR. The requested level field (PSSCR[RL]) in conjunction with PSSCR[PSLL] is always used. The values in the maximum transition level field (PSSCR[MTL]) and PSSCR[TR] are ignored in this design.

23.5.9.4 Latency and Power Savings in each Stop Level

Figure 23-5 shows the relative latency and power saving features of the various supported Stop levels. Latency is generally exponential with the amount of power saved; therefore, it is described with a logarithmic scale. These power and latency numbers are estimated targets and are subject to change.

Figure 23-5. Supported Stop Levels



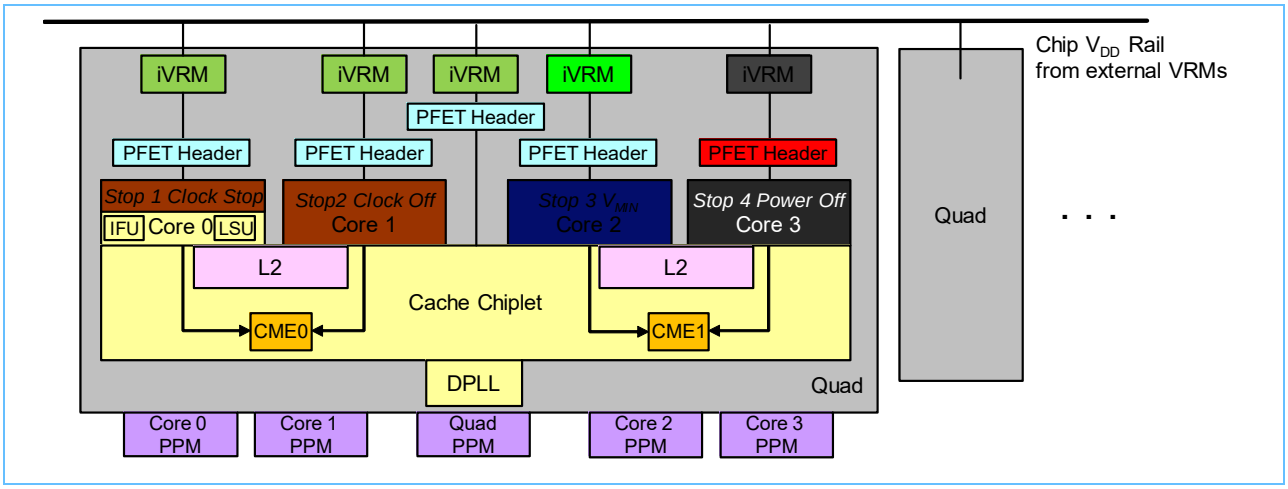
23.5.9.5 Stop Level Examples

Figure 23-6 through *Figure 23-9* on page 320 are graphical depictions of the various stop states, using the high-level Quad diagram in *Figure 23-3* on page 303.

Stop states 0 - 7 affect only the processor cores. Stop 0 only stops the core from dispatching instructions. Stop 1 clocks off a portion of the core, and Stop 2 clocks off the entire core. Stop 4 powers off the entire core, but leaves the L2 cache and timebase running. Stop 3 is an optional state that might in subsequent designs support lowering the voltage of a clocked off core to V_{MIN} to save additional leakage power.

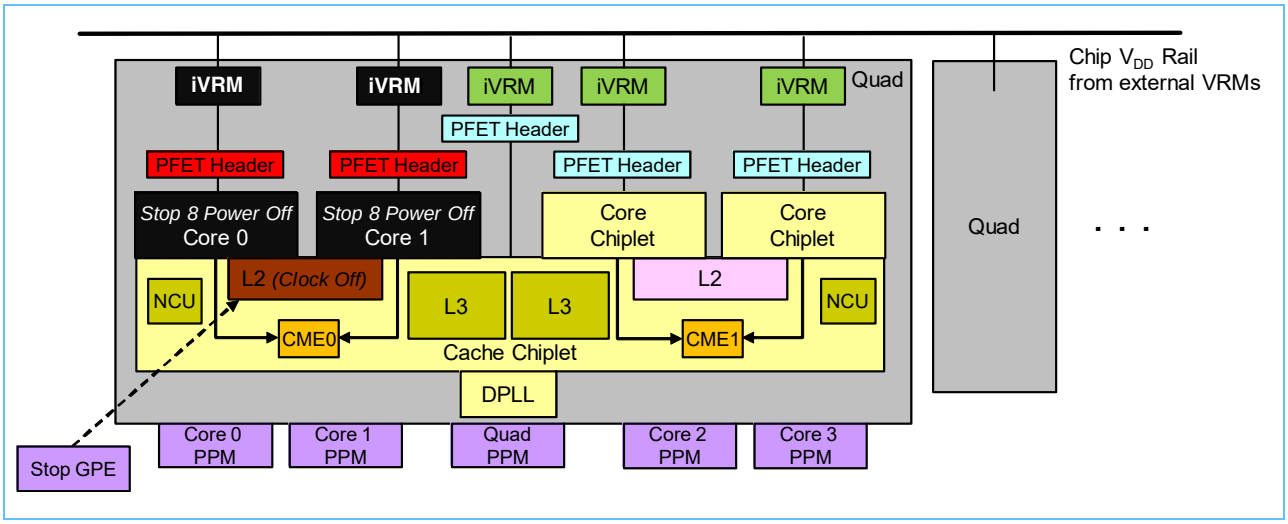
See *Figure 23-6* for more information.

Figure 23-6. Stop States (0 - 7)



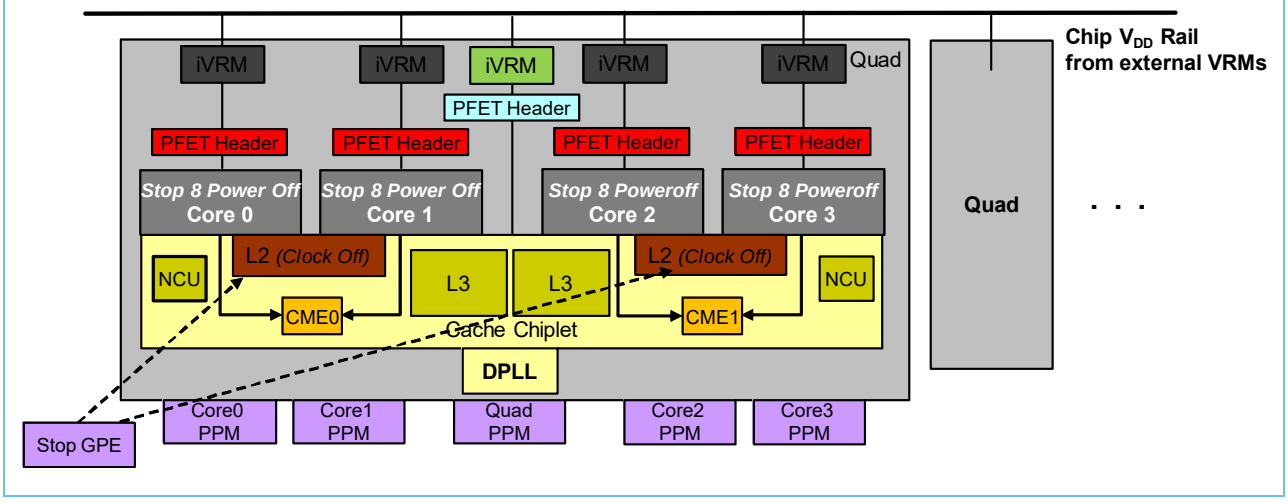
Stop level 8 is possible only if both cores in the pair are in at least level 8, such that the L2 cache can be clocked off. For this level, the Stop-GPE is invoked for assistance in controlling L2 unit clocks. The L2 cache unit is clocked off independently for each core pair. See *Figure 23-7* for more information.

Figure 23-7. Stop Level 8



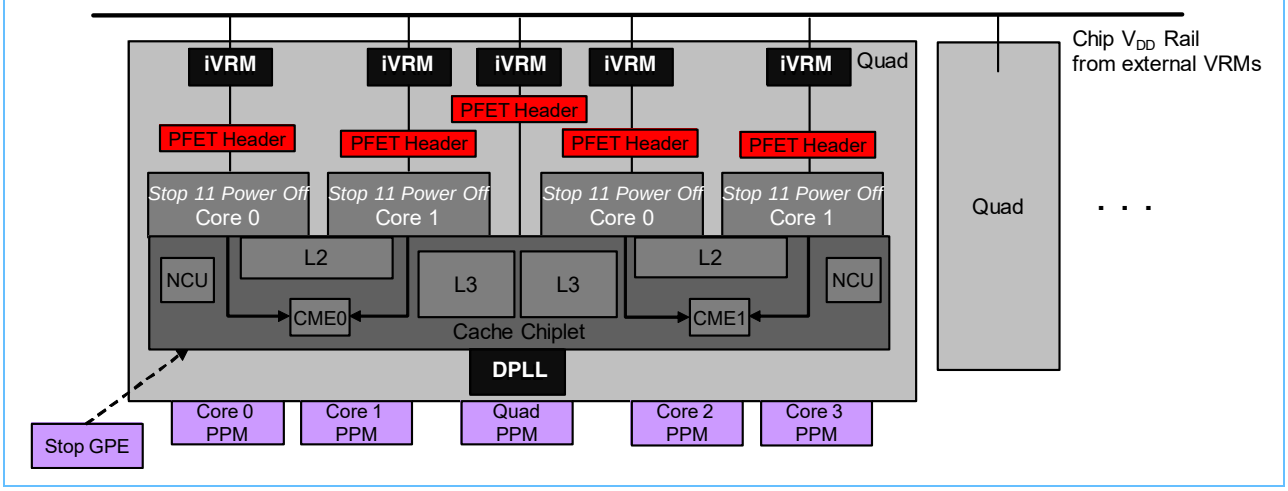
Both core pairs in Stop level 8 leave the L3 cache running even though both L2 caches are clocked off (see *Figure 23-8*).

Figure 23-8. Stop Level 8 (both Core Pairs)



The deepest implemented Stop level is 11. All five chiplets in the Quad are powered off. This implementation requires a complete IPL-like restoration of the Quad on wakeup, which includes restoration of the CME SRAMs (see *Figure 23-9*).

Figure 23-9. Stop Level 11



Note: Deconfigured or "bad" cores are considered to be in Stop level 15.

24. Specific Security Features

Secure boot and security capabilities in the POWER9 processor include those security enhancements made in legacy designs, as well as several additional features. The POWER9 design includes support for a secure and trusted boot of the hypervisor through a sequence of verification operations that extend the initial trust in the hardware and embedded firmware through the remaining firmware components and finally to the hypervisor itself. The security design for the POWER9 processor also includes hardware features to protect the processor state and customer data from unauthorized access once the system is up and running. The POWER9 design also adds dynamic root of trust for measurement (DRTM) support; a blacklist mechanism to control access to a sensitive processor state; and a capability to securely dump the processor state in the event of a checkstop.

Some members of the POWER9 processor family enable the optional secure memory facility (SMF). This function can be leveraged to create the protected execution facility (PEF). Combined with a trusted and protected execution ultravisor and co-requisite customizations in the hypervisor, the secure memory facility enables the creation of protected partitions whose memory or register state cannot be accessed by other partitions or by the hypervisor. The data integrity of the protected partitions are only dependent on the trusted ultravisor software constructs and guaranties provided by the secure memory facility in hardware. Hypervisor customizations are required for launching or operating with protected partitions. However, the aspect of data integrity is completely detached from hypervisor behavior; that is, as long as the ultravisor and hardware are implemented correctly, even a malicious hypervisor will not be able to extract any data from protected partitions.

24.1 Secure Boot

The goal of a secure boot is to extend the initial trust in hardware and secure code to each successively executed firmware and software component so that a chain of trust is established all the way up to the hypervisor. Trusted boot refers to a secure boot process in which secure measurements of code and configuration are maintained, such that a third party can determine the security of the system. Secure boot and trusted boot capabilities included in the POWER9 design are summarized in this section. The POWER9 processor also supports a DRTM, or late launch, capability that allows a system to rebuild a chain of trust after the initial one has been broken.

24.1.1 Secure Boot Sequence

The security threat model for POWER9 systems assumes that the development and manufacturing processes are secure. Therefore, the hardware itself is trusted. In addition, because it is maintained securely from the time a system is built, certain firmware is also trusted. That firmware is the security verification code (what was the security ROM code in POWER8) and the self-boot engine (SBE) code. These code components are trusted because their initial image is committed to a secure non-volatile memory (SEEPROM) during manufacture, and can only be modified by a secure process. The hardware and these firmware components comprise the initial root of trust for the system.

The flexible service processor (FSP), on the other hand, is not trusted because it is exposed to the network through potentially weak passwords and to incorrect or malicious use by a sysadmin. Any host code running on the system that is outside the chain of trust established during boot is also not trusted in our threat model, including operating systems and application code running within logical partitions. Software can implement processes to extend trust to additional code components, but that cannot be assumed in designing the hardware.

To extend the initial root of trust to the boot firmware and hypervisor, the following sequence is used:

1. FSP powers the system on, applies clocks, and then starts the SBE.
2. SBE initializes one core, then loads the bootloader and verification code from the SEEPROM.
3. SBE asserts the instruction_start signal to initiate load and verification of Hostboot on the master core.
4. Verification code authenticates the Hostboot code and then begins to execute it.
5. Hostboot code performs various boot procedures to initialize system components.
6. Hostboot code authenticates and then securely stores core power-restore images (winkle images).
7. Hostboot loads and authenticates the hypervisor, and then begins to execute it.

24.1.1.1 Code Authentication

Trust is extended from one code component to another by having the first authenticate the second. This is done through the use of cryptographic operations applied to the header and payload of a code container that contains the code to be authenticated. The container header includes a number of public keys and the signature of other parts of the container using those keys. The signature is an encrypted hash of some piece of data. It is authenticated by recomputing the hash, and then decrypting the signature and verifying that the result matches the hash.

The authentication procedure is summarized as follows. The container is loaded into memory. The code to be trusted is the payload of the container. It is signed using one or more (up to three) software private keys. The corresponding public keys are supplied in the header, as are the corresponding signatures. Each signature is verified by decrypting with the public key, hashing the payload, and comparing the results of these two operations. If all valid signatures are verified, the code is authenticated with respect to the software keys. However, the software public keys in the header must themselves be authenticated. The header segment containing those is signed by one or more hardware private keys, the corresponding signatures and hardware public keys also being included in the header. Those signatures are verified in a similar way. Finally, the hardware public keys are validated by hashing them and comparing to a hash that has been placed in the secure SEEPROM for this purpose. Successful validation of all signatures and keys is required for the code to be trusted and allowed to execute.

24.1.2 Trusted Boot

The term "trusted boot' is used to refer to a procedure that not only boots securely, but provides a means of securely recording the conditions and content of the boot procedure. That secure recording can later be used by the system to attest to the software that is currently running (and any that has been run since the last boot) so that a third party (called a remote party) can decide whether to interact with the system or not.

For example, a particular version of the hypervisor that is properly signed can be securely booted and used on many systems over some period of time. The trust in that code has to do with verifying its source, but does not imply correctness. If a security vulnerability is eventually discovered in that code, a new version is released to correct that vulnerability. Now, a third party that needs to provide sensitive information to the system will want to ensure that the system is running the new version of the hypervisor. Trusted boot of that hypervisor provides the system with the ability to securely attest to that hypervisor version number.

POWER9 systems use a Trusted Platform Module (TPM) version 2.0 to securely store the measurements associated with the various software components loaded during the trusted boot. The TPM contains a number of Platform Configuration Registers (PCRs) that can be used by firmware to securely record measurements. A measurement is a cryptographic hash of code or other data and is recorded in a PCR by an

"extend" operation to that PCR. The extend operation combines the new measurement with the previous PCR value to prevent insecure manipulation of these measurements. For this process to be secure, a reset to those PCR values must always be accompanied by a reboot of the system, and vice versa. In the POWER9 processor, this is guaranteed by the board wiring which connects the TPM reset pin and the processor standby_reset pin.

The POWER9 processor communicates with the TPM using an I^2C bus that is isolated from the FSP. Firmware uses that bus to send commands to the TPM, such as that for extending PCR values, and for obtaining a secure quote of current PCR values from the TPM. In this way, firmware maintains the PCR values based on configuration and code measurements that it takes during boot and beyond, and can provide those PCR values to a third party in a secure manner to support the remote attestation capability. The TPM can be used for several other security-related functions, including NV storage, random number generation, binding data and sealed storage. These additional features are available to software with no special hardware support on the processor chip.

To support trusted boot in a multi-node environment, a secure channel between nodes is needed early in the boot process so that nodes can exchange cryptographic material (nonces) that can later be used for inter-node authentication.

In the POWER9 processor, this secure channel is provided by the A-bus connecting each pair of nodes. Each A-bus link has four sets of mailbox registers associated with it. One of those is dedicated to this secure channel function, while the other three are available for general use. Hostboot uses the secure channel to exchange nonces, and then sets a security bit in pervasive to disable the channel. On the POWER9 processor, when this security bit is set, it prevents reads from the security mailbox register on all A-bus links. The security bit is sticky, such that once it is set, it cannot be reset by host code.

The sequence for making use of this secure channel is the following:

1. Train the A-bus links.
2. Use the security mailbox register on each A-bus link to trade nonces.
3. Assert the security bit to disable reads from the security mailbox on all A-bus links.
4. Set IOvalid for the A-bus links.

24.1.3 Dynamic Root of Trust for Measurement

A DRTM capability allows the system to re-establish trust after some untrusted code has been run, typically during the boot process. For example, the device firmware required to initialize the boot device might be resident on the device itself and not otherwise known to or trusted by the system. When this firmware runs during the boot sequence, the secure boot chain of trust is broken. The DRTM mechanism can then be invoked, causing the processor hardware to quiesce and then to come back-up through a partial re-boot in which trust is again established and extended to firmware and eventually to the hypervisor. The DRTM sequence can similarly be invoked sometime after the initial boot, when for example, a new Linux kernel is to be run such that its security does not depend on that of the previous version. As in the previous case, the security of the system after the DRTM operation should not depend on an assumption of trust prior to that operation.

24.1.3.1 DRTM Sequence

The DRTM procedure is implemented by a combination of new hardware mechanisms and changes to the boot firmware. The primary use case for DRTM is in booting a KVM-based system when untrusted device firmware is part of that boot sequence. In that case, the boot sequence proceeds as in a non-DRTM scenario,

including initialization of the processor and its interfaces, up to the point that the hypervisor is to be loaded. The untrusted device firmware is then executed to initialize the boot device. HAL_boot firmware then performs the following sequence to initiate the late launch.

Firmware Sequence Executed by HAL_boot Running on One Core

The firmware sequence executed by HAL_boot running on one core is as follows:

1. Load Linux/KVM into memory. This is the measured launch environment (MLE) to be authenticated by the authenticated code (AC).
2. Load the HAL_runtime into memory. This is the authenticated code. The AC must be loaded at a memory location known to the secure verification code that will authenticate it.
3. Put all other cores on this chip and on other chips in the system into a quiescent state (winkle mode).
4. Quiesce the NX, VAS, PHB, NPU/GPU, and CAPP/CAPI units on all chips.
5. Stop clocks to the OCC unit on all chips
6. Set the primary late_launch bit on this chip, and the secondary late_launch bit on all other chips. These are bits in the security register in the pervasive unit that when set, initiate the hardware DRTM sequence. When either of these late_launch bits transitions from '0' to '1', the local_quiesce_achieved and locality_4_access bits, also in the pervasive security register, are forced to '0'.
7. Put this core in winkle mode.

Hardware Sequence Executed by the SBE on Each Chip

The hardware sequence executed by the SBE on each chip is as follows:

1. When the SBE detects the assertion of either late_launch bit, it jumps to the appropriate entry point for late launch, as specified in the SBE base table in OTPROM. Hardware must ensure that the corresponding code is copied from the SEEPROM, to ensure a fresh copy in PIBMEM.
2. The SBE checks to see that all cores on its chip have been quiesced, by checking a pervasive bit for each that indicates it is in winkle mode.
3. The SBE code checks to see that the OCC clocks are stopped.
4. The SBE code asserts the quiesce_request SCOM bit in all PHB, CAPP, VAS, NX, and NPU units. With the cores quiescent, that bit remains asserted until the AC resets it. When the bit is asserted, the I/O and accelerators must go to a quiescent state and remain in the quiescent state until the bit is reset.
5. Now the SBE checks to see that the chip is quiescent. It checks the quiesce_achieved SCOM bit in each I/O and accelerator unit. It also checks again the status of each core.
6. Once the system has quiesced, the SBE checks to see that one and only one primary late_launch bit is set among all the chips. It also checks to see that all other chips have the secondary late_launch bit set. If these conditions are not met, the SBE forces a checkstop.
7. The SBE sets the local_quiesce_achieved bit in the pervasive security register, indicating that this chip is quiesced.
8. In a multi-chip system, the SBE now queries the local_quiesce_achieved bit of all other chips, and waits until all are set.
9. The SBE now sets the locality_4_access bit in the pervasive security register, allowing access to the locality 4 PCR in the TPM.

10. After the locality_4_access bit is set, each non-initiating SBE (having the secondary late_launch bit asserted) resets its late_launch bit and exits the late launch routine. The initiating SBE (having the primary late_launch bit asserted) resets its late_launch bit, scans one core to its IPL image, loads the DRTM security verification code from the SEEPROM, sets the NIA register in the core and asserts instruction_start.

The assertion of instruction_start causes the security verification code to start executing from an entry point specific to this DRTM sequence. This code verifies the AC that was previously loaded, and checks that it has the appropriate key code for an AC. After the AC is verified, the verification code jumps to it and the AC begins to execute.

24.2 Protection of Sensitive State

The secure boot process implemented on the POWER9 processor establishes a chain of trust from hardware and secure firmware to the running hypervisor. When a secure state is reached, it must be protected from potential attacks by untrusted code running on either host or auxiliary processors. Untrusted host code runs at a lower privilege level, and so can be controlled by higher privilege code such as the hypervisor. Untrusted code running on the FSP, which has side-band access to processor facilities through debug interfaces, must be explicitly controlled by limiting what it can access through those interfaces.

24.2.1 Blacklist for SCOM Write Access

The FSP uses the serial communication (SCOM) mechanism to read and write processor resources that it must control and monitor to properly boot and manage the system. However, allowing the FSP access to certain other facilities exposes the system to security attacks. The FSP must therefore request SCOM access to processor facilities such that untrusted access to sensitive facilities can be blocked.

The filtering of SCOM write requests from untrusted masters, including the FSP, is implemented in the POWER9 processor by the SBE using a blacklist approach. The blacklist identifies all sensitive facilities that must only be accessible to trusted masters.

In addition to filtering write requests, the SBE also filters read requests to a short list of facilities contained in a “read blacklist”. The facilities on this read blacklist are ones that produce a security-related side effect when read.

24.2.2 Secure Dump

The FSP is prevented by pervasive hardware from scanning the processor before clocks are started to boot the system, to prevent access to sensitive facilities that might open security holes. Scanning is also not possible while functional clocks are running. When clocks are stopped due to a checkstop, it is important to be able to scan out the system state for the sake of debug and diagnostics, but allowing the FSP to do so presents two potential security risks.

- Allowing the FSP to scan the data gives the FSP access to its content. This risk is considered small, as the data in the scan rings is not considered sensitive.
- Giving the FSP access to the scan rings allows it to manipulate the content of those rings. This is considered a significant risk, and so is addressed by the secure dump capability.

As with the dump procedure in previous systems, the secure dump is initiated by the FSP when a checkstop occurs. Instead of accessing the scan rings directly, which access is blocked by hardware, the FSP makes a request to the SBE for the dump of each ring. Using its hardware dump table, the FSP provides the length of the data it expects to receive. The SBE validates the request before performing the scan operation. If the request is valid, the SBE provides the ring contents to the FSP. The validation includes checking the ring length provided by the FSP against the SBE's own table of scan ring attributes to ensure that the FSP is requesting a full scan of the given ring.

24.3 Secure Memory Facility

The overall goal of the secure memory facility (SMF) is to provide register and memory isolation of a client compute stack (such as, a secure VM) from the rest of the untrusted system software stack (such as, an untrusted hypervisor or other untrusted VMs executing on the same machine). The *POWER9 Processor Programming Model Bulletin* provides this facility by implementing a privilege state bit defined as the Secure [S] bit in the Machine State Register (see Section 3.2.1 Machine State Register of the *POWER9 Processor Programming Model Bulletin*). The state bit is used in conjunction with existing privilege-level state bits to implement a higher privilege state known as the "ultravisor" state, which is defined as S = '1', HV = '1', and PR = '0'. This ultravisor state supersedes the hypervisor state (S = '0', HV = '1', PR = '0') in both system privilege and trust. Any software executing in this privilege state is known as the ultravisor and its correctness is critical to fulfill the goals of SMF function, such as register isolation. In addition, the host real address space is divided up into secure and nonsecure regions, demarcated by a designated host real address bit, thus providing memory isolation. The PEF requires hardware support in the form of the SMF, ultravisor software support, and certain customizations to the hypervisor to interact with the ultravisor to create and maintain protected partitions, which are also known as secure virtual machines. Some members of the POWER9 processor family implement the optional secure memory facility as described in the *POWER9 Processor Programming Model Bulletin*.

24.3.1 Protected Execution Facility in the POWER9 Processor

In the virtualized cloud-computing model, client-provided virtual machines (VMs) or logical partitions are often hosted by third-party open-source hypervisors such as XEN or KVM. Because a hypervisor can typically access any memory location in the host real address space, as well as observe any residual register state during a hypervisor call or hypervisor interrupt, all client data is effectively available to the hypervisor. Hypervisors are complex with many million lines of code. Over the years, many security exploitations have been found in hypervisors, whereby user code running under a VM breaks out into the hypervisor via privilege escalation attacks. Once they compromise the hypervisor from a VM, an attacker can compromise all client data from all VMs running in a server. Moreover, if the cloud provider is complicit, the hypervisor itself can be actively malicious and intentionally compromise client data.

The PEF, which leverages the SMF function, provides protection against this type of attack by creating the abstraction of a protected partition known as a secure virtual machine (secure VM or SVM), preventing the previously mentioned attacks from extracting any information. To achieve this, the ultravisor privilege layer intercepts any call or interrupt into the hypervisor, which enables it to clean up the VM's register state and provide register isolation before handing off control to the hypervisor. The ultravisor also establishes and manages memory regions (known as secure memory regions) that are exclusively accessible to trusted parts of the client computing stack (the applications and operating system belonging to a particular secure VM) and the ultravisor. This achieves memory isolation of trusted software components. The ultravisor system software component is considered trusted. This solution is also referred to as coarse-grain SMF, because this solution provides isolation at the granularity of an entire VM.

The security claims of the PEF are upheld by a combination of the following:

- SMF facility in hardware which provides:
  - Translation-based isolation in the memory management unit (MMU).
  - Interception of hypervisor interrupts to ultravisor privilege state in hardware.
- Ultravisor software guaranties:
  - Code procedural sequences to conceal state information of secure virtual machines on interrupt interception.
  - Page table structure layout for isolation of secure VMs.
  - Checks to be implemented in software as secondary mechanisms.

24.3.2 Deviations from the SMF Architecture Specification in the POWER9 Implementation

Certain architectural features described in the *POWER9 Processor Programming Model Bulletin* are not supported, or have restricted use in the POWER9 implementation of the secure memory facility, or deviate considerably from the description found in the *POWER9 Processor Programming Model Bulletin*. The following subsections describe these unsupported and restricted use features, as well as any notable deviations. Note that these deviations have no impact on data integrity guaranties of secure virtual machines provided by the PEF. The impact of these deviations only results in restricted software implementation choices or longer code sequences in the ultravisor/hypervisor software stack.

24.3.2.1 Unsupported Instructions: Processor Control Instructions Related to Ultravisor Doorbell Interrupts are not Available

Ultravisor privileged **msgsndu** and **msgclru** instructions described in Section 10.4 "Processor Control Instructions" of the *POWER9 Processor Programming Model Bulletin* are unavailable in the POWER9 implementations that support SMF. Any attempt to execute those instructions results in a system checkstop. It is recommended that ultravisor developers use **msgsnd** and **msgclr** instructions (which are available to hypervisor and ultravisor privilege states, and can cause hypervisor doorbell interrupts on targeted threads) in combination with an ultravisor call on the targeted threads (implemented by "**sc 2"** instruction) alongside software implementation-specific data structures to achieve the end goal of a directed ultravisor interrupt doorbell if necessary.

24.3.2.2 Implementation Restriction: Only URMOR[13:42] Bits are Implemented

Some members of the POWER9 processor family implement the secure memory facility described in the *POWER9 Processor Programming Model Bulletin*. Similar to the HRMOR register, only bits 13:42 of the URMOR Register are implemented. All other bits are reserved and return zero when read.

24.3.2.3 Implementation Deviation: Move to URMOR Instruction

Some members of the POWER9 processor family implement the secure memory facility as described in the *POWER9 Processor Programming Model Bulletin*. The ultravisor privileged **mturmor** instruction has a non-compliant implementation, where the instruction machine code of the **mturmor** instruction is added to the lower 32 bits of the source register value provided before the **mturmor** instruction is performed. Hence, the value to be placed into the source register must be the intended value of the URMOR value subtracted by the opcode for the instruction.

An example of this behavior follows:

If the instruction "**mturmor r31**" (big-endian machine code x'7FF97BA6') executes with a value of GPR31 = x'8FFF00000' (there is an extra bit set in the value given the intended value is x'8FFE0000'), the value to be moved into the URMOR will be x'7FF97BA6' + x'8FFF00000' = x'97FE97BA6'. Because only bits [13:42] are implemented, the actual value moved into the URMOR will be x'97FE00000'. Hence, in this case, the intended value to be moved into the register was x'8FFE00000'; therefore, the value (x'8FFE00000' - x'7FF97BA6') = x'87FE6845A' needs to be programmed into GPR31.

24.3.2.4 Implementation Restriction: UILE Bit is not Implemented and is a Constant Zero, Ultravisor Must Execute in Big-Endian Mode

Some members of the POWER9 processor family implement the secure memory facility described in the *POWER9 Processor Programming Model Bulletin*. The implementation-specific UILE bit, defined in Section 3.3 "Ultravisor Interrupt Little-Endian (UILE) Bit" of the *POWER9 Processor Programming Model Bulletin*, is not implemented in any special purpose register in the POWER9 processor family. The POWER9 processor assumes a fixed value of zero. Therefore, MSR[LE] is set to '0' whenever an interrupt that results in an ultravisor state occurs. In the POWER9 processor family, the ultravisor software must also ensure that the MSR[LE] is set to '0' when operating in the ultravisor state.

24.3.2.5 Implementation Restriction: SMFCTRL[62:63] Bits are Restricted to '10' Value Only

Some members of the POWER9 processor family implement the secure memory facility described in the *POWER9 Processor Programming Model Bulletin*. The implementation-specific bits [62:63] of the ultravisor privileged special purpose register SMFCTRL, described in Section 3.4 "Secure Memory Facility Control Register (SMFCTRL)" of the *POWER9 Processor Programming Model Bulletin*, must be set to '10' for correct operation of the secure memory facility when enabled by SMFCTRL[E] = '1' in the POWER9 processor. Other values can result in an erroneous system behavior.

24.3.3 Secure Memory Bit in System Memory Map

In the POWER9 processor family, host real address bit 15 is designated to differentiate secure versus nonsecure real-address access. Bit 15 in the system (host) real address is set to '1' to indicate an attempt to access secure memory.

Table 24-1. System Memory Map for 56-Bit System Address (8:63)

Bit 8:12	Bit 13:14	Bit 15	Bit 16:18	Bit 19:21	Bit 22:63
System Select	Memory Select	Secure Bit	Group Select	Chip Select	Chip Internal Address

24.3.4 Mandatory Software Procedures Followed by Ultravisor for Launching and Maintaining a Secure Virtual Machine

Because the security guarantees of the PEF are upheld by a combination of ultravisor software and SMF hardware extensions, it is imperative for the ultravisor to follow certain guidelines and implement certain checks. The ultravisor software layer can implement additional functions and checks, but the listed items are a bare minimum necessity.

Even though the architecture and implementation does not limit SMF to only radix translation, it is expected that SMF will only be used in radix translation mode in IBM products. Therefore, the following discussions only provide examples relevant to radix translation mode.

24.3.4.1 Essential Elements of Code Sequence to Convert a Non-Secure Virtual Machine into a Secure Virtual Machine

The actual code sequence to launch a secure VM by converting a non-secure VM might have many additional steps and actions in an actual implementation. However, the following steps must be present in some form. Supervisor software from a non-secure virtual machine invokes the ultravisor via an ultravisor system call (**sc 2**) with an indication of converting the non-secure VM into a secure VM. Ultravisor code prepares a VM to transform into a secure VM, which involves moving or copying all or some of its pages into secure memory and creating duplicates of all the necessary page tables in secure memory. Then, the ultravisor sets up USRR0/1 to the entry point of the secure VM: USRR0 = "EA of VM entry point",
USRR1(S, HV, PR) = '100', and performs an **urfid** instruction.

Table 24-2. Essential Elements of Code Sequence to Launch a Secure Virtual Machine

VM requesting conversion to secure VM makes an ultracall (**sc 2**) to the ultravisor.
After performing necessary validation on this ultracall, the ultravisor marks the partition table entry of this partition as secure and begins editing the entire translation structure of this partition.
The ultravisor moves the data pages and page tables of the non-secure VM into secure memory. This involves moving all the host real pages into secure memory and also moving all guest and host page tables associated with those translation into secure memory, as well as editing all necessary table entries to point to the new secure memory addresses.
Set USRR0 = <target instruction address in new secure VM>
Set USRR1(S, HV, PR) = '100'
urfid

24.3.4.2 Ensuring Isolation of Register State of a Secure VM from the Hypervisor

This section describes the procedure used to ensure that the isolation of the register state of a secure VM from the hypervisor by intercepting guest SVM hypervisor privileged interrupts by the ultravisor.

All hypervisor interrupts (hypervisor decrementer interrupt [HDEC], hypervisor instruction storage interrupt [HISI], hypervisor data storage interrupt [HDSI], hypervisor emulation interrupt, machine check, hypervisor mediated external) or a software-initiated system call to the hypervisor (**sc 1**) that causes hypervisor interrupts are intercepted by the ultravisor when SMF is enabled and are routed to an effective address of the associated interrupt + URMOR offset in ultravisor real mode. Note that on such an intercept, MSR[LE] is always set to zero, which implies all ultravisor code must be big endian.

Upon invocation of the ultravisor execution context via the previous mechanism, the following steps must be performed by the ultravisor to ensure that no residual register state from the secure VM is left over for the hypervisor to access.

1. Save all the context-specific registers that are considered sensitive data and should not be read by an untrusted hypervisor in secure memory. This includes general purpose registers (GPR), vector-scalar registers (VSR), floating-point registers (FPR), and any sensitive SPRs such as decrementer (DEC), and depending on the interrupt type SRR0/SRR1 or HSRR0/HSRR1. The register contents must be saved to secure memory (RA[15] = '1').
2. Populate the registers saved in step **1** with random values except for (H)SRR0/(H)SRR1, because these two registers are used to convey information regarding the interrupt to the hypervisor.

3. Encrypt the pages that might be required by the hypervisor and store the encrypted data in the normal memory area (RA[15] = '0').
4. Update (H)SRR1[S] = '1' before the interrupt is reflected into the hypervisor as if it came from the VM directly. The S bit in HSRR1 is used to indicate to the hypervisor that this interrupt originated from a secure VM and did not originate from a non-secure VM.
5. Setup USRR0/1 to reflect the interrupt to the hypervisor: USRR0 = "EA of interrupt", USRR1(S, HV, PR) = "010".
6. The ultravisor performs an **urfid** instruction to start execution in the HV state at the USRR0 address + HRMOR value.
7. The hypervisor handles the interrupt.
8. At the exit of the hypervisor, the handler tests the (H)SRR1[S] bit:
   - If (H)SRR1[S] = '1': perform an **sc 2** (ultravisor call) to return to the ultravisor, because the original interrupt came from a secure VM.
   - If (H)SRR1[S] = '0': perform a (**h**)**rfid** instruction to directly return to the non-secure VM.
   - Note that if a malicious hypervisor does not implement this step properly, it creates a disruption of execution but does not violate the memory and register state isolation guaranties.
9. Ultravisor restores the saved registers in step 1.
10. Check that (H)SRR0 (H)SRR1 are still the same as when the interrupt came to the UV (to prevent any misdirection attack from the hypervisor via returning to a different address in the secure VM).
11. Perform a (**h**)**rfid** in the ultravisor to return to the guest secure VM.

24.3.4.3 Ensuring Secure VM Translations for Secure Pages are Immutable by Hypervisor

To ensure that translations to secure memory pages can only be altered by the ultravisor, the following structures must be maintained within secure memory so that the hypervisor is required to make ultravisor system calls (**ucall**) to perform any changes. Hence, ultravisor software can perform the necessary checks. Hardware ensures that the root register for locating the partition table, namely the Partition Table Control Register (PTCR) is only ultravisor writable. The rest of the conditions must be enforced by the ultravisor software.

1. Partition table is maintained by the ultravisor and must be placed in secure memory.
2. Partition-scoped page tables (used during guest real address translation for the guest operating system) for a secure partition/SVM must be kept in secure memory.
3. Process-scoped page table (used during translation from a guest application, as well as when the guest operating system runs with translation enabled) must be kept in secure memory.

Alongside these restrictions, the ultravisor must also ensure that there is no code path in the ultravisor that can be called by the hypervisor to change the PTCR.

24.3.4.4 Ensuring Secure Memory Region Separation between Different Secure VMs

The ultravisor must ensure that during creation of a secure VM, all of the host real pages that are mapped to the secure VM belong within the bounds of the allocated area for that particular logical partition within the secure memory region. This is analogous to how hypervisors maintain partition separation.

In addition to original allocation during creation of the secure VM, the ultravisor must act on behalf of the hypervisor whenever a hypervisor fault (HISI, HDSI) occurs on the secure memory pages belonging to a SVM, because the host page tables for secure VMs are located in secure memory. The hypervisor can perform page table allocation and all policy decisions, but must use an ultravisor system call (**sc 2**) to ultimately update the page table entries in secure memory.

The ulravisor must check the following when such a page installation is taking place.

1. The LPID for which the translation is being installed is indeed a secure partition as per the partition table.
2. The secure host real address (RA[15] = '1') being installed in a leaf page-table entry must be checked against the data structures in the ultravisor, which contain the host real-size bounds of each secure partition to ensure that one secure VM is not overlapping with another secure VM.

24.3.5 Code Sequence to Change Value of URMOR Register

Table 24-3 shows a code sequence that the ultravisor privileged software can use to update the URMOR value in a core with multiple hardware threads and also potentially across the multiple cores of a system. The thread changing the value of URMOR is considered to be the master thread while all others are considered to be slave threads.

Table 24-3. Code Sequence to Set Value of URMOR

Master	Slave
Thread sync up point 1	Thread sync up point 1
Jump to instruction EA[0] = '1'	Jump to instruction EA[0] = '1'
Thread sync up point 2	Thread sync up point 2
Change URMOR (via **mtspr** on same core, SCOM write to other cores)	
Thread sync up point 3	Thread sync up point 3
isync	**isync**
slbia IH = x'7'	**slbia** IH = x'7'
isync	**isync**
Thread sync up point 4	Thread sync up point 4
Jump to instruction EA[0] = '0', new URMOR value will take effect now	Jump to instruction EA[0] = '0', new URMOR value will take effect now

24.3.6 Machine Check Conditions Specific to SMF

There are no SMF-specific machine check conditions in the POWER9 implementation.

25. Performance Profile

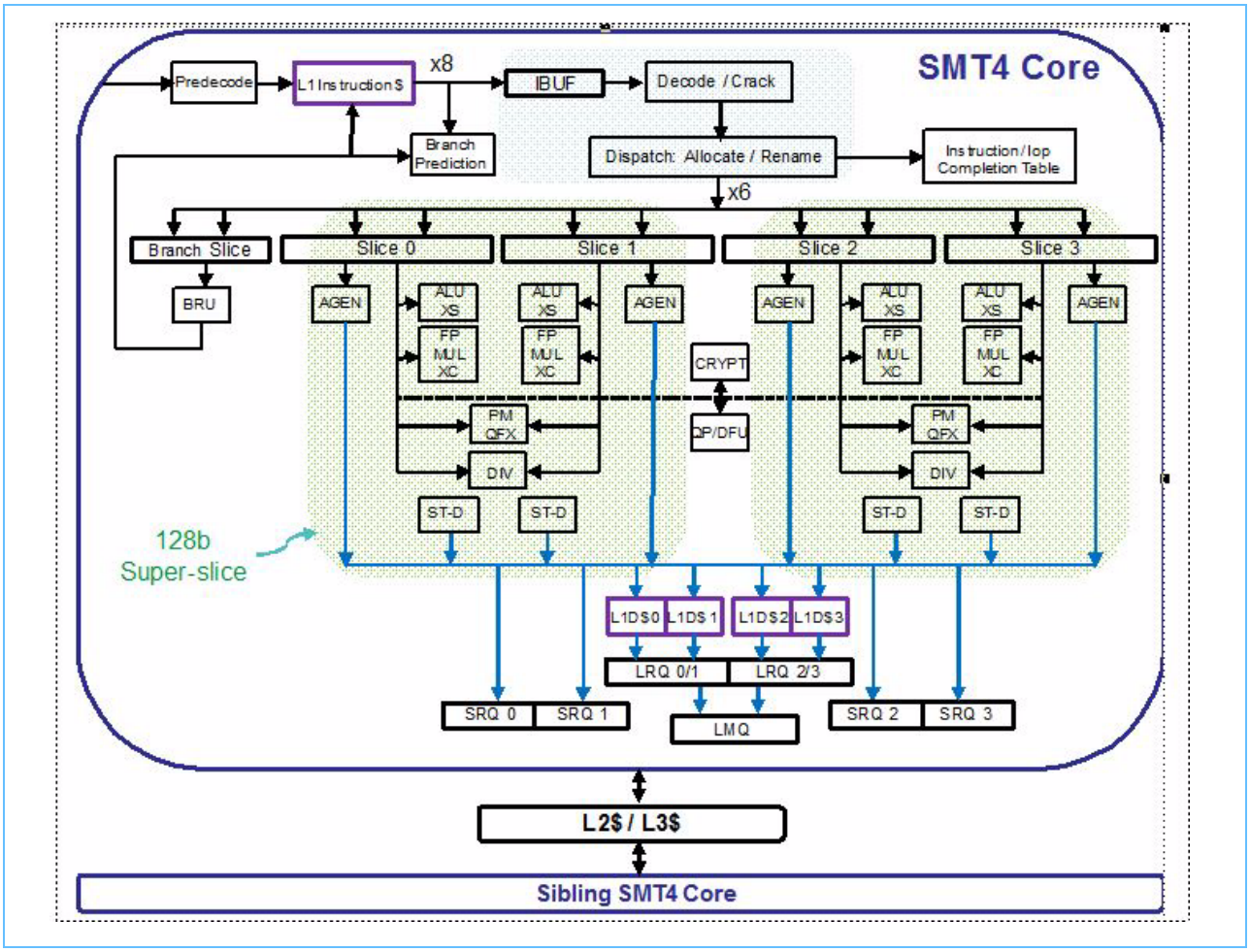
This chapter supplements the POWER9 design descriptions and architectural features supported with additional specifications relevant for software optimization.

25.1 Core

25.1.1 Microarchitecture and Pipeline Overview

Figure 25-1 shows the POWER9 core pipeline functions.

Figure 25-1. POWER9 Microarchitecture



The POWER9 core includes the following features, which are described in further detail in subsequent sections:

- Instruction fetch of up to eight instructions per cycle from the L1 instruction cache into the processor pipeline and buffered into a fetch instruction buffer (IBUF).
- Improved branch predictors for both direction and target.
- Instruction decode, cracking, resource allocation and dispatch of up to six internal-operations (iops) per cycle. *Figure 25-1* on page 333 shows this pipeline shaded in blue.
- Issue of up to nine iops per cycle.
- Execution across four execution slices, each slice contains a floating-point pipeline (DP) that supports multiply-add, divide, and square-root, as well as fixed-point multiply-add (MUL) and complex (XC) operations, a pipeline for arithmetic/logical unit (ALU) and simple (XS) operations, and 64-bit store data production (ST-D).
- Execution across two execution superslices, that each provide 128-bit dataflow through the execution slice pipelines and additionally provide: a permute (PM), a 128-bit fixed-point and BCD pipeline (DX), and a 64-bit fixed-point divide pipeline (DIV). *Figure 25-1* on page 333 shows each superslice (shaded in green); each superslice is composed of a pair of slices.
- Execution pipelines for cryptographic (CY), as well as decimal floating-point and quad-precision floating point operations (DFU).
- Execution of four load/store address generations per cycle, each broadcast to load/store slices (LS slices) together with store data from the execution slices. *Figure 25-1* on page 333 depicts load/store broadcasts outlined in blue.
- Execution across four LS slices providing independent access to one of four doubleword slices of the L1 data cache.
- LS slices supporting enhanced store forwarding and local re-issue from the load queue (LRQ) and store queue (SRQ) to handle cache misses, translation misses, and pipeline hazards.
- A connected cache subsystem providing a dedicated L2 and L3 cache region per pair of cores, as well as shared and local-castout (LCO) utilization of on-chip caches; providing up to 120 MB of on-chip L3 cache.
- Support for accessing the on-chip accelerator subsystem (NX) via the cut and paste architecture and for accessing an on-chip random number generator.
- Simultaneous multi-threading (SMT) that allows for up to four threads to share each processor core in one of three modes: ST (single-thread), SMT2 (up to two threads), and SMT4 (up to four threads).

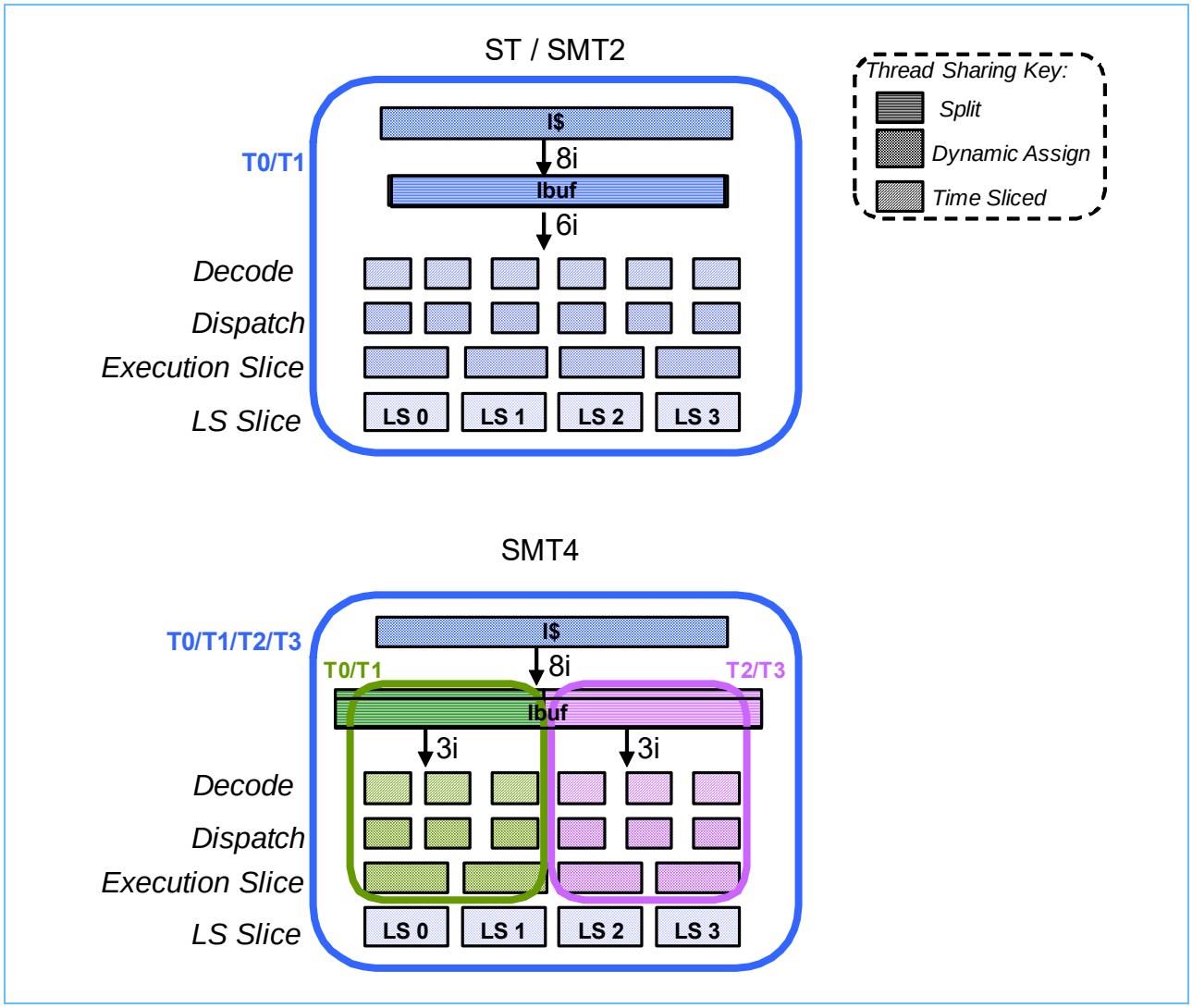
25.1.2 SMT Modes and Thread Count Sensitivity

Generally, all resources are shared within the pipeline between threads unless otherwise stated. Portions of the POWER9 pipeline and other resources are partitioned between threads depending on the active SMT mode.

The most significant partitioning related to threads occurs when more than two threads are active, placing the core in SMT4 mode. In SMT4 mode, the decode/dispatch pipeline, shown in the blue shaded area in *Figure 25-1* on page 333, is split into two pipelines, each pipeline is three iops wide and each pipeline serves two threads. The split decode/dispatch pipes each feed one of the two superslices, shown in the green shaded box in *Figure 25-1*, providing two execution slices for each pair of threads. The branch slice and LS-slices are shared between all threads.

Major partitioning of resources between thread modes is further depicted in *Figure 25-2*.

Figure 25-2. Partitioning of Resources Between Thread Modes

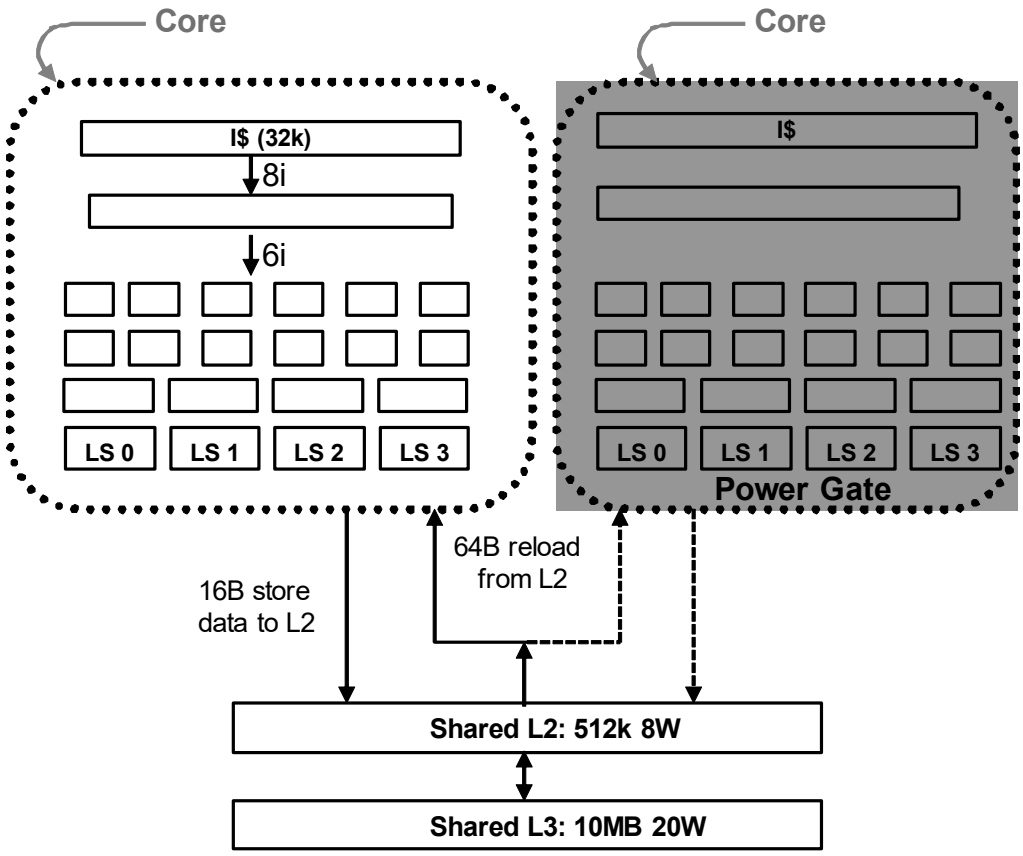


Additionally, the instruction completion table (ICT), which tracks iops in flight, is statically partitioned between two threads in SMT2 mode and between four threads in SMT4 mode.

Other pipeline and resource implications of SMT mode are discussed in subsequent sections.

As the number of active cores on the chip drops below the maximum, it is advantageous for the operating system and hypervisor to preserve only one active core per L2/L3 cache when possible. This has the advantage of providing the entire L2/L3 bandwidth and capacity to a single core as shown in *Figure 25-3* on page 336.

Figure 25-3. Single Core Active per L2/L3 Cache



25.1.3 Instruction Fetch

In each cycle, up to eight instructions (4 bytes each) are retrieved from the L1 instruction cache (I-cache), branch prediction is performed on all valid instructions, and an instruction prefetcher fetches lines speculatively to reduce I-cache miss occurrences. The compiler should align critical-code segments (targets, inner loops) on a quadword boundary.

The fetched instructions are packed eight at a time into the IBUF or bypassed, up to six at a time, into the decode stage.

In SMT2 and SMT4 modes, fetches are rotated between threads, round-robin style, until the IBUF is full for the thread. Requests for thread priority at fetch and decode are honored per specification of thread priority (see *Section 4 Power Architecture Compliance* on page 51). The POWER9 core also optimizes thread-priority selection to improve inter-thread efficiency during long latency stall events. By lowering thread priority during a low-productivity state, additional processing resources are provided to higher-priority threads.

Lowering the priority of a thread to "very low" causes the instruction fetch for that thread to enter a low-energy consumption state (low-power mode). In low-power mode, each thread performs one fetch of up to eight instructions approximately every 128 cycles. Low-power mode can also be used as a pacing mechanism for

thread execution. The context-synchronizing instruction, OR R31 R31 R31, has been implemented to enable entering low-power mode deterministically and executing instructions with paced execution. Use the following instruction sequence:

```
OR R31 R31 R31                          // Sets the CSI-bit and requests low-power mode
isync                                   // Causes a pipeline flush due to the CSI-bit being
                                           set to force entry into low-power mode

loop:
   <groups of eight instructions>       // Each group is executed approximately every 128
                                           cycles
   ...

OR R2 R2 R2                             // Return to normal priority
```

The I-cache can be accessed on any 16-byte quadword boundary and returns up to two 32 consecutive bytes per cycle, which results in fetching eight instructions for in-line code. Branch target fetches fetch 5 - 8 instructions in one cycle depending on quadword alignment, except when crossing a 128-byte cache-line boundary. Instruction fetches do not span a cache-line boundary in a single cycle.

25.1.3.1 L1 Instruction Cache

Each core contains a 32 KB, 8-way set-associative L1 instruction cache (I-cache). The I-cache is allocated in 128-byte lines with 32-byte sector valid tracking. The replacement algorithm is pseudo LRU. The I-cache is banked and allows concurrent reads and writes to different banks. The I-cache and associated effective address directory (EADIR) are accessed using EA bits 52:56 and tagged using EA bits 41:51. Entries in the I-cache directory are also tagged with MSR bits PR, LE, and HV. The EADIR is used to predict the way selection for each I-cache access. An EADIR hit that is later determined to be an I-cache miss adds approximately eight cycles to the base miss latency.

For each thread, aliasing can occur when a given EA(41:56) cache line is required for more than one real address, or a combination of MSR bits. Only one of the two combinations can be valid in the I-cache for a given thread at any one time, and an EADIR invalidate is required before fetching the other alias. Multi-thread EADIR aliasing results when two threads map the same EA(41:56) to two different real addresses or MSR combinations. When address aliasing occurs between threads, lines are brought in as private to the thread after first invaliding the existing entries.

On instruction fetches, effective address bits are used to index into the I-cache, the directory, and the instruction effective-to-real-address-translation (I-ERAT) table. The I-ERAT is a fully-associative 64-entry table and contains both the effective addresses and the associated real addresses. For an I-ERAT hit, the effective address of the instruction must match the effective address contained in the I-ERAT entry being indexed, and the I-ERAT entry must be valid. In addition, the IR, US, HV, and PR bits from the MSR at the time of I-ERAT miss are stored in the I-ERAT when the I-ERAT is loaded on an I-ERAT miss. These bits must match the corresponding bits in the MSR at the time of instruction fetch for an I-ERAT hit. The I-ERAT minimum miss penalty (assuming a TLB hit) is 18 cycles.

The I-ERAT directly supports 4 KB, 64 KB, and 16 MB page sizes. Other page sizes are stored in multiple entries using the next smaller supported page-size granule.

25.1.3.2 Instruction Prefetch

An instruction prefetch mechanism is used to fetch additional lines after an I-cache miss is detected. The instruction prefetcher uses the I-cache miss history to make decisions about the depth of prefetching on a per miss basis, ranging from 0 - 7 lines ahead. The mechanism is not active in SMT4 mode. The bandwidth of the instruction prefetcher is extended when the sibling core is inactive.

The banked cache design of the POWER9 instruction cache allows a concurrent read and write (as long as they reference different cache banks), so that writing prefetched lines into the instruction cache does not steal cycles from the fetching of instructions from the cache.

25.1.3.3 Software-Initiated Instruction Prefetch

An **icbt** instruction initiates the prefetch of a line into the L3 cache for use by the instruction cache. For processors on which the static hint bits are enabled, the static hint bits can be used to force an instruction prefetch by intentionally predicting down the wrong path. Note that the default setting for the POWER9 processor is to ignore the static hint bits. Because this method causes a pipeline flush, it should only be used when experimentation to demonstrate performance advantage can be performed on the target system. With this method, some instructions are speculatively executed or processed to some extent by the instruction fetch logic before they are discarded. The instruction in the (wrongly) predicted path can be used as a hint instruction to the memory subsystem. For example, software prefetching of instructions from location "Line_to_touch" can be initiated by forcing a branch misprediction as follows (the "a" bit in the bc instruction indicates "must agree with static prediction").

Short distance touches:

```
bc Line_to_touch                    // Static prediction taken, but CR bit is set to
                                       "not-taken"
```

Long distance touches:

```
bc Next                             // Static prediction not-taken, but CR bit is set to
                                       "taken"
b Line_to_touch                     // Initiate prefetch for cache line "Line_to_touch"
Next:...                            // Instructions in the actual instruction stream
```

This type of software prefetching is useful if the line to prefetch is in the L3 cache or beyond. Because of the high penalty for branch misprediction, it might not be beneficial if the referenced line is already in the L2 cache and even harmful if it is already in the I-cache. It is beneficial if the compiler makes special attempts to schedule code around such a branch that reduces the misprediction penalty. Attempts to reduce the forced branch misprediction penalty can be made by:

- Setting the CR bit used by the "bc" as early as possible.
- Scheduling such a branch in a code segment, where there are relatively few branches so that the branch does not wait too long in the branch issue queue behind other branches.
- Trying to overlap a likely D-cache miss with the forced branch misprediction.
- Scheduling such a branch after an existing long chain of flow dependency.

25.1.3.4 Branch Prediction

As instructions are fetched, they are scanned for branches. Up to eight branches are simultaneously processed by the branch prediction logic that predicts both the direction and/or target of the branches, depending on the branch type.

Branch direction prediction for conditional branches is performed using four branch history tables and a TAGE predictor. For nonrelative branches, target prediction is performed using a link stack for link returns and a count cache and pattern cache for other indirect branches. For relative branches, the target address is computed precisely. All conditional branches are predicted using the branch predictors. An incorrectly predicted branch results in a pipeline flush after the branch is executed.

The pipeline latency for a predicted taken branch from one of the predictors other than the TAGE is three cycles, and the latency for a TAGE prediction is five cycles. The POWER9 processor also includes a BTAC predictor that reduces the predicted branch taken latency to one cycle.

Static branch direction prediction is performed using hints as defined by *Power ISA User Instruction Set Architecture (Book I).* Branches that are statically predicted are treated as unconditional branches.

These predictors are described in more detail in the following subsections.

Branch Direction Prediction Using the Branch History Tables

The POWER9 core uses a set of four branch history tables (BHTs) to predict the direction of branch instructions, supplemented by a TAGE predictor. Each of the four tables has 8K entries:

- Local predictor: Predict branch taken based on history. Indexed by the branch address.
- Global predictor: Predict branch taken based on the path of execution to reach the branch. Indexed with a global-history-vector (GHV), formed from a taken branch history hashed with the address of the branch.
- Selector: Track which of the local or global predictor results should be used. Indexed by the global predictor index.
- Local selector: Tracks branches predicted well by the local predictor and prevents them from installing in (polluting) the global predictor. Indexed by the global predictor index.

Unconditional branches (including branches with the BO field set to '1z1zz') and statically predicted conditional branches (such as branches with the "a" bit set to '1') do not have an entry in the BHTs.

Branch Direction Prediction Using the TAGE

While the main BHTs are not tagged, the tagged geometric history length predictor (TAGE) has a 10-bit tag per entry to uniquely identify the optimal GHV length for a particular branch. The TAGE consists of four different global history predictors, each uses a different length of GHV. All four predictors are accessed concurrently and the longest predictor with a matching tag is used. The TAGE maintains a usefulness indicator with each table entry, when the indicator is showing poor prediction, the next longest table is updated.

Branch Direction Prediction Using Static Prediction

The POWER9 core normally ignores any software that attempts to override the dynamic branch prediction by setting the "a" bit in the BO field. This is done because historically programmers and compilers have made poor choices for setting the "a" bit, which limited the performance of codes where the hardware can do a

superior job of predicting the branches. However, the "a" bit can still be an important tool for certain performance-sensitive cases, such as those identified after careful analysis of branch misprediction on a POWER9 system. To force the honoring of the "a" bit, a change in firmware settings is required.

Incorrect setting of the "at" bits results in a pipeline flush; therefore, setting of the "a" bit should be avoided except in extraordinary cases where a thorough analysis has been performed.

The following cases are the only suggested uses of the "a" bit. When the "a" bit is set, then the "t" bit of the BO field specifies '1' for taken and '0' for not-taken. These uses are honored even when the hardware is configured to ignore the "a" bit.

- For the branches that close out a lock acquisition sequence, it is desirable to force the branch prediction to be *not taken*. This provides the best performance for the most common case where the lock is successfully acquired. Even if the lock is not successfully acquired on this iteration, it is still best for the branch predictor to behave as if the lock will be acquired in the next iteration.
  - Without static prediction, if the lock is not acquired in the first iteration, the branch history mechanism works to update the prediction to predict *taken*; that is, predict lock acquisition failure and cause more "**lwarx**" traffic for the next iteration. When the hardware detects a **l*arx** instruction near a static prediction, the static prediction is honored.

```
top:  lwarx
      add
      stwcx
      bc- top     <-- POWER9 core predicts this branch to be not taken, through
                      software directives that properly set the "a" and "t" bits.
```

Branch Target Address Prediction Using the Link Stack

The POWER9 core uses a link stack to predict the target address for a branch-to-link instruction that it believes corresponds to a subroutine return. By setting the hint bits in a branch-to-link instruction, software communicates to the processor whether the instruction represents a subroutine return, or a target address that is likely to repeat, or neither (see *Table 25-1*).

When instruction fetch logic fetches a branch and link instruction either unconditional or conditional but predicted taken, it pushes the address of the next instruction into the stack. When it fetches a branch-to-link instruction with "taken" prediction and with hint bits indicating a subroutine return, the stack is popped and instruction fetching starts from the popped address.

In the POWER9 core, the link stack is 64-entries deep and is split per thread mode: 32 entries per thread in SMT2 and 16 entries per thread in SMT4 mode.

Table 25-1 summarizes the handling of the POWER9 **bclr** and **bclrl** instructions.

*Table 25-1. Handling of **bclr** and **bclrl** Instructions*

Instruction	BH Field	POWER9 Design	Power ISA
bclrl	xx	If the branch is predicted taken, the link stack address is used as the predicted target address; however, the link stack is not popped.	Reserved.
bclr	00	If the branch is predicted taken, the link stack is popped and the popped address is used as the predicted target address.	The branch is a subroutine return.
bclr	01	If the branch is predicted taken, the target is predicted using the count cache. The count cache data and confidence fields might be updated when the branch is executed and resolved. No action is taken by the link stack.	Target address is likely to repeat.
bclr	10	Same as BH = '00'.	Reserved.
bclr	11	Same as BH = '01'.	Target is not predictable.

Branch Target Address Prediction Using the Count Cache

The count cache is used to predict the target address for branch to count (**bcctr**[**l**]) instructions and branch-to-link (**bclr**[**l**]) instructions which do not set the BH field to indicate a subroutine return and are therefore not predictable by the link stack.

The count cache predicts the branch target based on previous target addresses from previous executions of the same instruction. By setting the hint bits appropriately, software communicates to the hardware whether the target address for such branches are predictable using a cache. See *Table 25-1* and *Table 25-2*.

Note: The count caches can only be accessed for one branch per cycle. The **bcctr**[**l**] and **bclr**[**l**] instructions use the count cache, Therefore, no more than one such branch should be place per aligned 32-byte or octword block. Other branches that do not access the count cache are still predicted using the other predictors.

The POWER9 core maintains two count cache arrays:

- A global 512-entry array maintaining only the lower 32 bits of the target address (the upper bits are assumed to be unchanged). It is indexed similar to the global BHT using a GHV hashed with the instruction address.
- A local 256-entry array with a full 64 bit target address in each entry. It is indexed using the instruction address.

A 2-bit selector value is stored in the local array to select between use of the local and global caches.

*Table 25-2. Handling of **bcctr** and **bcctrl** Instructions* (Sheet 1 of 2)

Instruction	BH field	POWER9 Design	Power ISA
bcctr, bcctrl	00	If the branch is predicted taken, the target address is predicted using the count cache. Update the count cache when the branch is executed, if the branch is resolved as taken. For the **bcctrl** instruction, if the branch is predicted taken, push in the link stack the address of the next sequential instruction when the **bcctrl** instruction is fetched.	Target address is likely to repeat.
bcctr, bcctrl	01	Same as BH = '00'.	Reserved.
bcctr, bcctrl	10	Same as BH = '00'.	Reserved.

*Table 25-2. Handling of **bcctr** and **bcctrl** Instructions* (Sheet 2 of 2)

Instruction	BH field	POWER9 Design	Power ISA
bcctr, bcctrl	11	Same as BH = '00'.	Target is not predictable.

Branch Target Address Prediction using the Pattern Cache

The POWER9 core also predicts target addresses using a pattern cache, which predicts future targets based on a pattern relative to previous targets.

The pattern cache is a 256-entry table that is indexed and tagged by the previous instruction target address and is used to predict the lower 32 bits of the next target address.

When the pattern cache encounters a hit and the usefulness field is above a threshold, the pattern-cache target prediction is used instead of the address predicted by the count cache.

Branch Target Address Prediction Using the BTAC

The branch target address calculation (BTAC) is used to provide target fetch addresses for the current fetch group without wasting any fetch cycles due to a delay in a taken branch prediction. That is, the BTAC has a 1-cycle latency versus the 3 - 5 cycles of latency for other predictors.

The BTAC is only active in ST mode.

Obtaining the Next Instruction Address/BC+4 Handling

On the POWER9 core, the preferred method of obtaining the next instruction address is to use the **addpcis**; for example, **addpcis** with a displacement of 0.

Codes that instead use unconditional branches with the link bit set and a displacement of '4' to set the address of the next instruction into the link register are handled specially. Architecturally, these branches are taken and go to the next instruction. The hardware handles BCL + 4, with a BO = 20 (unconditional taken) as a special case. For this special case, the branch is always treated as *not taken* for fetch, resulting in no fetch penalty. When the special branch executes, it updates the link register, and does not cause a flush (even though fetch processed it as *not taken* and architecturally it was *taken*). The PMU sees these special branches as requiring direction prediction, direction predicted correctly, and branch not taken counts.

Branch Prediction Power Down

Branch history tables can be disabled based on sequences of consecutively predicted branches.

25.1.4 Instruction Decode and Dispatch Pipeline

After instructions are fetched from the I-cache, they are decoded into iops and then dispatched to slices for execution scheduling. The decode/dispatch pipeline can process up to six iops per cycle. In SMT4 mode, it is split into two independent pipelines, each handling up to two threads and each processing up to three iops per cycle.

25.1.4.1 Instruction Buffer

When the decode stage cannot accept all the fetched instructions due to the high fetch rate, or due to stalls in the pipeline, instructions are stored in the IBUF. This allows the instruction fetch and branch prediction to proceed during the pipeline stall condition.

Instructions that are not bypassed are written into the IBUF. Bypass from instruction fetch occurs even when instructions remain in the IBUF, such that six instructions are fed to the decode pipeline each cycle; either from the IBUF, fetch, or a combination of both. The IBUF holds 96 instruction entries and is partitioned per thread statically in SMT2 mode and SMT4 mode. Fetch for a given thread does not occur unless there are at least eight entries available in the IBUF.

25.1.4.2 Effective Address Tracking

The effective address table (EAT) stores branches and retains a correlation of branches to instruction addresses. The EAT hold 40 entries, 20 entries per thread in SMT2 mode and 10 entries per thread in SMT4 mode. A new EAT entry is consumed for each new 128-byte cache line that is accessed as part of the in-flight instruction stream and for each predicted-taken branch. Fetch is held for a thread when there are no remaining EAT entries for the thread, thus limiting the number of predicted-taken branches in flight to 40 for the core.

25.1.4.3 Instruction Decode/Cracking

Instruction decode maps instructions into iops.

Most instructions in the Power ISA are not cracked and are decoded into a single iop. However, some instructions are cracked or expanded into more than one iop. There are three categories of instruction cracking/expansion:

- 2-way crack: the operation is cracked in-line into two decode slots.
  - Cracking is done independently per split pipeline in SMT4 mode.
  - The cracked iops must decode together. For example, this might cause iops to shift decode to the following cycle if the first iop takes the last decode slot.
- 3-way crack: the operation is cracked into three decode slots that consume a decode cycle; that is, no additional instructions can be decoded in the same cycle.
  - Cracking is done independently per split pipeline in SMT4 mode.
- Microcode expansion: expanded instructions include those that crack into more than three iops or those that have a variable number of iops.
  - There is one microcode expansion engine per core, meaning only one expansion can be decoded at a time. This engine is shared between the two split decode pipelines in SMT4 mode.
  - The microcode expander produces up to three iops per cycle.
  - There is a 2-cycle decode startup penalty for each expanded instruction; that is, there are two cycles in the pipeline for which no instructions are decoded. In SMT4 mode, the decode penalty applies only to the split decode pipeline on which the expansion instruction is detected.

A listing of cracked and expanded instructions can be found in *Table A-1. Instruction Properties* on page 375, by examining the "Instruction Type" column.

When the decode/dispatch pipeline is empty, such as after a pipeline flush condition, the decode/crack pipeline bypasses two instructions, skipping one pipeline stage only if they do not require cracking or microcode expansion. If there were more than two instructions in the fetch group, or if the decode pipe backs up due to

downstream stalls, cracking, or expansion, the subsequent instructions go through the entire decode/crack pipeline without a stage bypass at the maximum rate. For example, up to six iops per cycle or three iops per cycle per pipeline half in SMT4 mode.

25.1.4.4 Instruction/IOP Completion Table

The instruction/iop completion table (ICT) tracks iops from dispatch through completion and is allocated at decode time. Each iop from a crack or microcode expansion consumes one entry in the ICT.

The ICT tracks 256 iops and is split for SMT: 128 entries per thread in SMT2 mode and 64 entries per thread in SMT4 mode.

25.1.4.5 IOP Dispatch

The iop dispatcher routes instructions to execution slices, the branch slice, and the ICT.

Resource availability per slice can limit which slices are dispatchable or the number of iops that can dispatch to a particular slice. Required resources for iops to dispatch to a given slice must be available in some cases in the cycle of dispatch, even if the specific iop does not absolutely require that resource. All iops dispatch as an execution slice iop with respect to resource requirements, unless they are executed by the branch (BR) unit as specified in the "Pipe Class" column of *Table A-1. Instruction Properties* on page 375. The required slice resources are as follows:

- Execution slice:
  - One issue queue entry available
  - One history buffer entry available
- Branch slice:
  - One issue queue entry available
  - One history buffer entry available
  - One link/count mapper entry available

When a single slice is busy, it does not preclude dispatch to other slices. If there are not sufficient resources in any of the slices for an iop, that iop will stall at dispatch for that thread until a resource becomes available.

Up to three iops can dispatch to each execution superslice (pair of slices) each cycle, with each slice receiving a maximum of two iops per cycle. Certain iops are precluded from dispatching as part of a 3-tuple to a superslice and can therefore limit the maximum dispatch rate in the cycle in which they are dispatched. See the "Tuple Restricted" dispatch rule in the following list.

The following list of iop attributes and dispatch rules provide additional requirements and behaviors for specific instructions. See the "Dispatch Rules" column of *Table A-1. Instruction Properties* on page 375 for which of these attributes are relevant for a given instruction/iop.

- Even slice ("E")- certain operations must be sent only to an even slice.
  - For example, this includes instructions that use the DIV (fixed-point divide) engine and SPR instructions.
  - Also consumes odd dispatch slice slot of the same superslice at dispatch
  - When a multiple of these instructions, such as a fixed-point divide, are scheduled in proximity and might be able to execute in parallel, it is ideal to have these instructions balanced between superslices. To achieve an optimal balance of these instructions between superslices, it is suggested to pair these types of instructions back-to-back.

- Vector ("V") - vector iops (128-bit operand) take only one decode and dispatch slot but are dispatched to both the even and odd slices of a superslice.
  - Both the even and odd slice of a superslice must have resources for dispatch.
  - Store-vector instructions are considered as vector for dispatch requirements, whereas load vector instructions are not.
- Paired ("P") - certain cracked and expanded iops are paired such that they must dispatch together to the same superslice.
  - Similar to a vector iop, a pair of slices on the same superslice must have sufficient resources for dispatch to occur.
- Tuple Restricted ("R") - certain iops preclude dispatching more than one operation per slice for the superslice to which they are dispatched. The following list shows the primary types of operations with this restriction:
  - Load vector operations, unless dispatched with a vector
  - Scalar operations with FPR/VSR targets.
  - Scalar operations with certain source operand attributes, includes most stores and three (GPR/FPR/VSR) source operations.

Iops are routed to the execution slices based on slice resource availability and iop requirements. The dispatcher generally rotates the distribution of iops to each slice, but might bias toward sending particular iops to the same slice or superslice.

The dispatcher can dispatch up to six iops per cycle, but some of these can be multi-routed to more than one slice, including 128-bit vector iops (vector dispatch rule) that go to two execution slices (one superslice), and **mf**/**mt** LINK/CNT/TAR that go to both the branch slice and an execution slice.

Load-vector iops consume only one decode and one dispatch slot and can dispatch up to two per superslice.

Branch Slice Iops

Each execution and branch slice can receive up to two iops per cycle, with the maximum total iops dispatched numbering six. In SMT4 mode, the branch slice can receive a maximum of one iop from each set of three decode pipes or one branch per two threads.

Operations that are routed to the branch slice include all branch iops, **mf**/**mt** LINK/CNT/TAR, **addpcis**, and svc.

Dispatch Interlock and Stop Conditions

Pipe drain conditions can stop dispatch for a thread and can cause a dispatch flush in SMT mode to clear the decode pipe to allow another thread to use the pipe. A dispatch flush removes iops from the decode/dispatch pipe and causes them to refetch into the IBUF until the stop condition is removed. See the Dispatch Interlock column of *Table A-1. Instruction Properties* on page 375.

A list of dispatch interlock conditions follows:

- **tlbie**, **ptesync**, **tlbsync**, **eieio** must wait for older loads and stores to drain
- **tbegin**: outer **tbegin** must wait for a previous transaction to complete
- **eieio**, **tsuspend**, **tresume**, **trechkpt** must wait for older operations in the pipeline to complete and drain
- Certain SPR access instructions are dispatch scoreboard checkers and must wait for certain older SPR writers (dispatch scoreboard writers) to drain the pipe

- Certain SPR access instructions must wait for the EAT to be drained

Dispatch Rules Summary

Basic rules are as follows:

- A maximum of six iops can dispatch per cycle.
- Up to three iops can be dispatched to each superslice (even/odd slice pair) each cycle.
- Each vector or even type operation consumes a slot for both slices of a superslice at dispatch.
- Each execution slice and branch slice can receive up to two iops per cycle.
- Dispatching an iop requires at least one issue queue entry and one history buffer entry.
- If it is a branch iop, one additional link/count mapper entry is required.
- NOPs count as one of six per cycle that can be dispatched.
- NOPs and branches consume at least one of the three execution superslice slots, unless there are no younger execution slice operations that can dispatch concurrently.
- The dispatcher rotates the distribution of iops to each slice subject to various optimizations.

Exceptions:

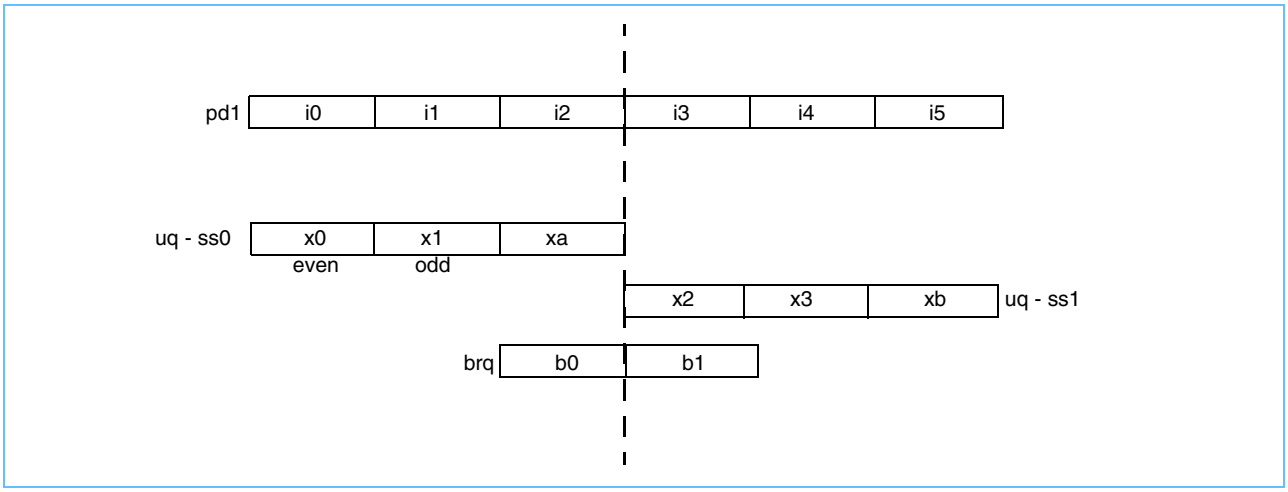
- In SMT4 mode, the branch slice can receive a maximum of one iop from each set of three decode pipes or one branch per two threads.
- If it is an even-slice instruction, it can dispatch only to an even slice (example: fx divide, spr instructions) but also requires that the odd slice be available at dispatch.
- If it is a vector operation (includes vector arithmetic or vector store iop), both slices in a superslice must be available for concurrent dispatch.
- Paired instructions must dispatch to the same superslice together.
- Certain iops cannot be part of the 3-tuple per superslice, for the superslice to which they are routed; limiting the superslice to receive one iop per slice. The operation set includes load vector (except with vector), scalar operations with three sources or special restrictions, and scalar writers of FPR/VSR registers.
- Dispatcher biases that are first to send iops requiring both slices of a superslice be together when allowed by program order.

Optimizing for Dispatch Rate

The available slice destinations are shown in *Figure 25-4* on page 347, where:

- The candidate iops for dispatch are shown in the register "pd1".
- x0, x1, x2, and x3 are the dedicated slice dispatch ports, where each corresponds to one of the four execution slices.
- b0 and b1 are dedicated dispatch ports into the branch slice.
- The third iop slots per superslice are labeled xa and xb, and correspond to superslice-0 and superslice-1 respectively. The xa and xb ports can be used to send an iop to either of the two slices of the superslice, but are restricted to iops with only two primary sources.
- A dotted line illustrates the split that occurs in SMT4 mode, where candidate dispatch iops are only dispatched to the slice dispatch ports shown on the same side of the line.

Figure 25-4. Available Slice Destinations



Optimizing Load-Vector Instructions Example

When interleaving load vectors with nonvector instructions, the compiler should bias toward dispatching load-vectors in pairs to maximize the dispatch rate. When interleaving load vector with vector instructions, optimal dispatch rate is obtained when the vector operations and load-vector operations are alternated in program order.

Similarly, the compiler should bias toward sending scalar iops in pairs or triplets (if they qualify for a 3-tuple dispatch to a superslice) when they are interleaved with vector operations to achieve the maximum dispatch rate.

Vector (SIMD) code that includes loads can benefit from pairing load vector operations together and interleaving with vector operations to maximize the dispatch rate. Consider the following example:

```
Scheduling for a loop with four 128-bit SIMD operations (V1,V2,V3,V4) and four 128-bit load-
vector operations (LV1, LV2,LV3,LV4):

Option A: V1 V2 V3 V4 LV1 LV2 LV3 LV4
Option B: V1 LV1 V2 LV2 V3 LV3 V4 LV4

Option A dispatches as follows in ST and SMT2 modes across a minimum of three cycles

Cycle 1: V1 V2 # note vector dispatch rate limited to 2 per cycle

Cycle 2: V3 V4

Cycle 3: LV3 LV4

Option B dispatches as follows in ST and SMT2 modes across a minimum of two cycles

Cycle 1: V1 LV1 V2 LV2

Cycle 2: V3 LV3 V4 LV4
```

Optimal scheduling in the example is shown in Option B. The same scheduling is preferred for SMT4 mode.

Note that vector and even routed iops should also be scheduled in pairs to maximize execution bandwidth for independent operations. Two independent fixed-point divide operations can execute in parallel on each core if they are routed to separate superslices. This is more probable when the two iops are scheduled back-to-back.

Optimizing Dispatch for Issue Latency

It is advantageous for dependent iops routed to the PM, DX, and DP pipelines to be routed to the same superslice, because this minimizes the issue latency of the dependent iop. See *Section 25.11 Instruction Properties* on page 372 for additional information.

Placement of dependent scalar DP iops back-to-back in the instruction stream increases the probability for placement into the same superslice. To increase the probability of being placed in the same superslice, dependent vector DP routed iops should not be placed back-to-back, because back-to-back placement increases the chance of not being dispatched to the same superslice.

25.1.4.6 Register Renaming

A register mapper renames target and source registers enabling out-of-order execution. Renaming is performed for GPR, FRP, VR, VSX, CR, as well as for fields of the XER and FPSCR. A history buffer (HB) provides a backing store for the previous architected mappings associated with each iop target register being executed and restores the register state after a pipeline flush.

The HB also serves as the backing store for active TM transactions. The HB tracks the pre-transactional registers so that they can be restored if a transaction is aborted.

The HB is kept per execution slice and handles the backup of all targets dispatched to the slice. The HB is partitioned into a primary and secondary level. The primary HB tracks previous targets until after the previous iop writing the same register is finished. The secondary HB tracks previous targets after the previous writing iop is finished. Therefore, codes with very high instruction-level parallelism should use multiple target registers and extend the same register target reuse distance to minimize chances of encountering HB resource limitations.

HB entries each track 64 bit operations, such that 128-bit target registers are backed up by two entries; for example, one in each of two slices.

The sizes for each HB are shown in *Table 25-3*. While there is ample room in the HBs to handle most transactions, if a program writes a very large number of registers in each transaction, they consume HB entries and can reduce the performance of the transaction, as well as degrade performance of other active threads.

Table 25-3. History Buffer Sizes

	Primary	Secondary	Total	Notes
64 bit GR/FR/VR/VSR	20 per slice x 4 = 80	96 per slice x 4 = 384	464	128 bit targets take two entries
CR/XER/FPSCR	12 per slice x 4 = 48	12 per slice x 4 = 48	96	

The LNK, CNT, and TAR Registers are renamed to a physical register pool using a mapper with ROB with 20 rename entries.

25.1.5 Iop Issue and Execution Slices

As iops are dispatched to a slice, they are held in an independent slice issue queue until all register sources and other dependencies have been resolved and they can be issued. Each of four execution slices has a 13-entry iop issue queue and the branch slice has a 16-entry iop issue queue, for a maximum of 68 issueable iops.

Each POWER9 core includes nine slice issue ports:

- One address generation (AGEN) issue per execution slice (×4)
- One execution (EXEC) issue per execution slice (×4)
- One branch (BR) issue from the branch slice (×1)

In each cycle, each issue port of each slice selects for issue of the oldest eligible iop, if any, from the set of iops held in its issue queue.

Iops that have 128-bit sources are issued synchronously from both slices of a superslice on the EXEC issue port. This includes iops that are vector, as well as paired and is indicated in the Issue Synchronization column in *Table A-1. Instruction Properties* on page 375. Examples of paired operations include cracked iops providing 128-bit operands to quad-precision operations.

Store iops are dual issued from a single-issue queue entry to both the AGEN and EXEC issue ports. Store-vector iops are both vector operations and dual issue operations, such that they are dual issued from one of the two slices and they perform a data only issue from a second slice. If the store iop's data source is ready, the store data can issue to the EXEC issue port as quickly as two cycles after the AGEN issue, or as quickly as three cycles after the AGEN issue for a store-vector.

Load/store iops issued from the AGEN ports are consumed by one or more LS slices for further processing. Once accepted, each LS slice takes over responsibility for executing the iop including the finishing of the iop, as well as the handling of all execution hazards encountered thereafter.

A data and address recirculation queue (DARQ) is provided to stage load/store addresses computed after AGEN issue and store data computed after EXEC issue, as needed, en route to each LS slice. The DARQ allows for iops issued for processing in LS slices to be released quickly from the issue queue.

25.1.5.1 Load/Store AGEN Issue

The AGEN port handles iops routed to the following pipelines (see Pipe Class in *Table A-1. Instruction Properties* on page 375):

- LD: load iop address generation, headed to load/store pipeline within LS slice
- LD2: +1 cycle to execute versus a normal LD iop
  - Includes load string indexed expansions
  - Includes **lvx**, **lvxl**, **lvebx**, **lvehx**, **lvewx**, when their computed EA(60:63) ≠ 0
- LD3: +3 cycles to execute versus a normal LD iop
  - Includes **lq**, **lfdp**, **lqarx**
- ST: store iop address generation, headed to load/store pipeline within LS slice
- ST2: +1 cycle to execute versus the normal ST
  - Includes store string indexed expansions
  - Includes **stvx**, **stvxl**, **stvebx**, **stvehx**, **stvewx**, when their computed EA(60:63) ≠ 0

An AGEN port iop is eligible to be issued in the next cycle on the AGEN port, if in the current cycle:

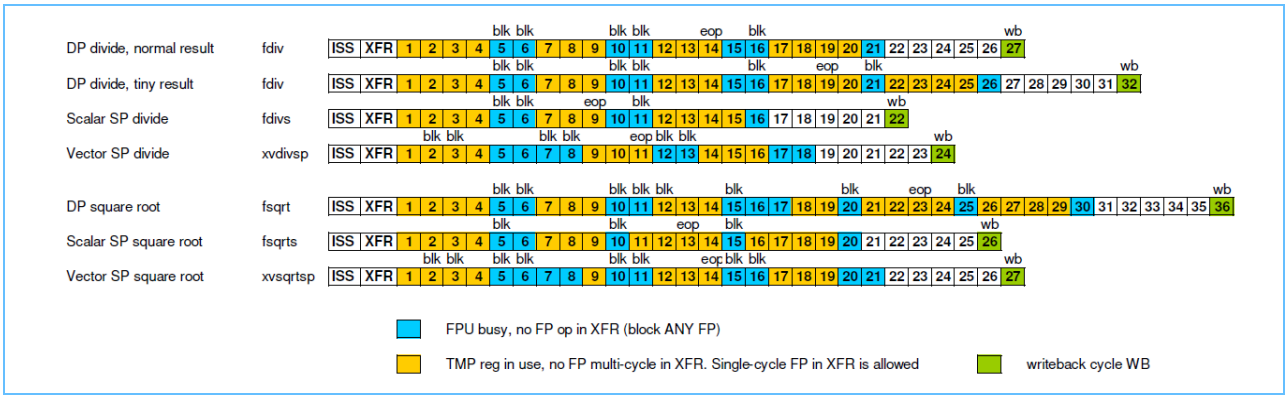
1. Its sources are ready.
2. It has not been flushed.
3. It was marked as an NTC issue (see *Table A-1. Instruction Properties* on page 375), the next older ITAG has completed.
4. DARQ has available entries to hold the AGEN result.
5. If it is a load and if there is an older store on which it is dependent because of LHS/SHL optimization, then that store agen has been launched in the LSU.
6. The same instruction was not issued in the current cycle.

25.1.5.2 EXEC Issue

The EXEC port handles iops routed to the following pipelines (see Pipe Class in *Table A-1. Instruction Properties* on page 375):

- ST/ST2 data pipeline - per slice (4)
  - store data (64 bit/128 bit) issued to the LS slice store queue (STQ)
- ALU/ALU2 pipeline - per slice (4)
  - ALU: 2-cycle arithmetic/logical iops
  - ALU2: 3-cycle arithmetic/logical iops
- DP/DP-XC/DP-MUL pipeline - per slice (4)
  - DP: double-precision iops
  - DP-XC: vector complex iops
  - DP-MUL: GPR target: fixed-point multiply iops (MUL)
  - Supports multi-cycle operations with interleaved issue. In *Table A-1. Instruction Properties* on page 375 note the Busy cycles for a count of cycles in which pipelined operations cannot issue while the multicycles operation is in progress. *Figure 25-5* on page 352 shows the interleaving opportunities for various length multicycle floating-point iops.
- PM/DX pipeline - per superslice (2)
  - PM: (128 bit) permute iops
  - DX: (128 bit) fixed-point and BCD iops
  - Operations issue from both slices synchronously
- DIV/SPR pipeline - per superslice (2)
  - DIV: fixed-point divide iops - entries can have a note indicating that the minimum latency is for early-out scenarios, such as for a power of 2 operand or for a value of '0' operand.
  - SPR: used for move-to special purpose register
  - Multi-cycle operations only
- CY pipeline - per core (1)
  - CY: cryptographic iops
- DFU pipeline - per core (1)
  - DFU: decimal floating-point iops and quad-precision floating-point iops
  - Multi-cycle operations and pipelined operations
- SPR - per core (1)
  - SPR-CTR/LR: move to CNT/LR/TAR registers
  - SPR: move to other SPR registers

Figure 25-5. Double-Precision Pipeline Multicycle Busy versus Issueable Cycles



An EXEC port iop is eligible to be issued in the next cycle, if in the current cycle:

- Its sources are ready.
- It has not been flushed.
- It was marked NTC issue at dispatch, then the next older ITAG has completed.
- It is a store data, then DARQ has room to accept the store data.
- There is not a higher latency operation already in the pipe (already issued from the same slice) that will produce the result in the same cycle as the one that is trying to get ready to issue in the next cycle.
  - There is only one writeback port per issue port such that as iops of different latencies are executed, only iops which will not produce writeback results at the same time can be chosen for issue.
  - The issue selection policy can schedule an iop of compatible length in lieu of an older iop with incompatible length.
  - The issue selection policy employs an arbitration policy for iops of different execution length and relative age.
- There is a long latency iop in the pipeline issued from this slice, such as floating-point divide (see *Table A-1. Instruction Properties* on page 375 for execution latencies over 13), then there is a cycle where issue is blocked for other iops to allow dependent iop wakeup.
- For the iop's execution pipeline, there cannot be a non-fully-pipelined operation causing a busy of the pipe; for example, DP/DP-XC pipe.
  - Non-fully-pipelined iops are identified in *Table A-1. Instruction Properties* on page 375 as those iops with busy cycles greater than one.
- If it is a non-fully-pipelined iop (see *Table A-1. Instruction Properties* on page 375 for instructions with busy cycles greater then one), there is not another non-fully-pipelined iop executing on the same pipeline; for example, DP/DP-XC pipe.
- If the same instruction was not issued in the current cycle.

25.1.5.3 Branch Issue

The BR port handles iops routed to the following pipelines (see Pipe Class in *Table A-1. Instruction Properties* on page 375):

- BR: per core (1)
  - BR-CR - CR dependency feeding branch pipe.
- SPR: per core (1)
  - SPR-CTR/LR - move from CTR/LR/TAR/NIA

A BR port iop is eligible to be issued in the next cycle, if in the current cycle:

- Its sources are ready.
- It has not been flushed.
- It was marked NTC issue at dispatch, then the next older ITAG has completed.
- It is an SPR instruction, then it has arbitrated successfully for a slice writeback port.
  - SPR-CTR/LR/TAR/NIA results use a slice writeback port and steal the port as long as there is not already a longer latency result scheduled for writeback in the same cycle.

25.1.5.4 Execution Pipeline Issue to Issue Latencies

Iops with register dependencies become eligible for issues as the register results are written back, or 1 - 2 cycles before depending on the respective iop pipeline. The pipeline-to-pipeline latencies for dependent issue between each execution pipeline are shown in *Table 25-4* on page 354. When forwarding data between PM, DX, or DP pipelines in a different superslice, an additional delay of 1 - 2 cycles occurs, depending on the specific pipeline, as shown in the +SS column of *Table 25-4*.

The POWER9 core is optimized to exchange data between various data types and registers providing ALU latency to exchange data between register types including GPR, FPR, VR, VSR, CR, XER; allowing for optimal movement via register in place of exchanges through storage. This has the effect of reducing the overhead for spilling registers, such as between GPR and FR.

Additional latency might be required for dependent iop issue in the following cases:

- Iops with a synchronous superslice issue ("S") subject to a +1 cycle issue-to-issue delay if they are dependent on the source iop, which did not have a synchronous superslice issue. For example, a 1-cycle delay might apply to an **mtvsrd** feeding an **xvadddp**, but would not apply for an **mtvsrd** iop feeding an **xsadddp**. This additional delay might be hidden by the processor by converting nonsynchronous issue operations to have the same issue pipeline as synchronous issue operations. When this occurs, the additional latency between an **mtvsrd** feeding an **xvadddp** would not be realized. However, in this mode loads feeding the nonsynchronous issue operation experience the same latency as feeding synchronous operations, see *Section 25.1.7.8 Load-to-Use Latency* on page 359.
- An additional issue delay of +3 cycles can be encountered when forwarding CR, XER, or FPSCR results between instructions. This delay is captured in the "Additional Latency for CR/XER/FPSCR/VSCR Source" column of *Table A-1. Instruction Properties* on page 375. Note that XER has a grouped field, CA/CA32/OC. For grouped fields, this delay applies when the producing iop is writing a field within the same group as the source field. See *Move-To and Move-From FPSCR* on page 354 for additional details related to latencies incurred for FPSCR exception summary, "sticky" field dependencies.

Table 25-4. Issue-to-Issue Latencies between Execution Pipelines

Consumer: Producer	WB	AGEN	ALU/ ALU2	PM/ DX	+SS	DP	+SS	DP-XC	+SS	MUL	+SS	DIV	CY	DFU	BR-CR
ALU	2	2	2	2		2		2		2		2	2	2	2
ALU2	3	3	3	3		3		3		3		3	3	3	3
PM	3	-	3	3		3	+1	3		-		-	3	3	3
DX	3	-	3	3		3	+1	3		-		-	3	3	3
DP	7	-	7	6	+1	5	+2	6	+1	-		-	7	7	7
DP-XC	7	-	7	7		7		7		-		-	7	7	7
MUL	5	5	5	-		-		-		5		5	-	-	5
DIV	12	12	12	-		-		-		12		12	-	-	12
CY	6	-	6	6		6		6		-		-	6	6	6
DFU	12	-	12	12		12		12		-		-	12	12	12
BR-CTR/LR/NIA			6							6		6			

Move from LNK, CNT, and TAR Registers are executed in the branch slice and have a latency of six cycles. These iops borrow a write-back port from an execution slice and therefore, consume an issue/writeback slot from the execution slice similar to an ALU operation. Move to LNK, CNT, and TAR Registers are executed from the execution slice and have a latency of five cycles. Other move from and SPR instructions are executed by one of two pipelines:

- LD pipeline: iop issue to dependent iop issue = 14 cycles
- SPR pipeline: iop issue to dependent iop issue = 12 cycles

25.1.6 Iop Execution

25.1.6.1 Execution Pipeline Hazards

XER-SO

The processor core speculates that the XER[SO] bit will not change during execution of operations; for example, that the SO bit is rarely cleared. When a change to SO is detected, the instruction that changes SO is flushed from the pipeline once it becomes NTC and is re-fetched for execution.

Move-To and Move-From FPSCR

The processor core manages a mode that speculates on the value of the FPSCR exception/sticky bits. The core favors speculation that sticky bits are not changing during the pipeline lifetime of sticky field readers, including certain move-from FPSCR instructions as well as certain move-to FPSCR instructions, see the iops marked with "F" in the "Next to Complete" column in *Table A-1. Instruction Properties* on page 375. This speculative FPSCR mode allows these instructions to execute without delay in an out-of-order fashion. When sticky field changes are encountered by these iops, they can cause a pipeline flush and the core will enter a nonspeculative mode for these iops. In the nonspeculative mode, the iops execute when they are next-to-complete. The processor maintains the nonspeculative mode while sticky changes continue to be observed.

Note: Lightweight move-to FPSCR instructions that only modify the control field of the FPSCR including **mffscrn**, **mffscrni**, **mffscdrn**, **mffscdrni**, **mffsl**, as well as **mtfsb0**, and **mtfsb1** that only update the control field, will never cause a flush and can always execute out-of-order.

25.1.6.2 FPR Result Forwarding Restrictions

The internal dataflow was optimized for common cases of data type exchange. Certain internal data type representations are not eligible for fowarding. If forwarding occurs due to an instruction dependency, a flush is generated. Avoid the following dependent scenarios:

- Convert float-to-integer feeding a floating-point input
- Floating-point output feeding convert integer-to-float
- 64-bit floating-point result feeding a 32-bit floating-point input operand; for example, **xvadddp** feeding **xvcvspuxds**
- 32-bit floating-point result feeding a 64-bit floating-point input operand; for example, **xscvdpsp** feeding **fadd**

25.1.7 Load/Store Processing

Load and store iops are broadcast to the four LS slices on address and store-data buses. Each iop is received by 1 - 3 of the doubleword (DW) aligned slices based on the computed operand effective address (EA) and length of the load or store operation as shown in *Table 25-5*. Each doubleword slice starts at one of four values of EA bits 59:60, each specifying one of slices 0, 1, 2, 3, and includes bytes from EA(61:63) = 0 through EA(61:63) = 7.

Table 25-5. Slices per Load/Store Operation

Operation Length	Fits in Aligned Doubleword?	Aligned to Doubleword Boundary?	Number of Slices
1 Byte	-	-	1
2 - 8 Bytes	Y	-	1
2 - 8 Bytes	N	-	2
16 Bytes	-	Y	2
16 Bytes	-	N	3

The "Fits in Aligned Doubleword" column (applicable to operations less then or equal to 8 bytes) means that the EA bits(59:60) of all bytes of the operation are equal; that is, the operation does not cross a doubleword boundary. Whereas, the column "Aligned to Doubleword Boundary" (applicable to 16-byte operations) means that the computed operation EA(61:63) = 0; that is, the operation is doubleword aligned.

All operations of less then or equal to 8 bytes that do not cross a doubleword boundary (which includes all naturally aligned operation sizes) and all 16-byte operations, which are either doubleword or quadword aligned, consume the minimum number of LS slices for execution: one slice for 8 bytes or less and two slices for 16 bytes.

For example:

A 4-byte load with EA(59:60) = 1 and EA(61:63) = {0, 1, 2, 3} goes to LS slice 1.
A 4-byte load with EA(59:60) = 1 and EA(61:63) = {5, 6, 7, 8} goes to LS slices 1 and 2.

A 16-byte load with EA(59:60) = 3 and EA(61:63) ≠ 0 starts at LS slice 3 and also goes to LS slices 0 and 1.

25.1.7.1 Tracking Load and Store Ordering

Load iops are tracked by a load reorder queue (LRQ) and store iops are tracked by a store reorder queue (SRQ) to maintain correct architectural storage ordering and cache coherency. The LRQ and SRQ are shown in *Figure 25-1*.

There is one LRQ per pair of LS slices, one for LS slices 0/1, and one for LS slices 2/3. The LRQ tracks each load iop which is executed in one of its slices and tracks up a 16-byte load in each entry.

A 16-byte load, which is naturally (quadword) aligned, is therefore tracked in a single LRQ entry. Whereas, 16-byte loads that span a quadword boundary are tracked in two LRQ entries; one in each LS slice pair.

The LRQs are two level: the first level (pre-finish) LRQ tracks loads that have not yet executed successfully, while the second level finish-LRQ tracks loads after they have finished and until they are completed. Each of the two LS slice pair LRQs has 10 pre-finish and 28 post-finish entries, for a total of 76 LRQ entries per core.

There is one SRQ for each of the four LS slices. The SRQ tracks each store iop which is executed in the slice. Therefore the number of SRQ entries per iop is given by the number of slices indicated in *Table 25-5* on page 355. Each slice SRQ holds 16 entries for a total of 64 SRQ entries per core.

25.1.7.2 LS Slice Execution

Load/store iops are received by the LS slices and are bypassed to each LS-slice pipeline for execution if the pipeline is not already processing a previously received load or store operation. When the load/store iop is not able to execute immediately, it is queued in either the Data Address Recirculation Queue (DARQ) or the Load Store Address Queue (LSAQ) per LS-slice pair for issue into the pipeline in a future cycle.

Queuing in the LSAQ allows iops to be freed from the main slice issue queues after their dependencies are met. Dependent operations for load/store iops are issued only after the load/store iop is finally issued into the pipeline.

Iops that have been executed but encounter an execution hazard, such as an L1 D-cache miss or effective-to-real address translation for data (D-ERAT) miss are re-issued from their LRQ or SRQ entry back into the slice pipeline after their hazard condition is resolved. See *Section 25.1.7.10 Load/Store Pipeline Hazards* for a complete list of LS-slice execution hazards that are tracked in the LRQ and SRQ entries.

Iop execution takes place in each LS slice independently from each other LS slice, such that iops that are routed to more then one slice are not completely finished with execution until they are finished in each slice on which they are executing.

25.1.7.3 L1 D-Cache

The L1 D-cache is the first level of cache available to load and store operations. It is 32 KB and organized as 8-way set-associative with 128-byte cache lines. Half cache lines, with 64 bytes of data each, are supported.

The L1 D-cache is sliced by a doubleword providing four independent ports (one per LS slice) each returning up to 8 bytes, for a total of 32 bytes of cache access per cycle. The L1 D-cache is reloaded by the L2 cache at a rate of up to 64 bytes per cycle. The L2 reload bus is dynamically shared with the other core of a core-pair (when active).

The L1 D-cache has eight banks. Cache writes and reads to different banks occur simultaneously from each LS slice.

When load and store operations execute, they use a set prediction directory (SETP) to reduce cache access latency. The SETP provides a cache-hit and set selection indication. In parallel to accessing the SETP, the L1 D-cache directory and ERAT are also accessed and are used to confirm the SETP cache-hit and set-selection.

When a load encounters an SETP hit and ERAT miss, a translation is performed to confirm the SETP prediction. If the translation results in a TLB miss or an SLB miss for a hashed-page table (HPT) translation mode, the load is flushed from the pipeline and is re-fetched while the translation process proceeds.

When a load encounters an L1 D-cache miss, the following occurs:

- A request is made to the L2 to retrieve the cache line.
- A load miss queue (LMQ) entry is allocated for the cache line or the request is merged into an existing LMQ entry for a matching cache line.
- An LRQ entry associated with the load waits for the cache line to return from the cache hierarchy before waking the load back up for re-issue into the execution pipeline.

The LMQ holds 12 cache-line miss requests per core, eight for general loads and one per thread to handle load-and-reserve operations, which are discussed in *Section 25.1.8.1* ***larx/stcx Instruction*** on page 365.

If there is already an LMQ entry active for the same cache line, the load becomes dependent on the same LMQ entry for re-execution, with no limit on the number of loads that are dependent on a particular LMQ entry. Otherwise, if there are no remaining LMQ entries, the load re-arbitrates for an LMQ entry once an entry becomes available.

The L1 D-cache is store-through. Stores that miss the cache write into the L2 cache after they are complete, but do not allocate an entry in the L1 D-cache. If a store hits the SETP/L1 D-cache, it writes the cache once it is complete as it is being sent to the L2 cache.

A write into the L1 D-cache by a store makes the line private to the thread of the store performing the write. If any other thread requests to access the line that is marked as private, the line is evicted from the L1 D-cache. If the request from the other thread was a load, the line is brought back into the cache in a non-private state available to all threads.

25.1.7.4 D-ERAT

Each pair of LS slices has two D-ERAT structures for performing address translation. The two pairs of D-ERAT are kept synchronized. Each D-ERAT is implemented as a fully-associative 64-entry array, with a binary LRU replacement algorithm. D-ERAT entries are created for 4 KB, 64 KB, 2 MB, and 16 MB pages only. 1 GB and 16 GB pages are broken into 16 MB entries in the D-ERAT, where the installed page contains the referenced address. Each D-ERAT can support one lookup per cycle and also supports hit-under-miss.

D-ERAT misses are tracked in a 4-entry ERAT miss queue (EMQ). One EMQ entry is allocated per cycle per core. EMQ entries arbitrate with I-ERAT misses to perform translation. The EMQ holds the ERAT miss while translation is being performed. After translation has been performed, all D-ERATs are loaded with the new effective-to-real address translation. Both load and store translations are requested and performed speculatively. That is, as iops miss the ERAT during execution, translations are performed and the D-ERATs are loaded based on the speculative translations.

D-ERAT entries are invalidated based on any **tlbie** for which a match is architecturally required. For a 256 MB, 1 GB, or 16 GB tlbie invalidate, all D-ERAT entries for the matching LPID/PID are invalidated.

25.1.7.5 Translation Look-Aside Buffer

The TLB has 1024 entries and is four-way associative. It supports translation for both D-ERAT and I-ERAT misses. The TLB uses a true LRU replacement algorithm.

When any thread is initialized on the processor that supports Radix page table (RPT) mode, the TLB is also used as a page-walk cache (PWC), which holds individual page table references cached from previous translations. In RPT mode, the TLB has 512 entries and the PWC has 128 entries for the L1, 128 entries for the L2, and 128 entries for L3 references. The PWC is implemented as direct mapped.

After a thread is activated with RPT mode active, the processor core TLB remains in the TLB/PWC split mode.

The TLBs are indexed with a hashed address calculated from portions of the virtual address and the page size. Each entry in the TLB represents a particular page size: 4 KB, 64 KB, 2MB, 1 GB, and 16 GB, that is, all page sizes are natively supported in the TLB and consume only one entry.

If a translation request does not hit in the TLB, a tablewalk is initiated that loads the TLB and either the D-ERAT or I-ERAT depending on if it was a load/store data access or instruction access respectively. Up to two outstanding tablewalks can be active at one time. The implementation allows tablewalks for speculative instructions but does not update the Ts or change (C) bit in a PTE entry unless the instruction is NTC when the PTE entry is found. The TLB is reloaded with the corresponding PTE entry even if the instruction that requested the translation is speculative.

If a store misses the TLB after missing translation, and the C bit is not on, then a second tablewalk is done after the iop is NTC, so that the C bit can be updated.

There are four 32-entry SLBs, one per thread, that are accessed before accessing the TLB during HPT mode. If there is an SLB miss, a segment tablewalk is performed. Four entries of the SLB are architected and can be loaded by the software (one per thread). The SLB supports FIFO replacement for the other 28 entries.

25.1.7.6 Store Forwarding

As loads are issued into each LS-slice pipeline, they check each SRQ entry for older stores on which they are dependent (that is, for stores with overlapping address ranges). A load-hit-store (LHS) is when an overlapping store is found. When a store with an overlapping EA(44:63) is found, the load is identified as a candidate for store-forwarding and its execution is delayed by two cycles, allowing store forwarding to take place. Any iop selected for issue within the same LS slice that collides with the execution of the two-cycle delayed iop, is also delayed by two cycles, instead of being rejected or recycled for future execution. This allows for a fully pipelined execution of iops that are able to perform store forwarding.

In addition to matching EA(44:63) bits between a load iop and a STQ entry, the rest of the EA, as well as the RA, is also compared to confirm the store forwarding condition. When the RA compare matches between a load and the store but the EA compare does not match, or vice-versa, and store forwarding was not restricted due to another reason, then the load iop is flushed from the pipeline and re-fetched for execution.

Store forwarding is supported within each LS slice by taking overlapping bytes from a single store and non-overlapping bytes from an L1 D-cache hit. However, in the following cases, store forwarding cannot take place:

- Forwarding is required from more than one store per LS slice; for example, different byte ranges within a single LS slice are being stored to by more than one store iop. The load iop must wait for an overlapping store to drain from the STQ after completion and be written to and read from the cache hierarchy before it is able to successfully execute.
- Forwarding is required from both a single store in the STQ-slice, and from non-overlapping bytes of the same cache line but the cache line data is not in the L1-D-cache; for example, an L1-D-cache miss. The load iop must wait for the store to drain and for a miss condition to bring the updated line back into the L1 D-cache.
- If the load iop or store iop are on pages that are caching inhibited or guarded.

To increase store forwarding opportunities, stores are kept in the STQ after they are drained when they were an L1 D-cache miss to allow for loads to perform store forwarding. These stores are evicted from the STQ if their STQ entry is required by younger executing stores.

25.1.7.7 Out-of-Order Load/Store Execution

As stores execute they check each LRQ entry for the corresponding LS slice for any younger loads accessing overlapping bytes of storage that have already accessed the cache or generated a cache miss. Any matching loads are flushed from the pipeline and re-executed. Such a condition is called store-hit-load (SHL) flush.

To avoid the SHL flush condition, the POWER9 core has mechanisms to restrict load execution ahead of older stores, as in the following cases:

- When a load detects an older store in the pipeline with a matching base register and displacement and/or base and index register, an execution ordering dependency is created between the load and the older store. This mechanism is subject to the distance of the load from the store.
- When a load at the same instruction EA has previously encountered an SHL flush. Upon re-execution, the load is made dependent on an older store. This mechanism is not employed for load-string and load-multiple. This mechanism is subject to the distance of the load from the store.

Because of the risk for an SHL flush, compilers should generally avoid producing loads dependent on older stores in near proximity, especially those where the base and displacement, and/or base/indexed registers do not match.

25.1.7.8 Load-to-Use Latency

Iops dependent on register results from older loads are able to issue with a nominal issue-to-issue latency of four core cycles. *Table 25-6* lists the full set of issue-to-issue latencies for various iop types and alignments when data is resident in the L1 D-cache or is available to be forwarded from an older store. Each of these latencies is further dependent on the actual LS-slice execution time per the description in *Section 25.1.7.2 LS Slice Execution* on page 356.

Table 25-6. Load Issue to Dependent Iop Issue Latencies for L1 Hit or Store Forwarding

Load Iop	Dependent Iop	Nominal Latency (core cycles)	Additional Latency (core cycles)	Notes
Base	Nonsynchronous Issue	4	-	
Base	Synchronous Issue	5	-	128-bit Operand Dependent Iops
≤8 bytes and Does Not 'Fit in Aligned DW'	-	6	-	
16 byte and Not 'DW Aligned'	-	6	-	
Crosses Aligned 32-byte Granule	-	-	+3	
Store Forwarding	-	-	+2	

Dependent iops that have a synchronous issue, including iops marked as having vector-dispatch and/or paired-issue in *Table A-1. Instruction Properties* on page 375, have a 5-cycle nominal issue latency after the load issues. Nonsynchronous iops that are concurrent in the execution pipeline together with synchronous iops can be issued with synchronous issue latency, nominal five cycles, to improve pipeline usage.

Load iops that take more then the minimum number of LS slices due to alignment, see *Table 25-5* on page 355, have a 6-cycle nominal issue latency to a dependent iop issue of any type.

Load iops that cross a 32-byte aligned granule take an additional three cycles to execute.

Note: The effective issue latency for single-precision floating-point load iops and load algebraic iops feeding dependent instructions is increased because they are cracked instructions. Each instruction is cracked into a load iop followed by a conversion iop that takes an additional two cycles to execute before a dependent instruction can issue.

25.1.7.9 Load/Store Throughput

Each of the two LS-slice pairs is capable of executing the following iops per cycle:

- An 8-byte load/store per slice; for example, either two doubleword contained iops or one iop spanning both slices
- A combination of any one load iop and any one store iop, each spanning one or both slices
  - For example, includes one 16-byte load and one 16-byte store per cycle per LS-slice pair if quadword aligned
  - For example, includes one unaligned 8-byte load and one unaligned 8-byte store per cycle per LS-slice pair

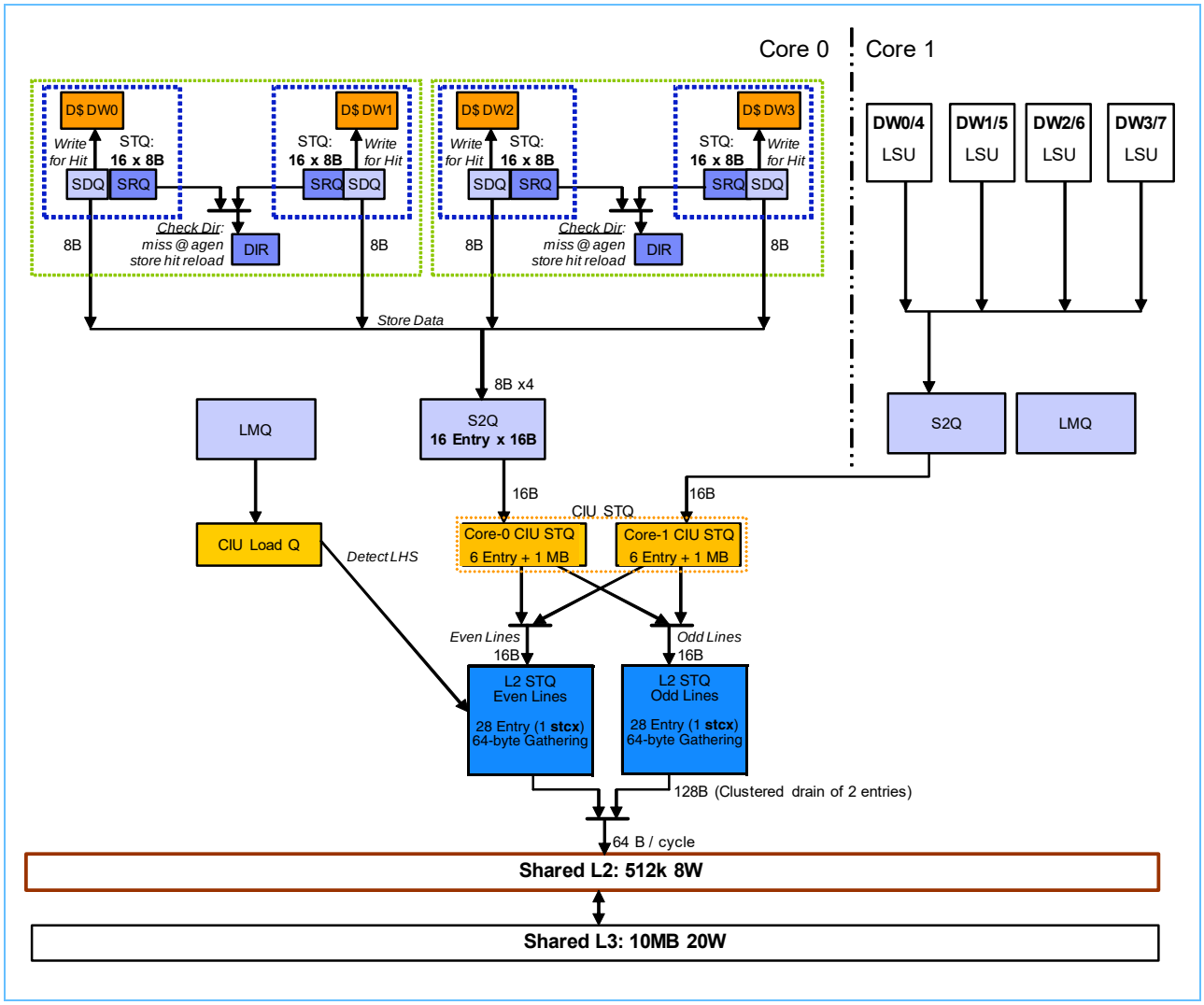
After they complete, stores are drained from each LS-slice STQ at a rate one iop (8 bytes) per cycle per STQ into a 16-entry per core store drain queue (S2Q), with each entry holding up to 16 bytes. For example, store-vector and store-quad instructions can store up to 16 bytes per cycle. The stores are then pipelined through a core interface unit queue (CIU-STQ) to the L2 STQ at a rate of one instruction per cycle, up to 16 bytes, into two L2 STQ banks, even and odd lines.

In the L2 STQ, the stores are gathered before writeback of up to 128 bytes per 2:1 clock.

The store drain interface to the L2 CIU STQ is private to a given core, but shares L2 STQ entries with the other core of the core pair, allowing up to 16 bytes per core to drain to the L2 STQ when stores are going to different L2 STQ banks. The two CIU STQs maximize their unload rate by searching for even/odd bank entries that can be unloaded in the same cycle.

In the L2 STQ, the stores are gathered before writeback to the L2 cache of up to 128 bytes per 2:1 clock. This allows stores with line locality to sustain a drain rate of 16 bytes per core into the L2 cache.

Figure 25-6. Store Drain Path from Core-to-L2 Cache



Note: When a load is detected as overlapping a store in the L2 STQ, the STQ entry is marked for quick write-back to allow the load request to be processed.

25.1.7.10 Load/Store Pipeline Hazards

The following conditions detected in the load/store pipeline result in a pipeline flush at or after the problem instruction:

- SETP hit but tag miss, see *Section 25.1.7.3 L1 D-Cache* on page 356.
- TLB miss after an SETP hit, ERAT miss, see *Section 25.1.7.3* on page 356.
- TLB miss followed by SETP/L1 D-cache miss, ERAT miss; invalidate or eviction in translation window.
- Load-hit-store flush: RA does not match, but forwarding was selected, see *Section 25.1.7.6 Store Forwarding* on page 358.
- Store-hit-load flush: Older store executes after younger load to same address, see *Section 25.1.7.7 Out-of-Order Load/Store Execution* on page 359.
- Cache inhibited load (I = '1') detected in PTE entry (not explicit cache-inhibited instruction).
- Out-of-order **larx** detected for same thread
- Snoop or store from the other thread, invalidates the younger load data while the older load in the pipeline is flushed.
- SAO mode: Snoop or store from the other thread invalidates the younger load data while the older load to same address in the pipeline is flushed.
- Snoop or store from other thread invalidates part of load-quad
- Snoop or store from other thread invalidates load while **sync** is pending
- Snoop or store from other thread invalidates load while **tend** is pending
- TLBIE snoop response expedited and flush impacted stores and load misses
- ECC: UE on data from memory

25.1.7.11 64-Byte Cache-Line Data

The POWER9 core implements a memory controller with adaptive behaviors to manage high-memory bandwidth utilization effectively. One of these features is the capability to fetch only 64 bytes of data (half cache lines), instead of the normal full cache-line size of 128 bytes of data from the memory when memory bandwidth utilization is very high.

The processor core is responsible for indicating on all cache-hierarchy requests if the request is allowed to return as 64 bytes; for example, 64B_ok. When this 64B_ok indication is not set, the memory controller must return the full 128-byte cache line. Additionally, the processor dynamically forces the 64B_ok indication to '0' to optimize performance. The 64B_ok indication is set to '0' in the following cases; otherwise, it is set to '1':

- Instruction fetch
- Data prefetch
- Translation tablewalks
- IOP crosses 64-byte granule
- **Larx**
- Partial hit in the L1 cache
- Control register specifies *require-128B*

25.1.7.12 Data Prefetch

The data prefetch engine can recognize sequentially increasing or decreasing accesses to adjacent cache lines and then request anticipated lines from more distant levels of the cache/memory hierarchy. The usefulness of these prefetches is reinforced, as repeated demand references are made along such a path or stream. The depth of prefetch is then increased until enough lines are being brought into L1, L2, and L3 that much or all of the load latency can be hidden. The most urgently needed lines are prefetched into the nearest cache levels.

Data Prefetch Features

- Eight active streams tracked
- EA based
- Software eDCBT/ST control support
- L1 and L3 prefetching
- Stride-N detection
- Duplicate stream removal
- Finite stream length support
- Full LRU
- Full set of user controls (HID, LPCR, DSCR, SCAN)
- Fully pipelined, 4 prefetches in-flight in core
- 32 prefetch engines in L3 shared by up to two paired cores
- Bandwidth sensitivity controls: adaptive prefetch

Track all Load and Store Addresses to Identify Stream Patterns

The prefetch subunit tracks load and store addresses using a register file referred to as the prefetch queue (PRQ). Loads that miss the L1 data cache, and which do not appear to be part of an existing stream that the PRQ is already tracking, are eligible for consideration to be added to PRQ as a new potential stream to track. This is referred to as allocating a new stream. When streams are allocated, the address of the next predicted cache line in the stream is written into the PRQ.

After a stream has been allocated, subsequent loads and stores that match the next predicted address in the stream are said to confirm the stream. This confirmation results in an update to the PRQ for that entry. The entry is updated to the address of the next predicted in the stream.[1]

During stream start up (allocate), a burst of lines can be requested from the memory subsystem, depending on the measured confidence. After a steady state is achieved, each stream confirm causes the engine to bring one additional line into the L1/L2 cache, and one additional line into the L3 cache.

1. The stream is always assumed to be going up in address space on an allocate (for example, N+ 1). However, if the next sequential load is going downward (for example, N-1 is seen), the stream direction is reversed. This reversal can happen again (for example, if N+1 is seen again), the stream direction is then reversed to an upward direction again.

PRQ tracks load and store addresses as they are accessed by the program. After a pattern has been identified and a stream is established, the prefetch subunit begins making L1/L3 prefetch requests by injecting into one of the LS-slice execution pipes for a single cycle and inserting the L1/L3 prefetch address into the dataflow. If the prefetch hits the L1 data cache, it is discarded.

L1 prefetches are then placed in the LMQ and the request is sent to the memory subsystem to bring the data back into the L1/L2 cache. L3 prefetches bypass the LMQ and are sent to the memory subsystem using the shared translate interface. Unlike L1 prefetch requests, the L3 prefetch requests only indicate to the memory subsystem that the data should be brought into the L3 cache.

Adaptive Prefetching

The POWER9 core implements a memory controller with adaptive behaviors to manage high-memory bandwidth use effectively. One of these features is the capability to drop prefetch requests based on the demands for memory bandwidth observed at the memory controller.

The data prefetcher is designed to provide guidance to the memory controller to control the priority per prefetch, by assigning a confidence level to each prefetch request. This allows the memory controller to drop less confident prefetches while holding on to more confident prefetches, depending on the extent of memory bandwidth contention. The data prefetcher predicts the confidence based on stream and program history.

The data prefetcher is able to identify phases of program execution where prefetching might be more effective. It uses this information, and also receives feedback from the memory controller to assist with making decisions about the ramp (how many prefetches to send relative to the number of confirms), and the depth (how far ahead to prefetch).

Adaptive prefetch assists programs in achieving optimal performance without detailed prefetch tuning. However, for data sensitive programs, it is recommended to perform prefetch tuning to achieve optimal performance. A DSCR URG setting of '1' forces the machine into the most conservative adaptive prefetch mode. Conversely, a DSCR URG setting of '7' precludes the adaptive mechanism from entering a conservative mode of operation.

As streams are detected, initial prefetches are sent out to look ahead; initially between 0 - 4 lines depending on urgency and the adaptive prefetch state. As load and store addresses for subsequent operations are detected to align with the ongoing stream, the stream is advanced. The stream can move ahead of the current load/store address position to a maximum depth, which is configurable. The default is between 4 - 24 lines ahead depending on the DSCR DPFD setting and SMT mode (other depths are available based on the configuration).

Stride-N prefetching on the POWER9 processor operates in all SMT modes. The hypervisor and operating system should initialize the DSCR with stride-N active.

25.1.7.13 Software-Initiated Data Prefetch

The **dcbt** TH = '0' and **dcbtst** instructions cause a single-line prefetch (**dcbt** into the L1 cache and **dcbtst** into the L3 cache).

The **dcbt**/**dcbtst** TH = x'8', x'B', x'A' values identify start and stop streams that bring data into the L1 cache:

- A stream takes a PRQ entry
- Streams can specify a finite length (UNITCNT) or unlimited. If the UNITCNT is never reached, the stream stays active occupying a PRQ entry until it is converted to a software stream on a context switch.
- At creation, software streams initiate prefetches until either the specified depth (as indicated in DSCR or LPCR DFPD) is reached or the UNITCNT is reached, which ever comes first.
- Software streams are converted to hardware streams as follows:
  - On a context switch
  - When prefetch stop is indicated
  - When the UNITCNT is reached
- Transient indication impacts L3 cache replacement, biasing toward being more easily replaced when set.

In practice, scheduling the prefetch (**dcbt**/**dcbtst**) far enough ahead for misses is challenging due to aggressive pipelining. To hide the latency of a memory miss, a long stream or a long lead time is required to start prefetching. However, when used effectively, software prefetch streams are a very powerful tool for boosting performance.

Software streams can also be used to guarantee that a particular stream of importance is allowed to gain maturity and remain active and not be evicted if it is deemed critical. This is because the software streams are able to stay resident and are eligible for eviction from the PRQ until they are either stopped or encounter a context switch.

DCBZ

The **dcbz** instruction enables invalidation and zeroing of lines before storing to them, which reduces traffic to the memory controller and reduces time to ownership of a line. The **dcbz** instruction is a powerful tool when software knows that a line is fresh (writable) without consequence.

The best practices are to issue the **dcbz** ahead of where stores will occur to a region.

25.1.8 Special Instruction Sequences

25.1.8.1 larx/stcx Instruction

Load reserved and store reserved sequences give the programmer/compiler the ability to share storage in an effective manner by exchanging locks and updating atomic variables between program threads.

The POWER9 core has improved lock performance for both uncontested and contested locks. However, care must be taken to follow the coding guideline reservations.

EH Bits

The EH hint bits are honored by the POWER9 core and should be used diligently to distinguish between:

- Critical sections (EH = '1'), where the lock-line is held while work is performed and the lock will later be released with a store to the lock granule.
- Atomic updates (EH = '0'), where a passing **stcx** completes the use of the lock.

If the correct usage is unknown, the compiler should set the EH hint bits to favor the atomic update case.

Contested Locks

Contested locks should be kept on separate and unique lines relative to shared data. To achieve this separation might require padding by the programmer. This allows the locks to be contested by the caches without the side effect of the lock granule false sharing in the same cache line as the critical section data. Looping on a contested lock is a common strategy. While looping, it is preferred to poll the lock with a load and then attempt a **larx** instruction after the lock is free. It is also common to go into a lower-thread priority state while polling. However, priority NOPs should be avoided within the inner polling loop, and instead the NOPs should be inserted before starting and after exiting the polling loop.

25.1.8.2 icbi Instruction

The **icbi** is treated as a NOP on the POWER9 core, except that it provides **isync** with the required synchronization around storing into the instruction stream.

25.1.8.3 isync Instruction

An **isync** pauses briefly at NTC to check if any special conditions have occurred. If no special conditions have occurred, it completes with no effect. However, if a special condition has occurred since the last flush of the pipeline, all subsequent instructions are flushed from the pipeline and re-fetched.

Some **isync** special conditions are as follows:

- Store invalidate to I-cache
- CSI required by move-to SPR scoreboard operations (see *Power ISA Operating Environment Architecture - Book III (version 3.0B)* for CSI registers)
- Set by the following operations when they are NTC: **icbi**, **ptesync**, **tlbie**, **tlbiel**, **slbie**, **slbia**, **mtsr**, **mtsrin**, **slbmte**

25.1.8.4 ptesync Instruction

A **ptesync** is held at dispatch until all loads have been executed; for example, including no outstanding load misses. It synchronizes at NTC with the L2 cache to ensure that no snoop or store from another thread matches any of the loads in the pipeline. If any does match, it causes a flush of the pipeline and a refetch of subsequent instructions.

25.1.8.5 sync Instruction

The **sync** (**hwsync**) instruction synchronizes at NTC with the L2 cache to ensure no snoop or store from another thread matches any loads in the pipeline. If a match is detected, a pipeline flush occurs along with a refetch of the subsequent instructions.

25.1.8.6 eieio Instruction

An **eieio** instruction is held at dispatch until all loads have been executed; for example, including no outstanding load misses. Cache-inhibited loads are rejected when an older **eieio** is active in the pipeline.

25.2 Cache and Memory Hierarchy

Each L2 and L3 cache are connected to a pair of processors and also to other caches and services on the chip through a fabric bus. Both caches support 128-byte cache lines and also support 64-byte cache line data valids allowing for half cache-line memory reads. Cache coherency is maintained on a 128-byte line size.

When the processor core requests the invalid portion of a 64-byte valid cache line, the L2 or L3 cache performs a read of only the other half of the cache line from memory and returns the entire 128-byte cache line to the L1 cache.

25.2.1 L2 Cache

Each L2 cache is 512 KB and 8-way set associative with fast access to its own private 10 MB L3 cache region through a private low-latency bus. The L2 cache maintains full hardware coherence within the system and can supply intervention data to the other cores on this POWER9 chip or to other cores on other POWER9 chips. Logically, the L2 cache is an in-line cache. Unlike the L1 caches, which are store-through, it is a store-in cache. The L2 cache is fully inclusive of the L1 D-caches and the L1 I-caches.

The L2 replacement policy uses an LRU with a vector-tracking tree that includes cache-invalidate state biasing and takes L1 access updates from each core.

The L2 cache is dual-banked, even versus odd lines, and can support a read to one bank while performing a write on a different bank.

On an L2 hit, the L2 cache returns data to the core at a rate of 64 bytes per core cycle, installing a full cache line over two back-to-back core cycles. On an L2 miss, the L2 cache returns 32 bytes per core cycle.

25.2.2 L3 Cache

Each L3 cache region on the POWER9 chip is a unified victim cache for its respective core/L2 cache, as well as for other L3 caches on chip. The resident cache lines installed from the attached L2 cache are referred to as L3.0 lines, and the resident cache lines installed from other on-chip L3 caches are referred to as L3.1 lines. When castout from an L3 cache are victimized, L3.1 lines go to memory and L3.0 lines have the option of being castout to other L3 caches on chip. The L3 cache is a 20-way associative 10 MB cache. The L3 cache maintains full hardware coherence within the system and can supply intervention data to the other cores on this POWER9 chip or to other cores on other POWER9 chips. Logically, the L3 cache is an in-line cache. The L3 cache is a victim cache[1] of the L2 cache. The L3 cache is not inclusive of the L2 cache.

1. All valid lines that are victimized in the L2 cache are castout to the L3 cache.

The L3 replacement policy uses a per entry state that is based on historical access rates, data sources, and requesting transfer types. The state is kept as multiple bits per entry over two sets of 10 entries. The decay rate for entries controls the stickiness of previously referenced cache lines and can be adjusted via SCOM. The install stickiness can be tuned and adjusted at IPL time for instruction-cache and translation operations.

The L3 cache returns data to the core at a rate of 32 bytes per core cycle, with a full cache line delivered over four core cycles.

The L3 cache supports full and partial cache line injection that allows for updates from the PCIe and other sources.

25.2.3 Cache Latencies and Bandwidth

Table 25-7 lists several key bandwidth and latency values for the chip. These represent best case values under ideal conditions. Actual values can vary due to resource limitations or queueing effects. The pclk in *Table 25-7* is the clock rate of the processor core and the latency is relative to the four core cycle nominal latency for load to a dependent issue.

Table 25-7. Cache and Memory Hierarchy Load to Issue Latencies and Bandwidth

Description	Latency	Bandwidth
L2 D-cache load hit (bypass)	15.5 pclks	64 bytes/pclk
L2 I-cache load hit (bypass)	16.5 pclks	64 bytes/pclk
L3 load hit	35.5 pclks	32 bytes/pclk
L2.1 load hit	variable	16 bytes/pclk
L3.1 load hit	variable	16 bytes/pclk
Memory load (local)[2]	68 ns	(see *Section 25.4* on page 369)

1. Pclks represent one processor core clock.
2. DDR4-2400, CAS latency = 17 t_{CK}, 2 GHz nest, 4 GHz core, nominal wire length, no refresh. Note that single and double refresh, along with memory speed, rank, and core location relative to the memory controller impacts memory latency.

25.3 NX Accelerators

The NX accelerators support the throughput shown in *Table 25-8*.

Table 25-8. NX Accelerator Throughput

Operation	Engine	Single-Engine Throughput at 2 GHz[1]
AES CBC encrypt 128-bit key 4 KB block	AES	8 Gbps
AES CBC encrypt 256-bit key 4 KB block	AES	6.4 Gbps
SHA 256 256-byte block	SHA	3.7 Gbps
SHA 512 256-byte block	SHA	5.8 Gbps
Compression	842	16 GBps peak into compressor[2]
Decompression	842	16 GBps peak out of decompressor[2]
Compression	Gzip	16 GBps peak into compressor[2, 3]
Decompression	Gzip	16 GBps peak out of decompressor[2, 3]
Random number output stream	RNG	80 Mbps[4]

Note:

1. Aggregate throughput can be less than the sum of individual engine peak throughputs. Processor bus ramp is limited to 32 GBps per direction. Use of indirect DDEs reduces throughput. ERAT hit assumed on all memory accesses. Not reduced for any coprocessor invocation overhead. Memory is assumed to be nodal and engine throughput is not limited by memory bandwidth.
2. Throughput in the reverse direction depends on compression ratio.
3. Dependent on compression ratio, fixed or dynamic Huffman coding, and data size.
4. Subject to lab validation. The stated bit rate assumes the RNG can deliver a high-quality random number stream with the RNG Pacing Control Register set to deliver this rate.

25.4 Direct Attach Memory

There are eight memory controllers on each POWER9 chip that can be configured into address interleave groupings. The maximum memory that a grouping can address is 4 TB and the maximum that can be addressed from a single chip is 4 TB. The maximum memory speed supported is 2667.

Table 25-9 shows the sustained memory bandwidth for a selection of workloads with single-refresh on, two ranks and one DIMM for each of the eight channels.

Table 25-9. POWER9 Memory Bandwidth for Eight Channels Active

Workload	Bin Speed	POWER9 Bandwidth (GBps)
Read-only (ddot)	2400	141
2:1 stream (daxpy)	2400	132
1:1 stream (copy)	2400	127

25.5 PCI Express

PCIe supports PCIe Gen4 with payload sizes of 512-byte write and 256-byte read, delivering an effective bandwidth of 28 GBps read and write.

The PCIe can process writes into the on-chip cache using DMA inject, which can write in increments as small as one byte.

25.6 CAPI

The POWER9 processor supports both the CAPI 2.0 interface that operates over the PCIe and the Open-CAPI 3.0 interface operating on up to 32 lanes of 25G link.

25.7 Interrupt Controller

The interrupt controller supports up to 40 million interrupts per second, requiring about 18 GBps read and 18 GBps write bandwidth onto the fabric.

25.8 Nest MMU

The Nest MMU (NMMU) processes address translation requests for the NX accelerators, NPU, and CAPI interfaces.

Table 25-10 shows the range of expected peak latencies and throughput (translations per second) that can be performed by the 12 tablewalk state machines.

Table 25-10. NMMU Translation Latency and Throughput

	Streaming	Random
Average Latency (ns)	46 - 52.5	76 - 85
Translation Rate (translations per second)	221 - 261 million	141 - 158 million

25.9 NVLink

The NPU provides a cache coherent interconnect between POWER9 and GPU chips over the NVLink interface. The NPU maintains cache coherency on 128-byte cache lines.

The NPU can process writes into the on-chip cache using DMA inject, which can write in increments as small as 1 byte.

The POWER9 processor supports up to six NVLink bricks. Each NVLink brick supports the following peak bandwidth shown in *Table 25-11* on page 371.

Table 25-11. NVLink Peak Bandwidths Per Brick

Workload	NVLink Bandwidth (GBps)	P9 SMP Bandwidth (GBps)	Effective Bandwidth (GBps) with Command Overhead	Chip Total Effective Bandwidth
Read	25	32	23.5	141
Write	25	32	21.1	127

Each pair of bricks is connected to the fabric bus with a 64 GBps link, providing an effective 32 GBps per brick to handle the NVLink data, as well as address translation.

25.10 WOF/Power Management

The POWER9 processor provides the capability to automatically boost the processor frequency for performance when there is available power headroom. Workload optimized frequency (WOF) enables various workloads to achieve optimal performance. To enable the full benefits of WOF, programs should make make use of the low-power mode capabiity per thread as described in *Section 25.1.3 Instruction Fetch* on page 336.

The POWER9 processor provides enhanced power management controls including a new STOP instruction. This instruction can be used to reach power-savings states, including sleeping (power-gate) the entire core with reduced overhead compared with prior designs, providing greater utility and opportunity to save power. Together with the WOF, appropriate use of the STOP instruction can provide increased total system performance.

25.11 Instruction Properties

Characteristics for instructions are listed in *Table A-1. Instruction Properties* on page 375 including various latency, throughput, and interlock specifications:

- **Instruction Mnemonic and Name**: For cracked and expanded operations, this field is only valid for the first iop and subsequent rows indicate the behaviors for additional iops. Includes the architectural name for this instruction. A small number of instructions differ in their behavior between little-endian and big-endian modes. These instructions are listed with "_le" and "_be" suffix to distinguish behavior. When applicable, the mask or sub fields are shown as a hexadecimal suffix appended to the mnemonic.
- **Cracked/Expanded:**
  - C2 - cracked into 2 iops
  - C3 - cracked into 3 iops
  - X - expanded
- **Operation Number**: the number of the iop for the instruction
  - "-" - single iop instruction
  - 1-N - iop number
  - Nu - Nth iop in the sequence is repeated depending on length field
- **Pipe Class:** designates the pipeline
  - ALU
  - ALU2
  - BR
  - CY
  - DIV
  - DFU
  - DP
  - DP-XC
  - DP-MUL
  - DX
  - PM
  - LD
  - LD2 - LD3
  - NOP
  - ST
  - ST2
- **Main Dst**: Indicates the main register target type.
  - GPR, FPR, VR, or VSR
- **CR Dst**: Indicates a CR target, if any.
- **XER/FPSCR/VSCR Dst**: Indicates XER, FPSCR, or VSCR target field, if any.
  - XER field groupings: ca/oc, dcds, fxcc, ov, reserved, string, tgcc
  - FPSCR field groupings: ctlr, excp, fpcc, fric (**Note**: excp is a sticky field)
  - VSCR field groupings: nj, sat
- **Maximum Operations Per Cycle**: The maximum rate of executing this iop within the processing pipeline: for multicycle instructions, this is shown as a ratio less than one. This is not always an indication of the peak sustainable execution rate for the specific instruction as various throughput limitations might apply.

- **Latency (Minimum)**: The minimum latency for executing a dependent iop relative to execution of this iop for the main register destination, if any. Additional latency for the non-main register destinations is added as specified in the consuming iop's field "Additional Latency for CR/XER//FPSR/VSCR Source". Note that issue-to-issue latencies should be consulted in *Section 25.1.5.4 Execution Pipeline Issue to Issue Latencies* on page 353 to identify additional latency components incurred when exchanging data between pipelines and between various data types. For load instructions, the latency reflects optimal address alignment. Additional latency can also be incurred as shown in *Section 25.1.7.8 Load-to-Use Latency* on page 359.
- **Latency (Maximum)**: The maximum latency for executing a dependent iop relative to execution of this iop for the main register destination, if any. Additional latency for nonmain register destinations is added as specified in the consuming iop's field "Additional Latency for CR/XER//FPSR/VSCR Source". Note that issue-to-issue latencies should be consulted in *Section 25.1.5.4 Execution Pipeline Issue to Issue Latencies* on page 353 to identify additional latency components incurred when exchanging data between pipelines and between various data types. For load instructions, the latency reflects optimal address alignment. Additional latency can also be incurred as shown in *Section 25.1.7.8 Load-to-Use Latency* on page 359.
- **Pipe Busy Cycles (Minimum)**: The number of cycles for which the execution pipeline is blocked by executing this iop. Multicycle instructions have a cycle count greater than one. Note that pipe-busy cycles are shared across shared pipeline groups {DIV}, {CY, DFU}, {DP, DP-XC, DP-MUL}, precluding execution within the group, during the busy cycles. See *Section 25.1.5.4 Execution Pipeline Issue to Issue Latencies* on page 353 for a description of multicycle instruction interactions.
- **Dispatch Rule**: See section *Section 25.1.4.5 IOP Dispatch* on page 344 for additional details.
  - "E" - Must dispatch to even slice, also consumes odd dispatch slice slot of the same superslice at dispatch
  - "P" - Dispatches together with previous iop from the same cracked instruction to the same superslice
  - "R" - Dispatch of this iop to a superslice restricts dispatch of a tuple of iops (restricts a second iop from going to either of the two slices of a superslice that cycle).
  - "R-st" - For ST and SMT2 modes, the dispatch of this iop to a superslice restricts dispatch of a tuple of iops (restricts a second iop from going to either of the two slices of a superslice that cycle).
  - "V" - Dispatches as a vector iop, a single decode iop is routed to both the even and odd slices of a superslice at dispatch
- **Dispatch Interlock**: See section *Section 25.1.4.5 IOP Dispatch* on page 344 and *Section 25.1.8 Special Instruction Sequences* on page 365 for additional details.
  - "WB" - This iop is held at dispatch if outstanding entries remain in the EAT; that is, until all older branches are completed and deallocated from the EAT.
  - "WS" - This iop is held at dispatch if any older dispatch score-board setting iops (Dispatch Interlock = SS) remain in the ICT; that is, until all older SS iops are completed.
  - "SS" - This iop sets a dispatch score-board. The score-board remains set while this iop is in the ICT.
- **Additional Dispatch to Issue Latency**: The number of additional minimum cycles that an iop must wait for issue beyond the nominal pipeline delay after first dispatched. This delay can overlap with source dependency delays.
- **Additional Latency for CR/XER/FPSR/VSCR Source**: The number of additional cycles that an iop must wait to issue relative to the producing iop if the iop is dependent on a register target other than the main register destination (CR/XER/FPSCR/VSCR). For dependency on a sticky field (such as FPSCR excp),

an "F" is shown indicating that the iop will issue depending on the current state of the FPSCR speculation mode, as described in *Move-To and Move-From FPSCR* on page 354.

- **Issue Synchronized**: "S" indicates that the iop issues concurrently across the even/odd slices of a super-slice as a vector or with a paired iop from the same cracked instruction.
- **Issue Depend on Previous Iop**: For a cracked instruction, this iop is dependent on a register destination of a prior iop from the same instruction:
  - "D1P" - this iop is dependent on the previous iop (OpNum - 1).
  - "D2P" - this iop is dependent on the second previous iop (OpNum - 2).
- **Issue Next-to-Complete**: This iop is held from issue until all older instructions in program order are complete. If this iop is a cracked iop, it is also held from issue until all previous iops from the same instruction (lower OpNum) are finished. For dependency on a sticky field (such as FPSCR excp), an "F" is shown indicating that the iop will issue depending on the current state of the FPSCR speculation mode as described in *Move-To and Move-From FPSCR* on page 354.

Appendix A. Instruction Properties

Table A-1. lists the POWER9 instruction characteristics including latency, throughput, and interlock specifications. See *Section 25.11 Instruction Properties* on page 372 for descriptions of the table headings and values.

Table A-1. Instruction Properties (Sheet 1 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
add	Add	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
add.	Add and Record	-	-	ALU	GPR	CR	fxcc	4	2	2	1	-	-			-	-	-
addc	Add Carrying	-	-	ALU	GPR		caoc	4	2	2	1	-	-			-	-	-
addc.	Add Carrying and Record	-	-	ALU	GPR	CR	caoc,fxcc	4	2	2	1	-	-			-	-	-
addco	Add Carrying and Record OV	-	-	ALU	GPR		caoc,ov	4	2	2	1	-	-			-	-	-
addco.	Add Carrying and Record OV and Record	-	-	ALU	GPR	CR	caoc,fxcc,ov	4	2	2	1	-	-			-	-	-
adde	Add Extended	-	-	ALU	GPR		caoc	4	2	2	1	-	-		3	-	-	-
adde.	Add Extended and Record	-	-	ALU	GPR	CR	caoc,fxcc	4	2	2	1	-	-		3	-	-	-
addeo	Add Extended and Record OV	-	-	ALU	GPR		caoc,ov	4	2	2	1	-	-		3	-	-	-
addeo.	Add Extended and Record OV and Record	-	-	ALU	GPR	CR	caoc,fxcc,ov	4	2	2	1	-	-		3	-	-	-
addex	Add Extended Using Alternate Carry Bit	-	-	ALU	GPR		ov	4	2	2	1	-	-		3	-	-	-
addg6s	Add and Generate Sixes	C2	1	ALU			dcds	4	2	2	1	-	-			-	-	-
			2	ALU2	GPR			4	3	3	1	-	-		3	-	D1P	-
addi	Add Immediate	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
addic	Add Immediate Carrying	-	-	ALU	GPR		caoc	4	2	2	1	-	-			-	-	-
addic.	Add Immediate Carrying	-	-	ALU	GPR	CR	caoc,fxcc	4	2	2	1	-	-			-	-	-
addis	Add Immediate Shifted	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-

Table A-1. Instruction Properties (Sheet 2 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
addme	Add to Minus One Extended	-	-	ALU	GPR		caoc	4	2	2	1	-	-		3	-	-	-
addme.	Add to Minus One Extended and Record	-	-	ALU	GPR	CR	caoc,fxcc	4	2	2	1	-	-		3	-	-	-
addmeo	Add to Minus One Extended and Record OV	-	-	ALU	GPR		caoc,ov	4	2	2	1	-	-		3	-	-	-
addmeo.	Add to Minus One Extended and Record OV and Record	-	-	ALU	GPR	CR	caoc,fxcc,ov	4	2	2	1	-	-		3	-	-	-
addo	Add and Record OV	-	-	ALU	GPR		ov	4	2	2	1	-	-			-	-	-
addo.	Add and Record OV and Record	-	-	ALU	GPR	CR	fxcc,ov	4	2	2	1	-	-			-	-	-
addpcis	Add PC Immediate Shifted	-	-	BR	GPR			1	5	5	1	-	-	2		-	-	-
addpcis	Add PC Immediate Shifted	C2	1	BR	GPR			1	5	5	1	-	-	2		-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	D1P	-
addze	Add to Zero Extended	-	-	ALU	GPR		caoc	4	2	2	1	-	-		3	-	-	-
addze.	Add to Zero Extended and Record	-	-	ALU	GPR	CR	caoc,fxcc	4	2	2	1	-	-		3	-	-	-
addzeo	Add to Zero Extended and Record OV	-	-	ALU	GPR		caoc,ov	4	2	2	1	-	-		3	-	-	-
addzeo.	Add to Zero Extended and Record OV and Record	-	-	ALU	GPR	CR	caoc,fxcc,ov	4	2	2	1	-	-		3	-	-	-
and	AND	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
and.	AND and Record	-	-	ALU	GPR	CR	fxcc	4	2	2	1	-	-			-	-	-
andc	AND with Complement	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
andc.	AND with Complement and Record	-	-	ALU	GPR	CR	fxcc	4	2	2	1	-	-			-	-	-
andi.	AND Immediate and Record	-	-	ALU	GPR	CR	fxcc	4	2	2	1	-	-			-	-	-
andis.	AND Immediate Shifted and Record	-	-	ALU	GPR	CR	fxcc	4	2	2	1	-	-			-	-	-

Table A-1. Instruction Properties (Sheet 3 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
b	Branch	-	-	BR				1	2	2	1	-	-			-	-	-
ba	Branch Absolute	-	-	BR				1	2	2	1	-	-			-	-	-
bc	Branch Conditional	-	-	BR				1	2	2	1	-	-			-	-	-
bca	Branch Conditional Absolute	-	-	BR				1	2	2	1	-	-			-	-	-
bcctr	Branch Conditional to CTR	-	-	BR				1	2	2	1	-	-			-	-	-
bcctrl	Branch Conditional to CTR and Link	-	-	BR				1	2	2	1	-	-			-	-	-
bcdadd.	Decimal Add Modulo and Record	-	-	DX	VR	CR		2	3	3	1	V	-	1		S	-	-
bcdcfn.	Decimal Convert From National and Record	-	-	DX	VR	CR		2	3	3	1	V	-	1		S	-	-
bcdcfsq.	Decimal Convert From Signed Qword and Record	-	-	DFU	VR	CR		1/26	37	37	25	V	-	1		S	-	-
bcdcfz.	Decimal Convert From Zoned and Record	-	-	DX	VR	CR		2	3	3	1	V	-	1		S	-	-
bcdcpsgn.	Decimal CopySign and Record	-	-	DX	VR	CR		2	3	3	1	V	-	1		S	-	-
bcdctn.	Decimal Convert to National and Record	-	-	DX	VR	CR		2	3	3	1	V	-	1		S	-	-
bcdctsq.	Decimal Convert to Signed Qword and Record	-	-	DFU	VR	CR		1/12	23	23	11	V	-	1		S	-	-
bcdctz.	Decimal Convert to Zoned and Record	-	-	DX	VR	CR		2	3	3	1	V	-	1		S	-	-
bcds.	Decimal Shift and Record	-	-	DX	VR	CR		2	3	3	1	V	-	1		S	-	-
bcdsetsgn.	Decimal Set Sign and Record	-	-	DX	VR	CR		2	3	3	1	V	-	1		S	-	-
bcdsr.	Decimal Shift and Round and Record	-	-	DFU	VR	CR		1	12	12	1	V	-	1		S	-	-
bcdsub.	Decimal Subtract Modulo and Record	-	-	DX	VR	CR		2	3	3	1	V	-	1		S	-	-

Table A-1. Instruction Properties (Sheet 4 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
bcdtrunc.	Decimal Truncate and Record	-	-	DX	VR	CR		2	3	3	1	V	-	1		S	-	-
bcdus.	Decimal Unsigned Shift and Record	-	-	DX	VR	CR		2	3	3	1	V	-	1		S	-	-
bcdutrunc.	Decimal Unsigned Truncate and Record	-	-	DX	VR	CR		2	3	3	1	V	-	1		S	-	-
bcl	Branch Conditional and Link	-	-	BR				1	2	2	1	-	-			-	-	-
bcla	Branch Conditional and Link Absolute	-	-	BR				1	2	2	1	-	-			-	-	-
bl	Branch and Link	-	-	BR				1	2	2	1	-	-			-	-	-
bla	Branch and Link Absolute	-	-	BR				1	2	2	1	-	-			-	-	-
bpermd	Bit Permute Dword	-	-	ALU2	GPR			4	3	3	1	-	-			-	-	-
cbcdtd	Convert Binary Coded Decimal to Declets	-	-	ALU2	GPR			4	3	3	1	-	-			-	-	-
cdtbcd	Convert Declets to Binary Coded Decimal	-	-	ALU2	GPR			4	3	3	1	-	-			-	-	-
cmp	Compare	-	-	ALU		CR	fxcc	4	2	2	1	-	-			-	-	-
cmpb	Compare Bytes	-	-	ALU2	GPR			4	3	3	1	-	-			-	-	-
cmpeqb	Compare Equal Byte	-	-	ALU2		CR	fxcc	4	3	3	1	R	-			-	-	-
cmpi	Compare Immediate	-	-	ALU		CR	fxcc	4	2	2	1	-	-			-	-	-
cmpl	Compare Logical	-	-	ALU		CR	fxcc	4	2	2	1	-	-			-	-	-
cmpli	Compare Logical Immediate	-	-	ALU		CR	fxcc	4	2	2	1	-	-			-	-	-
cmprb	Compare Ranged Byte	-	-	ALU2		CR	fxcc	4	3	3	1	R	-			-	-	-
cntlzd	Count Leading Zeros Dword	-	-	ALU2	GPR			4	3	3	1	-	-			-	-	-
cntlzd.	Count Leading Zeros Dword and Record	-	-	ALU2	GPR	CR	fxcc	4	3	3	1	-	-			-	-	-
cntlzw	Count Leading Zeros Word	-	-	ALU2	GPR			4	3	3	1	-	-			-	-	-

Table A-1. Instruction Properties (Sheet 5 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
cntlzw.	Count Leading Zeros Word and Record	-	-	ALU2	GPR	CR	fxcc	4	3	3	1	-	-			-	-	-
cnttzd	Count Trailing Zeros Dword	-	-	ALU2	GPR			4	3	3	1	-	-			-	-	-
cnttzd.	Count Trailing Zeros Dword and Record	-	-	ALU2	GPR	CR	fxcc	4	3	3	1	-	-			-	-	-
cnttzw	Count Trailing Zeros Word	-	-	ALU2	GPR			4	3	3	1	-	-			-	-	-
cnttzw.	Count Trailing Zeros Word and Record	-	-	ALU2	GPR	CR	fxcc	4	3	3	1	-	-			-	-	-
copy	Copy	-	-	LD				4	4	4	1	-	-			-	-	-
cp_abort	CP_Abort	-	-	LD				4	4	4	1	-	-			-	-	-
crand	CR AND	-	-	ALU		CR		4	2	2	1	R	-		3	-	-	-
crandc	CR AND with Complement	-	-	ALU		CR		4	2	2	1	R	-		3	-	-	-
creqv	CR Equivalent	-	-	ALU		CR		4	2	2	1	R	-		3	-	-	-
crnand	CR NAND	-	-	ALU		CR		4	2	2	1	R	-		3	-	-	-
crnor	CR NOR	-	-	ALU		CR		4	2	2	1	R	-		3	-	-	-
cror	CR OR	-	-	ALU		CR		4	2	2	1	R	-		3	-	-	-
crorc	CR OR with Complement	-	-	ALU		CR		4	2	2	1	R	-		3	-	-	-
crxor	CR XOR	-	-	ALU		CR		4	2	2	1	R	-		3	-	-	-
dadd	DFP Add	-	-	DFU	FPR		fpcc,fric,excp	1	12	12	1	E	-		3	-	-	-
dadd.	DFP Add and Record	C2	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
daddq	DFP Add Quad	C3	1	ALU	VSR			2	2	2	1	V	-	1		S	-	-
			2	DFU	FPR		fpcc,fric,excp	1	12	12	1	P	-		3	S	-	-
			3	DFU	FPR			1	12	12	1	P	-		3	S	-	-

Table A-1. Instruction Properties (Sheet 6 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
daddq.	DFP Add Quad and Record	X	1	ALU	VSR			2	2	2	1	V	-	1		S	-	-
			2	NOP				6	0	0	1	-	-			-	-	-
			3	NOP				6	0	0	1	-	-			-	-	-
			4	DFU	FPR		fpcc,fric,excp	1	12	12	1	P	-		3	S	-	-
			5	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			6	ALU2		CR		4	3	3	1	-	-		F	-	-	F
darn	Deliver A Random Number	-	-	LD	GPR			4	4	4	1	-	-			-	-	-
darn	Deliver A Random Number	-	-	NOP				6	0	0	1	-	-			-	-	-
dcbf	Data Cache Block Flush	-	-	LD				4	4	4	1	-	-			-	-	-
dcbst	Data Cache Block Store	-	-	LD				4	4	4	1	-	-			-	-	-
dcbt. TH00xxx	Data Cache Block Touch	-	-	LD				4	4	4	1	-	-			-	-	-
dcbt. TH01000	Data Cache Block Touch	-	-	LD				4	4	4	1	-	-			-	-	-
dcbt. TH01001	Data Cache Block Touch	-	-	LD				4	4	4	1	-	-			-	-	-
dcbt. TH01010	Data Cache Block Touch	-	-	LD				4	4	4	1	-	-			-	-	-
dcbt. TH01011	Data Cache Block Touch	-	-	LD				4	4	4	1	-	-			-	-	-
dcbt. TH011xx	Data Cache Block Touch	-	-	LD				4	4	4	1	-	-			-	-	-
dcbt. TH10000	Data Cache Block Touch	-	-	LD				4	4	4	1	-	-			-	-	-
dcbt. TH10001	Data Cache Block Touch	-	-	LD				4	4	4	1	-	-			-	-	-

Table A-1. Instruction Properties (Sheet 7 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
dcbt. TH1001x	Data Cache Block Touch	-	-	LD				4	4	4	1	-	-			-	-	-
dcbt. TH101xx	Data Cache Block Touch	-	-	LD				4	4	4	1	-	-			-	-	-
dcbt. TH11000	Data Cache Block Touch	-	-	LD				4	4	4	1	-	-			-	-	-
dcbt. TH11001	Data Cache Block Touch	-	-	LD				4	4	4	1	-	-			-	-	-
dcbt. TH1101x	Data Cache Block Touch	-	-	LD				4	4	4	1	-	-			-	-	-
dcbt. TH111xx	Data Cache Block Touch	-	-	LD				4	4	4	1	-	-			-	-	-
dcbtst. TH00xxx	Data Cache Block Touch for Store	-	-	LD				4	4	4	1	-	-			-	-	-
dcbtst. TH01000	Data Cache Block Touch for Store	-	-	LD				4	4	4	1	-	-			-	-	-
dcbtst. TH01001	Data Cache Block Touch for Store	-	-	LD				4	4	4	1	-	-			-	-	-
dcbtst. TH01010	Data Cache Block Touch for Store	-	-	LD				4	4	4	1	-	-			-	-	-
dcbtst. TH01011	Data Cache Block Touch for Store	-	-	LD				4	4	4	1	-	-			-	-	-
dcbtst. TH011xx	Data Cache Block Touch for Store	-	-	LD				4	4	4	1	-	-			-	-	-
dcbtst. TH10000	Data Cache Block Touch for Store	-	-	LD				4	4	4	1	-	-			-	-	-
dcbtst. TH10001	Data Cache Block Touch for Store	-	-	LD				4	4	4	1	-	-			-	-	-
dcbtst. TH1001x	Data Cache Block Touch for Store	-	-	LD				4	4	4	1	-	-			-	-	-

Table A-1. Instruction Properties (Sheet 8 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
dcbtst. TH101xx	Data Cache Block Touch for Store	-	-	LD				4	4	4	1	-	-			-	-	-
dcbtst. TH11000	Data Cache Block Touch for Store	-	-	LD				4	4	4	1	-	-			-	-	-
dcbtst. TH11001	Data Cache Block Touch for Store	-	-	LD				4	4	4	1	-	-			-	-	-
dcbtst. TH1101x	Data Cache Block Touch for Store	-	-	LD				4	4	4	1	-	-			-	-	-
dcbtst. TH111xx	Data Cache Block Touch for Store	-	-	LD				4	4	4	1	-	-			-	-	-
dcbz	Data Cache Block Zero	-	-	LD				4	4	4	1	-	-			-	-	-
dcffix	DFP Convert From Fixed	-	-	DFU	FPR		fpcc,fric,excp	1/21	32	32	20	E	-		3	-	-	-
dcffix.	DFP Convert From Fixed and Record	C2	1	DFU	FPR		fpcc,fric,excp	1/21	32	32	20	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
dcffixq	DFP Convert From Fixed Quad	C2	1	DFU	FPR		fpcc,fric,excp	1/21	32	32	20	P	-		3	S	-	-
			2	DFU	FPR			1/21	32	32	20	P	-		3	S	-	-
dcffixq.	DFP Convert From Fixed Quad and Record	C3	1	DFU	FPR		fpcc,fric	1/21	32	32	20	P	-		3	S	-	-
			2	DFU	FPR			1/21	32	32	20	P	-		3	S	-	-
			3	ALU2		CR		4	3	3	1	-	-		F	-	-	F
dcmpo	DFP Compare Ordered	-	-	DFU		CR	excp,fpcc	1	12	12	1	E	-		3	-	-	-
dcmpoq	DFP Compare Ordered Quad	C3	1	ALU	VSR			2	2	2	1	V	-	1		S	-	-
			2	DFU		CR	fpcc,excp	1	12	12	1	P	-		3	S	-	-
			3	DFU				1	12	12	1	P	-		3	S	-	-
dcmpu	DFP Compare Unordered	-	-	DFU		CR	excp,fpcc	1	12	12	1	E	-		3	-	-	-

Table A-1. Instruction Properties (Sheet 9 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
dcmpuq	DFP Compare Unordered Quad	C3	1	ALU	VSR			2	2	2	1	V	-	1		S	-	-
			2	DFU		CR	fpcc,excp	1	12	12	1	P	-		3	S	-	-
			3	DFU				1	12	12	1	P	-		3	S	-	-
dctdp	DFP Convert to DFP Long	-	-	DFU	FPR		fpcc,fric	1	12	12	1	E	-		3	-	-	-
dctdp.	DFP Convert to DFP Long and Record	C2	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
dctfix	DFP Convert to Fixed	-	-	DFU	FPR		fpcc,fric,excp	1/14	25	25	13	E	-		3	-	-	-
dctfix.	DFP Convert to Fixed and Record	C2	1	DFU	FPR		fpcc,fric,excp	1/14	25	25	13	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
dctfixq	DFP Convert to Fixed Quad	C2	1	DFU	FPR		fpcc,fric,excp	1/14	25	25	13	P	-		3	S	-	-
			2	DFU				1/14	25	25	13	P	-		3	S	-	-
dctfixq.	DFP Convert to Fixed Quad and Record	C3	1	DFU	FPR		fpcc,fric,excp	1/14	25	25	13	P	-		3	S	-	-
			2	DFU				1/14	25	25	13	P	-		3	S	-	-
			3	ALU2		CR		4	3	3	1	-	-		F	-	D2P	F
dctqpq	DFP Convert to DFP Extended	C2	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-
dctqpq.	DFP Convert to DFP Extended and Record	C3	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			3	ALU2		CR		4	3	3	1	-	-		F	-	D2P	F
ddedpd	DFP Decode DPD to BCD	-	-	DFU	FPR			1	12	12	1	E	-			-	-	-
ddedpd.	DFP Decode DPD to BCD and Record	C2	1	DFU	FPR			1	12	12	1	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	-	F

Table A-1. Instruction Properties (Sheet 10 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
ddedpdq	DFP Decode DPD to BCD Quad	C2	1	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-
ddedpdq.	DFP Decode DPD to BCD Quad and Record	C3	1	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			3	ALU2		CR		4	3	3	1	-	-		F	-	-	F
ddiv	DFP Divide	-	-	DFU	FPR		fpcc,fric,excp	1/28	39	99	27	E	-		3	-	-	-
ddiv.	DFP Divide and Record	C2	1	DFU	FPR		fpcc,fric,excp	1/28	39	99	27	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
ddivq	DFP Divide Quad	C3	1	ALU	VSR			2	2	2	1	V	-	1		S	-	-
			2	DFU	FPR		fpcc,fric,excp	1/28	39	171	27	P	-		3	S	-	-
			3	DFU	FPR			1/28	39	171	27	P	-		3	S	-	-
ddivq.	DFP Divide Quad and Record	X	1	ALU	VSR			2	2	2	1	V	-	1		S	-	-
			2	NOP				6	0	0	1	-	-			-	-	-
			3	NOP				6	0	0	1	-	-			-	-	-
			4	DFU	FPR		fpcc,fric,excp	1/28	39	171	27	P	-		3	S	-	-
			5	DFU	FPR			1/28	39	171	27	P	-		3	S	-	-
			6	ALU2		CR		4	3	3	1	-	-		F	-	-	F
denbcd	DFP Encode BCD to DPD	-	-	DFU	FPR		fpcc,fric,excp	1	12	12	1	E	-		3	-	-	-
denbcd.	DFP Encode BCD to DPD and Record	C2	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
denbcdq	DFP Encode BCD to DPD Quad	C2	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-

Table A-1. Instruction Properties (Sheet 11 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
denbcdq.	DFP Encode BCD to DPD Quad and Record	C3	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			3	ALU2		CR		4	3	3	1	-	-		F	-	D2P	F
diex	DFP Insert Exponent	-	-	DFU	FPR			1	12	12	1	E	-			-	-	-
diex.	DFP Insert Exponent and Record	C2	1	DFU	FPR			1	12	12	1	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	-	F
diexq	DFP Insert Exponent Quad	C2	1	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-
diexq.	DFP Insert Exponent Quad and Record	C3	1	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			3	ALU2		CR		4	3	3	1	-	-		F	-	-	F
divd	Divide Dword	-	-	DIV	GPR			2/9	12	24	8	E	-			-	-	-
divd.	Divide Dword and Record	C2	1	DIV	GPR			2/9	12	24	8	E	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
divde	Divide Dword Extended	-	-	DIV	GPR			2/9	12	40	8	E	-			-	-	-
divde.	Divide Dword Extended and Record	C2	1	DIV	GPR			2/9	12	40	8	E	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
divdeo	Divide Dword Extended and Record OV	-	-	DIV	GPR		ov	2/9	12	40	8	E	-			-	-	-
divdeo.	Divide Dword Extended and Record OV and Record	C2	1	DIV	GPR		ov	2/9	12	40	8	E	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-		3	-	D1P	-
divdeu	Divide Dword Extended Unsigned	-	-	DIV	GPR			2/9	12	40	8	E	-			-	-	-

Table A-1. Instruction Properties (Sheet 12 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
divdeu.	Divide Dword Extended Unsigned and Record	C2	1	DIV	GPR			2/9	12	40	8	E	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
divdeuo	Divide Dword Extended Unsigned and Record OV	-	-	DIV	GPR		ov	2/9	12	40	8	E	-			-	-	-
divdeuo.	Divide Dword Extended Unsigned and Record OV and Record	C2	1	DIV	GPR		ov	2/9	12	40	8	E	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-		3	-	D1P	-
divdo	Divide Dword and Record OV	-	-	DIV	GPR		ov	2/9	12	24	8	E	-			-	-	-
divdo.	Divide Dword and Record OV and Record	C2	1	DIV	GPR		ov	2/9	12	24	8	E	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-		3	-	D1P	-
divdu	Divide Dword Unsigned	-	-	DIV	GPR			2/9	12	24	8	E	-			-	-	-
divdu.	Divide Dword Unsigned and Record	C2	1	DIV	GPR			2/9	12	24	8	E	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
divduo	Divide Dword Unsigned and Record OV	-	-	DIV	GPR		ov	2/9	12	24	8	E	-			-	-	-
divduo.	Divide Dword Unsigned and Record OV and Record	C2	1	DIV	GPR		ov	2/9	12	24	8	E	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-		3	-	D1P	-
divw	Divide Word	-	-	DIV	GPR			2/9	12	16	8	E	-			-	-	-
divw.	Divide Word and Record	C2	1	DIV	GPR			2/9	12	16	8	E	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
divwe	Divide Word Extended	-	-	DIV	GPR			2/9	12	24	8	E	-			-	-	-
divwe.	Divide Word Extended and Record	C2	1	DIV	GPR			2/9	12	24	8	E	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
divweo	Divide Word Extended and Record OV	-	-	DIV	GPR		ov	2/9	12	24	8	E	-			-	-	-

Table A-1. Instruction Properties (Sheet 13 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
divweo.	Divide Word Extended and Record OV and Record	C2	1	DIV	GPR		ov	2/9	12	24	8	E	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-		3	-	D1P	-
divweu	Divide Word Extended Unsigned	-	-	DIV	GPR			2/9	12	24	8	E	-			-	-	-
divweu.	Divide Word Extended Unsigned and Record	C2	1	DIV	GPR			2/9	12	24	8	E	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
divweuo	Divide Word Extended Unsigned and Record OV	-	-	DIV	GPR		ov	2/9	12	24	8	E	-			-	-	-
divweuo.	Divide Word Extended Unsigned and Record OV and Record	C2	1	DIV	GPR		ov	2/9	12	24	8	E	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-		3	-	D1P	-
divwo	Divide Word and Record OV	-	-	DIV	GPR		ov	2/9	12	16	8	E	-			-	-	-
divwo.	Divide Word and Record OV and Record	C2	1	DIV	GPR		ov	2/9	12	16	8	E	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-		3	-	D1P	-
divwu	Divide Word Unsigned	-	-	DIV	GPR			2/9	12	16	8	E	-			-	-	-
divwu.	Divide Word Unsigned and Record	C2	1	DIV	GPR			2/9	12	16	8	E	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
divwuo	Divide Word Unsigned and Record OV	-	-	DIV	GPR		ov	2/9	12	16	8	E	-			-	-	-
divwuo.	Divide Word Unsigned and Record OV and Record	C2	1	DIV	GPR		ov	2/9	12	16	8	E	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-		3	-	D1P	-
dmul	DFP Multiply	-	-	DFU	FPR		fpcc,fric,excp	1/11	24	39	12	E	-		3	-	-	-
dmul.	DFP Multiply and Record	C2	1	DFU	FPR		fpcc,fric,excp	1/11	22	27	10	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F

Table A-1. Instruction Properties (Sheet 14 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
dmulq	DFP Multiply Quad	C3	1	ALU	VSR			2	2	2	1	V	-	1		S	-	-
			2	DFU	FPR		fpcc,fric,excp	1/7	18	84	6	P	-		3	S	-	-
			3	DFU	FPR			1/7	18	84	6	P	-		3	S	-	-
dmulq.	DFP Multiply Quad and Record	X	1	ALU	VSR			2	2	2	1	V	-	1		S	-	-
			2	NOP				6	0	0	1	-	-			-	-	-
			3	NOP				6	0	0	1	-	-			-	-	-
			4	DFU	FPR		fpcc,fric,excp	1/7	18	84	6	P	-		3	S	-	-
			5	DFU	FPR			1/7	18	84	6	P	-		3	S	-	-
			6	ALU2		CR		4	3	3	1	-	-		F	-	-	F
dqua	DFP Quantize	-	-	DFU	FPR		fpcc,fric,excp	1	12	12	1	E	-		3	-	-	-
dqua.	DFP Quantize and Record	C2	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
dquai	DFP Quantize Immediate	-	-	DFU	FPR		fpcc,fric,excp	1	12	12	1	E	-		3	-	-	-
dquai.	DFP Quantize Immediate and Record	C2	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
dquaiq	DFP Quantize Immediate Quad	C2	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-
dquaiq.	DFP Quantize Immediate Quad and Record	C3	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			3	ALU2		CR		4	3	3	1	-	-		F	-	D2P	F
dquaq	DFP Quantize Quad	C3	1	ALU	VSR			2	2	2	1	V	-	1		S	-	-
			2	DFU	FPR		fpcc,fric,excp	1	12	12	1	P	-		3	S	-	-
			3	DFU	FPR			1	12	12	1	P	-		3	S	-	-

Table A-1. Instruction Properties (Sheet 15 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
dquaq.	DFP Quantize Quad and Record	X	1	ALU	VSR			2	2	2	1	V	-	1		S	-	-
			2	NOP				6	0	0	1	-	-			-	-	-
			3	NOP				6	0	0	1	-	-			-	-	-
			4	DFU	FPR		fpcc,fric,excp	1	12	12	1	P	-		3	S	-	-
			5	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			6	ALU2		CR		4	3	3	1	-	-		F	-	-	F
drdpq	DFP Round to DFP Long	C2	1	DFU	FPR		fpcc,fric,excp	1/13	24	24	12	P	-		3	S	-	-
			2	DFU	FPR			1/13	24	24	12	P	-		3	S	-	-
drdpq.	DFP Round to DFP Long and Record	C3	1	DFU	FPR		fpcc,fric,excp	1/13	24	24	12	P	-		3	S	-	-
			2	DFU	FPR			1/13	24	24	12	P	-		3	S	-	-
			3	ALU2		CR		4	3	3	1	-	-		F	-	D2P	F
drintn	DFP Round to FP Integer Without Inexact	-	-	DFU	FPR		fpcc,fric,excp	1	12	12	1	E	-		3	-	-	-
drintn.	DFP Round to FP Integer Without Inexact and Record	C2	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
drintnq	DFP Round to FP Integer Without Inexact Quad	C2	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-
drintnq.	DFP Round to FP Integer Without Inexact Quad and Record	C3	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			3	ALU2		CR		4	3	3	1	-	-		F	-	D2P	F
drintx	DFP Round to FP Integer With Inexact	-	-	DFU	FPR		fpcc,fric,excp	1	12	12	1	E	-		3	-	-	-
drintx.	DFP Round to FP Integer With Inexact and Record	C2	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F

Table A-1. Instruction Properties (Sheet 16 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
drintxq	DFP Round to FP Integer With Inexact Quad	C2	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-
drintxq.	DFP Round to FP Integer With Inexact Quad and Record	C3	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			3	ALU2		CR		4	3	3	1	-	-		F	-	D2P	F
drrnd	DFP Reround	-	-	DFU	FPR		fpcc,fric,excp	1	12	12	1	E	-		3	-	-	-
drrnd.	DFP Reround and Record	C2	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
drrndq	DFP Reround Quad	C2	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-
drrndq.	DFP Reround Quad and Record	C3	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			3	ALU2		CR		4	3	3	1	-	-		F	-	D2P	F
drsp	DFP Round to DFP Short	-	-	DFU	FPR		fpcc,fric,excp	1/13	24	24	12	E	-		3	-	-	-
drsp.	DFP Round to DFP Short and Record	C2	1	DFU	FPR		fpcc,fric,excp	1/13	24	24	12	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
dscli	DFP Shift Significand Left Immediate	-	-	DFU	FPR			1	12	12	1	E	-			-	-	-
dscli.	DFP Shift Significand Left Immediate and Record	C2	1	DFU	FPR			1	12	12	1	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	-	F
dscliq	DFP Shift Significand Left Immediate Quad	C2	1	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-

Table A-1. Instruction Properties (Sheet 17 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
dscliq.	DFP Shift Significand Left Immediate Quad and Record	C3	1	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			3	ALU2		CR		4	3	3	1	-	-		F	-	-	F
dscri	DFP Shift Significand Right Immediate	-	-	DFU	FPR			1	12	12	1	E	-			-	-	-
dscri.	DFP Shift Significand Right Immediate and Record	C2	1	DFU	FPR			1	12	12	1	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	-	F
dscriq	DFP Shift Significand Right Immediate Quad	C2	1	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-
dscriq.	DFP Shift Significand Right Immediate Quad and Record	C3	1	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			2	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			3	ALU2		CR		4	3	3	1	-	-		F	-	-	F
dsub	DFP Subtract	-	-	DFU	FPR		fpcc,fric,excp	1	12	12	1	E	-		3	-	-	-
dsub.	DFP Subtract and Record	C2	1	DFU	FPR		fpcc,fric,excp	1	12	12	1	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
dsubq	DFP Subtract Quad	C3	1	ALU	VSR			2	2	2	1	V	-	1		S	-	-
			2	DFU	FPR		fpcc,fric,excp	1	12	12	1	P	-		3	S	-	-
			3	DFU	FPR			1	12	12	1	P	-		3	S	-	-

Table A-1. Instruction Properties (Sheet 18 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
dsubq.	DFP Subtract Quad and Record	X	1	ALU	VSR			2	2	2	1	V	-	1		S	-	-
			2	NOP				6	0	0	1	-	-			-	-	-
			3	NOP				6	0	0	1	-	-			-	-	-
			4	DFU	FPR		fpcc,fric,excp	1	12	12	1	P	-		3	S	-	-
			5	DFU	FPR			1	12	12	1	P	-		3	S	-	-
			6	ALU2		CR		4	3	3	1	-	-		F	-	-	F
dtstdc	DFP Test Data Class	-	-	DFU		CR	fpcc	1	12	12	1	E	-			-	-	-
dtstdcq	DFP Test Data Class Quad	C2	1	DFU		CR	fpcc	1	12	12	1	P	-		3	S	-	-
			2	DFU				1	12	12	1	P	-		3	S	-	-
dtstdg	DFP Test Data Group	-	-	DFU		CR	fpcc	1	12	12	1	E	-			-	-	-
dtstdgq	DFP Test Data Group Quad	C2	1	DFU		CR	fpcc	1	12	12	1	P	-		3	S	-	-
			2	DFU				1	12	12	1	P	-		3	S	-	-
dtstex	DFP Test Exponent	-	-	DFU		CR	fpcc	1	12	12	1	E	-		3	-	-	-
dtstexq	DFP Test Exponent Quad	-	-	DFU		CR	fpcc	1	12	12	1	E	-		3	-	-	-
dtstsf	DFP Test Significance	-	-	DX		CR	fpcc	2	3	3	1	E	-		3	-	-	-
dtstsfi	DFP Test Significance Immediate	-	-	DX		CR	fpcc	2	3	3	1	E	-			-	-	-
dtstsfiq	DFP Test Significance Immediate Quad	C2	1	DX		CR	fpcc	2	3	3	1	P	-			-	-	-
			2	DX				2	3	3	1	P	-			-	-	-
dtstsfq	DFP Test Significance Quad	C2	1	DX		CR	fpcc	2	3	3	1	P	-			-	-	-
			2	DX				2	3	3	1	P	-			-	-	-
dxex	DFP Extract Exponent	-	-	DFU	FPR			1	12	12	1	E	-			-	-	-
dxex.	DFP Extract Exponent and Record	C2	1	DFU	FPR			1	12	12	1	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	-	F

Table A-1. Instruction Properties (Sheet 19 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
dxexq	DFP Extract Exponent Quad	-	-	DFU	FPR			1	12	12	1	E	-			-	-	-
dxexq.	DFP Extract Exponent Quad and Record	C2	1	DFU	FPR			1	12	12	1	E	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	-	F
eieio	Enforce In-order Execution of I/O	-	-	LD				4	4	4	1	-	-			-	-	-
eqv	Equivalent	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
eqv.	Equivalent and Record	-	-	ALU	GPR	CR	fxcc	4	2	2	1	-	-			-	-	-
extsb	Extend Sign Byte	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
extsb.	Extend Sign Byte and Record	-	-	ALU	GPR	CR	fxcc	4	2	2	1	-	-			-	-	-
extsh	Extend Sign Hword	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
extsh.	Extend Sign Hword and Record	-	-	ALU	GPR	CR	fxcc	4	2	2	1	-	-			-	-	-
extsw	Extend Sign Word	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
extsw.	Extend Sign Word and Record	-	-	ALU	GPR	CR	fxcc	4	2	2	1	-	-			-	-	-
extswsli	Extend Sign Word and Shift Left Immediate	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
extswsli.	Extend Sign Word and Shift Left Immediate and Record	C2	1	ALU	GPR			4	2	2	1	-	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
fabs	Floating Absolute	-	-	ALU	FPR			4	2	2	1	R	-			-	-	-
fabs.	Floating Absolute and Record	C2	1	ALU	FPR			4	2	2	1	R	-			-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	-	F
fadd	Floating Add	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fadd.	Floating Add and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fadds	Floating Add Single	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-

Table A-1. Instruction Properties (Sheet 20 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
fadds.	Floating Add Single and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fcfid	Floating Convert From Integer Dword	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fcfid.	Floating Convert From Integer Dword and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fcfids	Floating Convert From Integer Dword Single	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fcfids.	Floating Convert From Integer Dword Single and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fcfidu	Floating Convert From Integer Dword Unsigned	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fcfidu.	Floating Convert From Integer Dword Unsigned and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fcfidus	Floating Convert From Integer Dword Unsigned Single	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fcfidus.	Floating Convert From Integer Dword Unsigned Single and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fcmpo	Floating Compare Ordered	-	-	ALU2		CR	excp,fpcc	4	3	3	1	R	-		3	-	-	-
fcmpu	Floating Compare Unordered	-	-	ALU2		CR	excp,fpcc	4	3	3	1	R	-		3	-	-	-
fcpsgn	Floating Copy Sign	-	-	ALU	FPR			4	2	2	1	R	-			-	-	-
fcpsgn.	Floating Copy Sign and Record	C2	1	ALU	FPR			4	2	2	1	R	-			-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	-	F
fctid	Floating Convert to Integer Dword	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-

Table A-1. Instruction Properties (Sheet 21 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
fctid.	Floating Convert to Integer Dword and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fctidu	Floating Convert to Integer Dword Unsigned	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fctidu.	Floating Convert to Integer Dword Unsigned and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fctiduz	Floating Convert to Integer Dword Unsigned truncate	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fctiduz.	Floating Convert to Integer Dword Unsigned truncate and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fctidz	Floating Convert to Integer Dword truncate	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fctidz.	Floating Convert to Integer Dword truncate and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fctiw	Floating Convert to Integer Word	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fctiw.	Floating Convert to Integer Word and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fctiwu	Floating Convert to Integer Word Unsigned	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fctiwu.	Floating Convert to Integer Word Unsigned and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fctiwuz	Floating Convert to Integer Word Unsigned truncate	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fctiwuz.	Floating Convert to Integer Word Unsigned truncate and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F

Table A-1. Instruction Properties (Sheet 22 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
fctiwz	Floating Convert to Integer Word truncate	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fctiwz.	Floating Convert to Integer Word truncate and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fdiv	Floating Divide	-	-	DP	FPR		fpcc,fric,excp	4/21	27	33	7-8	R	-		3	-	-	-
fdiv.	Floating Divide and Record	C2	1	DP	FPR		fpcc,fric,excp	4/21	27	33	7-8	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fdivs	Floating Divide Single	-	-	DP	FPR		fpcc,fric,excp	4/20	22	22	5	R	-		3	-	-	-
fdivs.	Floating Divide Single and Record	C2	1	DP	FPR		fpcc,fric,excp	4/20	22	22	5	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fmadd	Floating Multiply-Add	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fmadd.	Floating Multiply-Add and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fmadds	Floating Multiply-Add Single	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fmadds.	Floating Multiply-Add Single and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fmr	Floating Move Register	-	-	ALU	FPR			4	2	2	1	R	-			-	-	-
fmr.	Floating Move Register and Record	C2	1	ALU	FPR			4	2	2	1	R	-			-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	-	F
fmrgew	Floating Merge Even Word	-	-	ALU	FPR			4	2	2	1	R	-			-	-	-
fmrgow	Floating Merge Odd Word	-	-	ALU	FPR			4	2	2	1	R	-			-	-	-
fmsub	Floating Multiply-Subtract	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-

Table A-1. Instruction Properties (Sheet 23 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
fmsub.	Floating Multiply-Subtract and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fmsubs	Floating Multiply-Subtract Single	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fmsubs.	Floating Multiply-Subtract Single and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fmul	Floating Multiply	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fmul.	Floating Multiply and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fmuls	Floating Multiply Single	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fmuls.	Floating Multiply Single and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fnabs	Floating Negative Absolute Value	-	-	ALU	FPR			4	2	2	1	R	-			-	-	-
fnabs.	Floating Negative Absolute Value and Record	C2	1	ALU	FPR			4	2	2	1	R	-			-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	-	F
fneg	Floating Negate	-	-	ALU	FPR			4	2	2	1	R	-			-	-	-
fneg.	Floating Negate and Record	C2	1	ALU	FPR			4	2	2	1	R	-			-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	-	F
fnmadd	Floating Negative Multiply-Add	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fnmadd.	Floating Negative Multiply-Add and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fnmadds	Floating Negative Multiply-Add Single	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-

Table A-1. Instruction Properties (Sheet 24 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
fnmadds.	Floating Negative Multiply-Add Single and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fnmsub	Floating Negative Multiply-Subtract	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fnmsub.	Floating Negative Multiply-Subtract and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fnmsubs	Floating Negative Multiply-Subtract Single	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fnmsubs.	Floating Negative Multiply-Subtract Single and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fre	Floating Reciprocal Estimate	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fre.	Floating Reciprocal Estimate and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fres	Floating Reciprocal Estimate Single	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fres.	Floating Reciprocal Estimate Single and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
frim	Floating Round to Integer Minus	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
frim.	Floating Round to Integer Minus and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
frin	Floating Round to Integer Nearest	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
frin.	Floating Round to Integer Nearest and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
frip	Floating Round to Integer Plus	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-

Table A-1. Instruction Properties (Sheet 25 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
frip.	Floating Round to Integer Plus and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
friz	Floating Round to Integer Zero	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
friz.	Floating Round to Integer Zero and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
frsp	Floating Round to SP	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
frsp.	Floating Round to SP and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
frsqrte	Floating Reciprocal Square Root Estimate	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
frsqrte.	Floating Reciprocal Square Root Estimate and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
frsqrtes	Floating Reciprocal Square Root Estimate Single	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
frsqrtes.	Floating Reciprocal Square Root Estimate Single and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fsel	Floating Select	-	-	DP	FPR			4	5	7	1	R	-			-	-	-
fsel.	Floating Select and Record	C2	1	DP	FPR			4	5	7	1	R	-			-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	-	F
fsqrt	Floating Square Root	-	-	DP	FPR		fpcc,fric,excp	4/37	36	36	10	R	-		3	-	-	-
fsqrt.	Floating Square Root and Record	C2	1	DP	FPR		fpcc,fric,excp	4/37	36	36	10	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fsqrts	Floating Square Root Single	-	-	DP	FPR		fpcc,fric,excp	4/20	26	26	5	R	-		3	-	-	-

Table A-1. Instruction Properties (Sheet 26 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
fsqrts.	Floating Square Root Single and Record	C2	1	DP	FPR		fpcc,fric,excp	4/20	26	26	5	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fsub	Floating Subtract	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fsub.	Floating Subtract and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
fsubs	Floating Subtract Single	-	-	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
fsubs.	Floating Subtract Single and Record	C2	1	DP	FPR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
ftdiv	Floating Test for Software Divide	-	-	ALU2		CR		4	3	3	1	R	-			-	-	-
ftsqrt	Floating Test for Software Square Root	-	-	ALU2		CR		4	3	3	1	R	-			-	-	-
icbi	Instruction Cache Block Invalidate	-	-	LD				4	4	4	1	-	-			-	-	N
icbt	Instruction Cache Block Touch	-	-	LD				4	4	4	1	-	-			-	-	-
isel	Integer Select	-	-	ALU	GPR			4	2	2	1	R	-		3	-	-	-
isync	Instruction Synchronize	-	-	LD				4	4	4	1	-	-			-	-	-
lbarx	Load Byte And Reserve Indexed	-	-	LD	GPR			4	4	4	1	-	-			-	-	-
lbz	Load Byte and Zero	-	-	LD	GPR			4	4	4	1	-	-			-	-	-
lbzcix	Load Byte and Zero Caching Inhibited Indexed	-	-	LD	GPR			4	4	4	1	-	-			-	-	-
lbzu	Load Byte and Zero with Update	C2	1	LD	GPR			4	4	4	1	-	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	-	-
lbzux	Load Byte and Zero with Update Indexed	C2	1	LD	GPR			4	4	4	1	P	-			-	-	-
			2	ALU	GPR			4	2	2	1	P	-			-	-	-
lbzx	Load Byte and Zero Indexed	-	-	LD	GPR			4	4	4	1	-	-			-	-	-

Table A-1. Instruction Properties (Sheet 27 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
ld	Load Dword	-	-	LD	GPR			4	4	4	1	-	-			-	-	-
ldarx	Load Dword And Reserve Indexed	-	-	LD	GPR			4	4	4	1	-	-			-	-	-
ldat	Load Dword Atomic	X	1	NOP				6	0	0	1	-	-			-	-	-
			2	ST				4	-	-	1	R	-			-	-	-
			3	ST				4	-	-	1	R	-			-	-	-
			4	LD				4	4	4	1	-	-			-	-	-
			5	LD	GPR			4	4	4	1	-	-			-	-	N
			6	LD				4	4	4	1	-	-			-	-	-
ldbrx	Load Dword Byte-Reverse Indexed	-	-	LD	GPR			4	4	4	1	-	-			-	-	-
ldcix	Load Dword Caching Inhibited Indexed	-	-	LD	GPR			4	4	4	1	-	-			-	-	-
ldmx	Load Dword Monitored Indexed	C2	1	LD	GPR			4	4	4	1	-	-			-	-	-
			2	NOP				6	0	0	1	-	-			-	-	-
ldmx	Load Dword Monitored Indexed	C2	1	LD	GPR			4	4	4	1	-	-			-	-	-
			2	NOP				6	0	0	1	-	-			-	-	-
ldmx	Load Dword Monitored Indexed	C2	1	LD	GPR			4	4	4	1	-	-			-	-	-
			2	LD				4	4	4	1	-	-			-	D1P	-
ldu	Load Dword with Update	C2	1	LD	GPR			4	4	4	1	-	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	-	-
ldux	Load Dword with Update Indexed	C2	1	LD	GPR			4	4	4	1	P	-			-	-	-
			2	ALU	GPR			4	2	2	1	P	-			-	-	-
ldx	Load Dword Indexed	-	-	LD	GPR			4	4	4	1	-	-			-	-	-
lfd	Load Floating Double	-	-	LD	FPR			4	4	4	1	R	-			-	-	-

Table A-1. Instruction Properties (Sheet 28 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
lfdp	Load Floating Double Pair	C2	1	LD3	FPR			4	6	6	1	P	-			-	-	-
			2	LD3	FPR			4	6	6	1	P	-			-	-	-
lfdpx	Load Floating Double Pair Indexed	C2	1	LD3	FPR			4	6	6	1	P	-			-	-	-
			2	LD3	FPR			4	6	6	1	P	-			-	-	-
lfdu	Load Floating Double with Update	C2	1	LD	FPR			4	4	4	1	R	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	-	-
lfdux	Load Floating Double with Update Indexed	C2	1	LD	FPR			4	4	4	1	R	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	-	-
lfdx	Load Floating Double Indexed	-	-	LD	FPR			4	4	4	1	R	-			-	-	-
lfiwax	Load Floating as Integer Word Algebraic Indexed	C2	1	LD	FPR			4	4	4	1	R	-			-	-	-
			2	ALU	FPR			4	2	2	1	R	-			-	D1P	-
lfiwzx	Load Floating as Integer Word and Zero Indexed	-	-	LD	FPR			4	4	4	1	R	-			-	-	-
lfs	Load Floating Single	C2	1	LD	FPR			4	4	4	1	R	-			-	-	-
			2	ALU2	FPR			4	3	3	1	R	-			-	D1P	-
lfsu	Load Floating Single with Update	C3	1	LD	FPR			4	4	4	1	R	-			-	-	-
			2	ALU2	FPR			4	3	3	1	R	-			-	D1P	-
			3	ALU	GPR			4	2	2	1	-	-			-	-	-
lfsux	Load Floating Single with Update Indexed	C3	1	LD	FPR			4	4	4	1	R	-			-	-	-
			2	ALU2	FPR			4	3	3	1	R	-			-	D1P	-
			3	ALU	GPR			4	2	2	1	-	-			-	-	-
lfsx	Load Floating Single Indexed	C2	1	LD	FPR			4	4	4	1	R	-			-	-	-
			2	ALU2	FPR			4	3	3	1	R	-			-	D1P	-

Table A-1. Instruction Properties (Sheet 29 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
lha	Load Hword Algebraic	C2	1	LD	GPR			4	4	4	1	-	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	D1P	-
lharx	Load Hword And Reserve Indexed Xform	-	-	LD	GPR			4	4	4	1	-	-			-	-	-
lhau	Load Hword Algebraic with Update	C3	1	LD	GPR			4	4	4	1	-	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	-	-
			3	ALU	GPR			4	2	2	1	-	-			-	D2P	-
lhaux	Load Hword Algebraic with Update Indexed	C3	1	LD	GPR			4	4	4	1	P	-			-	-	-
			2	ALU	GPR			4	2	2	1	P	-			-	-	-
			3	ALU	GPR			4	2	2	1	-	-			-	D2P	-
lhax	Load Hword Algebraic Indexed	C2	1	LD	GPR			4	4	4	1	-	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	D1P	-
lhbrx	Load Hword Byte-Reverse Indexed	-	-	LD	GPR			4	4	4	1	-	-			-	-	-
lhz	Load Hword and Zero	-	-	LD	GPR			4	4	4	1	-	-			-	-	-
lhzcix	Load Hword and Zero Caching Inhibited Indexed	-	-	LD	GPR			4	4	4	1	-	-			-	-	-
lhzu	Load Hword and Zero with Update	C2	1	LD	GPR			4	4	4	1	-	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	-	-
lhzux	Load Hword and Zero with Update Indexed	C2	1	LD	GPR			4	4	4	1	P	-			-	-	-
			2	ALU	GPR			4	2	2	1	P	-			-	-	-
lhzx	Load Hword and Zero Indexed	-	-	LD	GPR			4	4	4	1	-	-			-	-	-
lmw	Load Multiple Word	X	1u	LD	GPR			4	4	4	1	-	-			-	-	-

Table A-1. Instruction Properties (Sheet 30 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
lq	Load Qword	C2	1	LD3	GPR			4	6	6	1	P	-			-	-	-
			2	LD3	GPR			4	6	6	1	P	-			-	-	-
lqarx	Load Qword And Reserve Indexed	C2	1	LD3	GPR			4	6	6	1	P	-			-	-	-
			2	LD3	GPR			4	6	6	1	P	-			-	-	-
lswi	Load String Word Immediate	X	1u	LD	GPR			4	4	4	1	-	-			-	-	-
lswx	Load String Word Indexed	X	1	DIV				2	12	12	1	E	-		3	-	-	-
			2u	LD2	GPR			4	5	5	1	-	-			-	-	-
lvebx	Load Vector Element Byte Indexed	-	-	LD	VR			2	5	5	1	-	-			-	-	-
lvehx	Load Vector Element Hword Indexed	-	-	LD	VR			2	5	5	1	-	-			-	-	-
lvewx	Load Vector Element Word Indexed	-	-	LD	VR			2	5	5	1	-	-			-	-	-
lvsl	Load Vector for Shift Left	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
lvsr	Load Vector for Shift Right	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
lvx	Load Vector Indexed	-	-	LD	VR			2	5	5	1	-	-			-	-	-
lvxl	Load Vector Indexed Last	-	-	LD	VR			2	5	5	1	-	-			-	-	-
lwa	Load Word Algebraic	C2	1	LD	GPR			4	4	4	1	-	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	D1P	-
lwarx	Load Word and Reserve Indexed	-	-	LD	GPR			4	4	4	1	-	-			-	-	-

Table A-1. Instruction Properties (Sheet 31 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
lwat	Load Word Atomic	X	1	NOP				6	0	0	1	-	-			-	-	-
			2	ST				4	-	-	1	R	-			-	-	-
			3	ST				4	-	-	1	R	-			-	-	-
			4	LD				4	4	4	1	-	-			-	-	-
			5	LD	GPR			4	4	4	1	-	-			-	-	N
			6	LD				4	4	4	1	-	-			-	-	-
lwaux	Load Word Algebraic with Update Indexed	C3	1	LD	GPR			4	4	4	1	P	-			-	-	-
			2	ALU	GPR			4	2	2	1	P	-			-	-	-
			3	ALU	GPR			4	2	2	1	-	-			-	D2P	-
lwax	Load Word Algebraic Indexed	C2	1	LD	GPR			4	4	4	1	-	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	D1P	-
lwbrx	Load Word Byte-Reverse Indexed	-	-	LD	GPR			4	4	4	1	-	-			-	-	-
lwz	Load Word and Zero	-	-	LD	GPR			4	4	4	1	-	-			-	-	-
lwzcix	Load Word and Zero Caching Inhibited Indexed	-	-	LD	GPR			4	4	4	1	-	-			-	-	-
lwzu	Load Word and Zero with Update	C2	1	LD	GPR			4	4	4	1	-	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	-	-
lwzux	Load Word and Zero with Update Indexed	C2	1	LD	GPR			4	4	4	1	P	-			-	-	-
			2	ALU	GPR			4	2	2	1	P	-			-	-	-
lwzx	Load Word and Zero Indexed	-	-	LD	GPR			4	4	4	1	-	-			-	-	-
lxsd	Load VSX Scalar Dword	-	-	LD	VR			2	5	5	1	-	-			-	-	-
lxsdx	Load VSX Scalar Dword Indexed	-	-	LD	VSR			2	5	5	1	-	-			-	-	-
lxsibzx	Load VSX Scalar as Integer Byte and Zero Indexed	-	-	LD	VSR			2	5	5	1	-	-			-	-	-

Table A-1. Instruction Properties (Sheet 32 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
lxsihzx	Load VSX Scalar as Integer Hword and Zero Indexed	-	-	LD	VSR			2	5	5	1	-	-			-	-	-
lxsiwax	Load VSX Scalar as Integer Word Algebraic Indexed	C2	1	LD	VSR			2	5	5	1	-	-			-	-	-
			2	ALU	VSR			2	2	2	1	-	-			-	D1P	-
lxsiwzx	Load VSX Scalar as Integer Word and Zero Indexed	-	-	LD	VSR			2	5	5	1	-	-			-	-	-
lxssp	Load VSX Scalar Single	C2	1	LD	VR			2	5	5	1	-	-			-	-	-
			2	ALU2	VR			2	3	3	1	-	-			-	D1P	-
lxsspx	Load VSX Scalar SP Indexed	C2	1	LD	VSR			2	5	5	1	-	-			-	-	-
			2	ALU2	VSR			2	3	3	1	-	-			-	D1P	-
lxv	Load VSX Vector	-	-	LD	VSR			2	5	5	1	-	-			-	-	-
lxvb16x	Load VSX Vector Byte*16 Indexed	-	-	LD	VSR			2	5	5	1	-	-			-	-	-
lxvd2x	Load VSX Vector Dword*2 Indexed	-	-	LD	VSR			2	5	5	1	-	-			-	-	-
lxvdsx_be	Load VSX Vector Dword and Splat Indexed	C2	1	LD	VSR			2	5	5	1	-	-			-	-	-
			2	NOP				6	0	0	1	-	-			-	-	-
lxvdsx_le	Load VSX Vector Dword and Splat Indexed	C2	1	LD	VSR			2	5	5	1	-	-			-	-	-
			2	PM	VSR			2	3	3	1	V	-	1		S	D1P	-
lxvh8x_be	Load VSX Vector Hword*8 Indexed	C2	1	LD	VSR			2	5	5	1	-	-			-	-	-
			2	NOP				6	0	0	1	-	-			-	-	-
lxvh8x_le	Load VSX Vector Hword*8 Indexed	C2	1	LD	VSR			2	5	5	1	-	-			-	-	-
			2	PM	VSR			2	3	3	1	V	-	1		S	D1P	-
lxvl	Load VSX Vector with Length	-	-	LD2	VSR			2	6	6	1	-	-			-	-	-
lxvll	Load VSX Vector Left-justified with Length	-	-	LD2	VSR			2	6	6	1	-	-			-	-	-

Table A-1. Instruction Properties (Sheet 33 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
lxvw4x_be	Load VSX Vector Word*4 Indexed	C2	1	LD	VSR			2	5	5	1	-	-			-	-	-
			2	NOP				6	0	0	1	-	-			-	-	-
lxvw4x_le	Load VSX Vector Word*4 Indexed	C2	1	LD	VSR			2	5	5	1	-	-			-	-	-
			2	PM	VSR			2	3	3	1	V	-	1		S	D1P	-
lxvwsx_be	Load VSX Vector Word and Splat Indexed	C2	1	LD	VSR			2	5	5	1	-	-			-	-	-
			2	NOP				6	0	0	1	-	-			-	-	-
lxvwsx_le	Load VSX Vector Word and Splat Indexed	C2	1	LD	VSR			2	5	5	1	-	-			-	-	-
			2	NOP				6	0	0	1	-	-			-	-	-
lxvx	Load VSX Vector Indexed	-	-	LD	VSR			2	5	5	1	-	-			-	-	-
maddhd	Multiply-Add High Dword	-	-	DP-MUL	GPR			4	5	5	1	R	-			-	-	-
maddhdu	Multiply-Add High Dword Unsigned	-	-	DP-MUL	GPR			4	5	5	1	R	-			-	-	-
maddld	Multiply-Add Low Dword	-	-	DP-MUL	GPR			4	5	5	1	R	-			-	-	-
mcrf	Move CR Field	-	-	ALU		CR		4	2	2	1	-	-		3	-	-	-
mcrfs	Move to CR from FPSCR	C2	1	ALU2		CR		4	3	3	1	-	-		F	-	-	F
			2	ALU2			ctrl,excp_clr	4	3	3	1	-	-		3	-	-	N
mcrfs	Move to CR from FPSCR	-	-	ALU2		CR		4	3	3	1	-	-		3	-	-	-
mcrxrx	Move XER to CR Extended	-	-	ALU		CR		4	2	2	1	-	-		3	-	-	-
mfcr	Move From CR	C3	1	ALU	GPR			4	2	2	1	R	-		3	-	-	-
			2	ALU	GPR			4	2	2	1	R	-		3	-	D1P	-
			3	ALU	GPR			4	2	2	1	R	-		3	-	D1P	-

Table A-1. Instruction Properties (Sheet 34 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mffs	Move From FPSCR	C2	1	ALU2	FPR			4	3	3	1	R	-		3	-	-	-
			2	ALU2	FPR			4	3	3	1	R	-		F	-	D1P	F
mffs.	Move From FPSCR and Record	C2	1	ALU2	FPR			4	3	3	1	R	-		3	-	-	-
			2	ALU2	FPR	CR		4	3	3	1	R	-		F	-	D1P	F
mffscdrn	Move From FPSCR Control and set DRN	-	-	ALU2	FPR		ctrl	4	3	3	1	R	-		3	-	-	-
mffscdrni	Move From FPSCR Control and set DRN Immediate	-	-	ALU2	FPR		ctrl	4	3	3	1	R	-		3	-	-	-
mffsce	Move From FPSCR and Clear Enables	C2	1	ALU2	FPR		ctrl	4	3	3	1	R	-		3	-	-	-
			2	ALU2	FPR			4	3	3	1	R	-		F	-	D1P	F
mffscrn	Move From FPSCR Control and set RN	-	-	ALU2	FPR		ctrl	4	3	3	1	R	-		3	-	-	-
mffscrni	Move From FPSCR Control and set RN Immediate	-	-	ALU2	FPR		ctrl	4	3	3	1	R	-		3	-	-	-
mffsl	Move From FPSCR Lightweight	C2	1	ALU2	FPR			4	3	3	1	R	-		3	-	-	-
			2	ALU2	FPR			4	3	3	1	R	-		3	-	D1P	-
mfmsr	Move From MSR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfocrf	Move From One CR Field	-	-	ALU	GPR			4	2	2	1	R	-		3	-	-	-
mfspr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_acop	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_amor	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_amr	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_apscr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_apscru	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_asdr	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-

Table A-1. Instruction Properties (Sheet 35 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mfspr_bescr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_bescrr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_bescrru	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_bescrs	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_bescrsu	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_cfar	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_ciabr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_cir	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_ctr	Move From SPR	-	-	BR	GPR			1	6	6	1	E	-	2		-	-	-
mfspr_ctrl	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_dar	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_dawr0	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_dawrx0	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	N
mfspr_dec	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_dhdes	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_dpdes	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_dscr	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_dsisr	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_ebbhr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_ebbrr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_fscr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_gsr	Move From SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mfspr_hdar	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-

Table A-1. Instruction Properties (Sheet 36 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mfspr_hdec	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_hdsisr	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_heir	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_hfscr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_hid	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_hmeer	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_hmer	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_hpmc1	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_hpmc2	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_hpmc3	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_hpmc4	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_hrmor	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_hsprg0	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_hsprg1	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_hsrr0	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_hsrr1	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_iamr	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_ic	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_imc	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_l2hadsr	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_l2mvsr0	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_l2mvsr1	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_l2qosr	Move From SPR	-	-	NOP				6	0	0	1	-	-			-	-	-

Table A-1. Instruction Properties (Sheet 37 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mfspr_l3harpr	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_l3hawpr	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_ldbar	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_lmrr	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_lmrr	Move From SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mfspr_lmser	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_lmser	Move From SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mfspr_lpcr	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_lpidr	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_lr	Move From SPR	-	-	BR	GPR			1	6	6	1	E	-	2		-	-	-
mfspr_mmcr0	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_mmcr1	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_mmcr2	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_mmcra	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_mmcrc	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_mmcrh	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_mmcrs	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_mppr	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_pcr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_pidr	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_pir	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_pmc1	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_pmc2	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-

Table A-1. Instruction Properties (Sheet 38 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mfspr_pmc3	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_pmc4	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_pmc5	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_pmc6	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_pmcr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_pmicr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_pmmar	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_pmsr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_ppr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_ppr32	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_pspb	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_psscr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_ptcr	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_purr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_pvr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_reserved808	Move From SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mfspr_reserved809	Move From SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mfspr_reserved810	Move From SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mfspr_reserved811	Move From SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mfspr_rpr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_rwmr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N

Table A-1. Instruction Properties (Sheet 39 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mfspr_sdar	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	N
mfspr_siar	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_sier	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_smfctrl	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_spmc1	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_spmc2	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_sprc	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_sprd	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_sprg0	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_sprg1	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_sprg2	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_sprg3	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_spurr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_srr0	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_srr1	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_tar	Move From SPR	-	-	BR	GPR			1	6	6	1	E	-	2		-	-	-
mfspr_tb	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_tbl	Move From SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mfspr_tbu	Move From SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mfspr_tbu40	Move From SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mfspr_texasr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_texasru	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_tfhar	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N

Table A-1. Instruction Properties (Sheet 40 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mfspr_tfiar	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_tfmr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	N
mfspr_tidr	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_tir	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_trace	Move From SPR	-	-	DIV				2	12	12	1	-	-			-	-	-
mfspr_trig0	Move From SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mfspr_trig1	Move From SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mfspr_trig2	Move From SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mfspr_tscr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_tsr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_ttr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_uamor	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_uamr	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_urmor	Move From SPR	-	-	LD	GPR			4	4	4	1	E	WB,WS			-	-	-
mfspr_usprg0	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_usprg1	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_usrr0	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_usrr1	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_vr	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_worc	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_wort	Move From SPR	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfspr_xer	Move From SPR	C2	1	ALU2	GPR			4	3	3	1	R	-		3	-	-	-
			2	ALU2	GPR			4	3	3	1	R	-		3	-	D1P	-

Table A-1. Instruction Properties (Sheet 41 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mftb_old	Move From Time Base	-	-	DIV	GPR			2	12	12	1	E	WB,WS			-	-	-
mfvscr	Move From VSCR	-	-	ALU2	VR			2	3	3	1	V	-	1	3	S	-	N
mfvsrd	Move From VSR Dword	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
mfvsrld	Move From VSR Lower Dword	-	-	PM	GPR			4	3	3	1	V	-	1		S	-	-
mfvsrwz	Move From VSR Word and Zero	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
modsd	Modulo Signed Dword	-	-	DIV	GPR			2/9	12	24	8	E	-			-	-	-
modsw	Modulo Signed Word	-	-	DIV	GPR			2/9	12	16	8	E	-			-	-	-
modud	Modulo Unsigned Dword	-	-	DIV	GPR			2/9	12	24	8	E	-			-	-	-
moduw	Modulo Unsigned Word	-	-	DIV	GPR			2/9	12	24	8	E	-			-	-	-
msgclr	Message Clear	-	-	DIV				2	12	12	1	E	SS			-	-	N
msgclrp	Message Clear Privileged	-	-	DIV				2	12	12	1	E	SS			-	-	N
msgsnd	Message Send	-	-	LD				4	4	4	1	-	-			-	-	N
msgsndp	Message Send Privileged	-	-	DIV				2	12	12	1	E	SS			-	-	N
msgsync	Message Synchronize	-	-	LD				4	4	4	1	-	-			-	-	N
mtcrf	Move to CR Fields	C2	1	ALU		CR	CR*	4	2	2	1	R	-			-	-	-
			2	ALU		CR	CR*	4	2	2	1	R	-			-	-	-
mtfsb0_ctrl	Move to FPSCR Bit 0	-	-	ALU2			ctrl	4	3	3	1	R	-		3	-	-	-
mtfsb0_ctrl.	Move to FPSCR Bit 0	C2	1	ALU2			ctrl	4	3	3	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	-	F
mtfsb0_excp	Move to FPSCR Bit 0	-	-	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
mtfsb0_excp.	Move to FPSCR Bit 0	C2	1	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F

Table A-1. Instruction Properties (Sheet 42 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mtfsb0_excp.	Move to FPSCR Bit 0	C2	1	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
			2	ALU2		CR		4	3	3	1	-	-		F	-	D1P	F
mtfsb0_fpcc	Move to FPSCR Bit 0	-	-	ALU2			fpcc	4	3	3	1	R	-		3	-	-	-
mtfsb0_fpcc.	Move to FPSCR Bit 0	C2	1	ALU2			fpcc	4	3	3	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	-	F
mtfsb0_fric	Move to FPSCR Bit 0	-	-	ALU2			fric	4	3	3	1	R	-		3	-	-	-
mtfsb0_fric.	Move to FPSCR Bit 0	C2	1	ALU2			fric	4	3	3	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	-	F
mtfsb1_ctrl	Move to FPSCR Bit 1	C2	1	ALU2				4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl	4	3	3	1	R	-		3	-	-	-
mtfsb1_ctrl.	Move to FPSCR Bit 1	C2	1	ALU2		CR		4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl	4	3	3	1	R	-		3	-	-	-
mtfsb1_excp	Move to FPSCR Bit 1	-	-	ALU2			excp	4	3	3	1	R	-		3	-	-	-
mtfsb1_excp.	Move to FPSCR Bit 1	-	-	ALU2		CR	excp	4	3	3	1	R	-		F	-	-	F
mtfsb1_fpcc	Move to FPSCR Bit 1	-	-	ALU2			fpcc	4	3	3	1	R	-		3	-	-	-
mtfsb1_fpcc.	Move to FPSCR Bit 1	C2	1	ALU2			fpcc	4	3	3	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	-	F
mtfsb1_fric	Move to FPSCR Bit 1	-	-	ALU2			fric	4	3	3	1	R	-		3	-	-	-
mtfsb1_fric.	Move to FPSCR Bit 1	C2	1	ALU2			fric	4	3	3	1	R	-		3	-	-	-
			2	ALU2		CR		4	3	3	1	-	-		F	-	-	F
mtfsfi_ctrl	Move to FPSCR Field Immediate	C2	1	ALU2				4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl	4	3	3	1	R	-		3	-	-	-

Table A-1. Instruction Properties (Sheet 43 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mtfsfi_ctrl.	Move to FPSCR Field Immediate	C2	1	ALU2		CR		4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl	4	3	3	1	R	-		3	-	-	-
mtfsfi_drn	Move to FPSCR Field Immediate	-	-	ALU2			ctrl	4	3	3	1	R	-		3	-	-	-
mtfsfi_drn.	Move to FPSCR Field Immediate	C2	1	ALU2		CR		4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl	4	3	3	1	R	-		3	-	-	-
mtfsfi_excp0	Move to FPSCR Field Immediate	C2	1	ALU2			excp	4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
mtfsfi_excp0.	Move to FPSCR Field Immediate	C2	1	ALU2		CR	excp	4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
mtfsfi_excp2	Move to FPSCR Field Immediate	C2	1	ALU2			excp	4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
mtfsfi_excp2.	Move to FPSCR Field Immediate	C2	1	ALU2		CR	excp	4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
mtfsfi_excp5	Move to FPSCR Field Immediate	C2	1	ALU2			excp	4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
mtfsfi_excp5.	Move to FPSCR Field Immediate	C2	1	ALU2		CR	excp	4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
mtfsfi_fpcc	Move to FPSCR Field Immediate	-	-	ALU2			fpcc	4	3	3	1	R	-			-	-	-
mtfsfi_fpcc.	Move to FPSCR Field Immediate	-	-	ALU2		CR	fpcc	4	3	3	1	R	-		F	-	-	F
mtfsfi_fric	Move to FPSCR Field Immediate	C2	1	ALU2			fric,excp	4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
mtfsfi_fric.	Move to FPSCR Field Immediate	C2	1	ALU2		CR	fric,excp	4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N

Table A-1. Instruction Properties (Sheet 44 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mtfsf_l0_fpcc	Move to FPSCR Fields	C2	1	ALU2			fpcc,excp	4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
mtfsf_l0_fpcc.	Move to FPSCR Fields	C2	1	ALU2		CR	fpcc,excp	4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
mtfsf_l0_fpcc_fric	Move to FPSCR Fields	C2	1	ALU2			fric,fpcc,excp	4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
mtfsf_l0_fpcc_fric.	Move to FPSCR Fields	C2	1	ALU2		CR	fric,fpcc,excp	4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
mtfsf_l0_fric	Move to FPSCR Fields	C2	1	ALU2			fric,excp	4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
mtfsf_l0_fric.	Move to FPSCR Fields	C2	1	ALU2		CR	fric,excp	4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
mtfsf_l0_w0	Move to FPSCR Fields	C2	1	ALU2			excp	4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
mtfsf_l0_w0.	Move to FPSCR Fields	C2	1	ALU2		CR	excp	4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
mtfsf_l0_w1.	Move to FPSCR Fields	C2	1	ALU2		CR		4	3	3	1	R	-		F	-	-	F
			2	ALU2			ctrl	4	3	3	1	R	-		3	-	-	-
mtfsf_l1	Move to FPSCR Fields	C2	1	ALU2			fric,fpcc,excp	4	3	3	1	R	-		3	-	-	-
			2	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
mtfsf_l1.	Move to FPSCR Fields	C2	1	ALU2		CR	fric,fpcc,excp	4	3	3	1	R	-		3	-	-	-
			2	ALU2			ctrl,excp_clr	4	3	3	1	R	-		3	-	-	N
mtmsr	Move to MSR	-	-	DIV				2	12	12	1	E	-			-	-	N

Table A-1. Instruction Properties (Sheet 45 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mtmsrd	Move to MSR Dword	-	-	DIV				2	12	12	1	E	-			-	-	N
mtocrf	Move to One CR Field	-	-	ALU		CR		4	2	2	1	R	-			-	-	-
mtspr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_acop	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_amor	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_amr	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_apscr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_apscru	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_asdr	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_bescr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_bescrr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_bescrru	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_bescrs	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_bescrsu	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_cfar	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_ciabr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_cir	Move to SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mtspr_ctr	Move to SPR	-	-	DIV				2	5	5	1	E	-			-	-	-
mtspr_ctrl	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_dar	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_dawr0	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_dawrx0	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_dec	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N

Table A-1. Instruction Properties (Sheet 46 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mtspr_dhdes	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_dpdes	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_dscr	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_dsisr	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_ebbhr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_ebbrr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_fscr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_gsr	Move to SPR	-	-	DIV				2	12	12	1	-	-			-	-	-
mtspr_hdar	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_hdec	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_hdsisr	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_heir	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_hfscr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_hid	Move to SPR	C2	1	DIV				2	12	12	1	E	SS			-	-	N
			2	LD				4	4	4	1	-	SS			-	-	N
mtspr_hmeer	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_hmer	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_hpmc1	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_hpmc2	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_hpmc3	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_hpmc4	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_hrmor	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_hsprg0	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N

Table A-1. Instruction Properties (Sheet 47 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mtspr_hsprg1	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_hsrr0	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_hsrr1	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_iamr	Move to SPR	C2	1	LD				4	4	4	1	P	SS			-	-	N
			2	LD				4	4	4	1	P	SS			-	-	N
mtspr_ic	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_imc	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_l2hadsr	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_l2mvsr0	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_l2mvsr1	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_l2qosr	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_l3harpr	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_l3hawpr	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_ldbar	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_lmrr	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_lmser	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_lpcr	Move to SPR	C2	1	DIV				2	12	12	1	E	SS			-	-	N
			2	LD				4	4	4	1	-	SS			-	-	N
mtspr_lpidr	Move to SPR	C2	1	LD				4	4	4	1	P	SS			-	-	N
			2	LD				4	4	4	1	P	SS			-	-	N
mtspr_lr	Move to SPR	-	-	DIV				2	5	5	1	E	-			-	-	-
mtspr_mmcr0	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N

Table A-1. Instruction Properties (Sheet 48 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mtspr_mmcr1	Move to SPR	C2	1	DIV				2	12	12	1	E	SS			-	-	N
			2	LD				4	4	4	1	-	SS			-	-	N
mtspr_mmcr2	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_mmcra	Move to SPR	C2	1	DIV				2	12	12	1	E	SS			-	-	N
			2	LD				4	4	4	1	-	SS			-	-	N
mtspr_mmcrc	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_mmcrh	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_mmcrs	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_mppr	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_pcr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_pidr	Move to SPR	C2	1	LD				4	4	4	1	P	SS			-	-	N
			2	LD				4	4	4	1	P	SS			-	-	N
mtspr_pir	Move to SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mtspr_pmc1	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_pmc2	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_pmc3	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_pmc4	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_pmc5	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_pmc6	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_pmcr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_pmicr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_pmmar	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_pmsr	Move to SPR	-	-	NOP				6	0	0	1	-	-			-	-	-

Table A-1. Instruction Properties (Sheet 49 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mtspr_ppr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_ppr32	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_pspb	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_psscr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_ptcr	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_purr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_pvr	Move to SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mtspr_reserved808	Move to SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mtspr_reserved809	Move to SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mtspr_reserved810	Move to SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mtspr_reserved811	Move to SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mtspr_rpr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_rwmr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_sdar	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_siar	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_sier	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_smfctrl	Move to SPR	C2	1	DIV				2	12	12	1	E	SS			-	-	N
			2	LD				4	4	4	1	-	SS			-	-	N
mtspr_spmc1	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_spmc2	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_sprc	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N

Table A-1. Instruction Properties (Sheet 50 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mtspr_sprd	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_sprg0	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_sprg1	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_sprg2	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_sprg3	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_spurr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_srr0	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_srr1	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_tar	Move to SPR	-	-	DIV				2	5	5	1	E	-			-	-	-
mtspr_tb	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_tbl	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_tbu	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_tbu40	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_texasr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_texasru	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_tfhar	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_tfiar	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_tfmr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_tidr	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_tir	Move to SPR	-	-	NOP				6	0	0	1	-	-			-	-	-
mtspr_trace	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_trig0	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_trig1	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N

Table A-1. Instruction Properties (Sheet 51 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mtspr_trig2	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_tscr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_tsr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_ttr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_uamor	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_uamr	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_urmor	Move to SPR	-	-	LD				4	4	4	1	E	SS			-	-	N
mtspr_usprg0	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_usprg1	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_usrr0	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_usrr1	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_vr	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_worc	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_wort	Move to SPR	-	-	DIV				2	12	12	1	E	SS			-	-	N
mtspr_xer	Move to SPR	C2	1	ALU2			caoc,fxcc,ov,dcds	4	3	3	1	R	-			-	-	-
			2	ALU2			tgcc,string,reserved	4	3	3	1	R	-			-	-	-
mtvscr	Move to VSCR	-	-	ALU2			nj,sat	4	3	3	1	V	-	1		S	-	N
mtvsrd	Move to VSR Dword	-	-	ALU	VSR			2	2	2	1	-	-			-	-	-
mtvsrdd	Move to VSR Double Dword	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
mtvsrwa	Move to VSR Word Algebraic	-	-	ALU	VSR			2	2	2	1	-	-			-	-	-
mtvsrws	Move to VSR Word and Splat	-	-	PM	VSR			2	3	3	1	V	-	1		S	-	-
mtvsrwz	Move to VSR Word and Zero	-	-	ALU	VSR			2	2	2	1	-	-			-	-	-

Table A-1. Instruction Properties (Sheet 52 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mulhd	Multiply High Dword	-	-	DP-MUL	GPR			4	5	5	1	R	-			-	-	-
mulhd.	Multiply High Dword and Record	C2	1	DP-MUL	GPR			4	5	5	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
mulhdu	Multiply High Dword Unsigned	-	-	DP-MUL	GPR			4	5	5	1	R	-			-	-	-
mulhdu.	Multiply High Dword Unsigned and Record	C2	1	DP-MUL	GPR			4	5	5	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
mulhw	Multiply High Word	-	-	DP-MUL	GPR			4	5	5	1	R	-			-	-	-
mulhw.	Multiply High Word and Record	C2	1	DP-MUL	GPR			4	5	5	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
mulhwu	Multiply High Word Unsigned	-	-	DP-MUL	GPR			4	5	5	1	R	-			-	-	-
mulhwu.	Multiply High Word Unsigned and Record	C2	1	DP-MUL	GPR			4	5	5	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
mulld	Multiply Low Dword	-	-	DP-MUL	GPR			4	5	5	1	R	-			-	-	-
mulld.	Multiply Low Dword and Record	C2	1	DP-MUL	GPR			4	5	5	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
mulldo	Multiply Low Dword and Record OV	-	-	DP-MUL	GPR		ov	4	5	5	1	R	-			-	-	-

Table A-1. Instruction Properties (Sheet 53 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
mulldo.	Multiply Low Dword and Record OV and Record	C2	1	DP-MUL	GPR		ov	4	5	5	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-		3	-	D1P	-
mulli	Multiply Low Immediate	-	-	DP-MUL	GPR			4	5	5	1	R	-			-	-	-
mullw	Multiply Low Word	-	-	DP-MUL	GPR			4	5	5	1	R	-			-	-	-
mullw.	Multiply Low Word and Record	C2	1	DP-MUL	GPR			4	5	5	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
mullwo	Multiply Low Word and Record OV	-	-	DP-MUL	GPR		ov	4	5	5	1	R	-			-	-	-
mullwo.	Multiply Low Word and Record OV and Record	C2	1	DP-MUL	GPR		ov	4	5	5	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-		3	-	D1P	-
nand	NAND	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
nand.	NAND and Record	-	-	ALU	GPR	CR	fxcc	4	2	2	1	-	-			-	-	-
neg	Negate	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
neg.	Negate and Record	-	-	ALU	GPR	CR	fxcc	4	2	2	1	-	-			-	-	-
nego	Negate and Record OV	-	-	ALU	GPR		ov	4	2	2	1	-	-			-	-	-
nego.	Negate and Record OV and Record	-	-	ALU	GPR	CR	fxcc,ov	4	2	2	1	-	-			-	-	-
nor	NOR	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
nor.	NOR and Record	-	-	ALU	GPR	CR	fxcc	4	2	2	1	-	-			-	-	-
or	OR	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
or.	OR and Record	-	-	ALU	GPR	CR	fxcc	4	2	2	1	-	-			-	-	-

Table A-1. Instruction Properties (Sheet 54 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
orc	OR with Complement	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
orc.	OR with Complement and Record	-	-	ALU	GPR	CR	fxcc	4	2	2	1	-	-			-	-	-
ori	OR Immediate	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
oris	OR Immediate Shifted	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
paste	Paste	-	-	LD				4	4	4	1	-	-			-	-	-
paste.	Paste and Record	C2	1	LD			fxcc	4	4	4	1	-	-			-	-	-
			2	ALU		CR		4	2	2	1	-	-		3	-	D1P	-
popcntb	Population Count Byte	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
popcntd	Population Count Dword	-	-	ALU2	GPR			4	3	3	1	-	-			-	-	-
popcntw	Population Count Words	-	-	ALU2	GPR			4	3	3	1	-	-			-	-	-
prtyd	Parity Dword	-	-	ALU2	GPR			4	3	3	1	-	-			-	-	-
prtyw	Parity Word	-	-	ALU2	GPR			4	3	3	1	-	-			-	-	-
rfebb	Return from Event Based Branch	-	-	ALU				4	2	2	1	-	-			-	-	-
rldcl	Rotate Left Dword then Clear Left	-	-	ALU	GPR			4	2	2	1	R	-			-	-	-
rldcl.	Rotate Left Dword then Clear Left and Record	C2	1	ALU	GPR			4	2	2	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
rldcr	Rotate Left Dword then Clear Right	-	-	ALU	GPR			4	2	2	1	R	-			-	-	-
rldcr.	Rotate Left Dword then Clear Right and Record	C2	1	ALU	GPR			4	2	2	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
rldic	Rotate Left Dword Immediate then Clear	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
rldic.	Rotate Left Dword Immediate then Clear and Record	C2	1	ALU	GPR			4	2	2	1	-	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-

Table A-1. Instruction Properties (Sheet 55 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
rldicl	Rotate Left Dword Immediate then Clear Left	-	-	ALU	GPR			4	2	2	1	R	-			-	-	-
rldicl.	Rotate Left Dword Immediate then Clear Left and Record	C2	1	ALU	GPR			4	2	2	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
rldicr	Rotate Left Dword Immediate then Clear Right	-	-	ALU	GPR			4	2	2	1	R	-			-	-	-
rldicr.	Rotate Left Dword Immediate then Clear Right and Record	C2	1	ALU	GPR			4	2	2	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
rldimi	Rotate Left Dword Immediate then Mask Insert	-	-	ALU	GPR			4	2	2	1	R	-			-	-	-
rldimi.	Rotate Left Dword Immediate then Mask Insert and Record	C2	1	ALU	GPR			4	2	2	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
rlwimi	Rotate Left Word Immediate then Mask Insert	-	-	ALU	GPR			4	2	2	1	R	-			-	-	-
rlwimi.	Rotate Left Word Immediate then Mask Insert and Record	C2	1	ALU	GPR			4	2	2	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
rlwinm	Rotate Left Word Immediate then AND with Mask	-	-	ALU	GPR			4	2	2	1	R	-			-	-	-
rlwinm.	Rotate Left Word Immediate then AND with Mask and Record	C2	1	ALU	GPR			4	2	2	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
rlwnm	Rotate Left Word then AND with Mask	-	-	ALU	GPR			4	2	2	1	R	-			-	-	-
rlwnm.	Rotate Left Word then AND with Mask and Record	C2	1	ALU	GPR			4	2	2	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
setb	Set Boolean	-	-	ALU2	GPR			4	3	3	1	-	-		3	-	-	-

Table A-1. Instruction Properties (Sheet 56 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
slbfee.	SLB Find Entry ESID and Record	C3	1	LD				4	4	4	1	P	-			-	-	N
			2	LD	GPR		fxcc	4	4	4	1	P	-			-	-	N
			3	ALU		CR		4	2	2	1	P	-		3	-	D1P	N
slbia.IH000	SLB Invalidate All	C2	1	LD				4	4	4	1	P	WB,WS			-	-	N
			2	LD				4	4	4	1	P	WB,WS			-	-	N
slbia.IH001	SLB Invalidate All	C2	1	LD				4	4	4	1	P	WB,WS			-	-	N
			2	LD				4	4	4	1	P	WB,WS			-	-	N
slbia.IH010	SLB Invalidate All	C2	1	LD				4	4	4	1	P	WB,WS			-	-	N
			2	LD				4	4	4	1	P	WB,WS			-	-	N
slbia.IH011	SLB Invalidate All	C2	1	LD				4	4	4	1	P	WB,WS			-	-	N
			2	LD				4	4	4	1	P	WB,WS			-	-	N
slbia.IH100	SLB Invalidate All	C2	1	LD				4	4	4	1	P	WB,WS			-	-	N
			2	LD				4	4	4	1	P	WB,WS			-	-	N
slbia.IH101	SLB Invalidate All	C2	1	LD				4	4	4	1	P	WB,WS			-	-	N
			2	LD				4	4	4	1	P	WB,WS			-	-	N
slbia.IH110	SLB Invalidate All	C2	1	LD				4	4	4	1	P	WB,WS			-	-	N
			2	LD				4	4	4	1	P	WB,WS			-	-	N
slbia.IH111	SLB Invalidate All	C2	1	LD				4	4	4	1	P	WB,WS			-	-	N
			2	LD				4	4	4	1	P	WB,WS			-	-	N
slbie	SLB Invalidate Entry	C2	1	LD				4	4	4	1	P	-			-	-	N
			2	LD				4	4	4	1	P	-			-	-	N
slbieg	SLB Invalidate Entry Global	-	-	ST				4	-	-	1	R	-			-	-	-

Table A-1. Instruction Properties (Sheet 57 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
slbmfee	SLB Move From Entry ESID	C2	1	LD				4	4	4	1	P	-			-	-	N
			2	LD	GPR			4	4	4	1	P	-			-	-	N
slbmfev	SLB Move From Entry VSID	C2	1	LD				4	4	4	1	P	-			-	-	N
			2	LD	GPR			4	4	4	1	P	-			-	-	N
slbmte	SLB Move to Entry	C2	1	LD				4	4	4	1	P	-			-	-	N
			2	LD				4	4	4	1	P	-			-	-	N
sld	Shift Left Dword	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
sld.	Shift Left Dword and Record	C2	1	ALU	GPR			4	2	2	1	-	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
slw	Shift Left Word	-	-	ALU	GPR			4	2	2	1	R	-			-	-	-
slw.	Shift Left Word and Record	C2	1	ALU	GPR			4	2	2	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
srad	Shift Right Algebraic Dword	-	-	ALU	GPR		caoc	4	2	2	1	-	-			-	-	-
srad.	Shift Right Algebraic Dword and Record	C2	1	ALU	GPR		caoc	4	2	2	1	-	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
sradi	Shift Right Algebraic Dword Immediate	-	-	ALU	GPR		caoc	4	2	2	1	-	-			-	-	-
sradi.	Shift Right Algebraic Dword Immediate and Record	C2	1	ALU	GPR		caoc	4	2	2	1	-	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
sraw	Shift Right Algebraic Word	-	-	ALU	GPR		caoc	4	2	2	1	R	-			-	-	-
sraw.	Shift Right Algebraic Word and Record	C2	1	ALU	GPR		caoc	4	2	2	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
srawi	Shift Right Algebraic Word Immediate	-	-	ALU	GPR		caoc	4	2	2	1	R	-			-	-	-

Table A-1. Instruction Properties (Sheet 58 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
srawi.	Shift Right Algebraic Word Immediate and Record	C2	1	ALU	GPR		caoc	4	2	2	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
srd	Shift Right Dword	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
srd.	Shift Right Dword and Record	C2	1	ALU	GPR			4	2	2	1	-	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
srw	Shift Right Word	-	-	ALU	GPR			4	2	2	1	R	-			-	-	-
srw.	Shift Right Word and Record	C2	1	ALU	GPR			4	2	2	1	R	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	D1P	-
stb	Store Byte	-	-	ST				4	-	-	1	R-st	-			-	-	-
stbcix	Store Byte Caching Inhibited Indexed	-	-	ST				4	-	-	1	R	-			-	-	-
stbcx.	Store Byte Conditional Indexed and Record	C2	1	ST			fxcc	4	-	-	1	R	-			-	-	-
			2	ALU		CR		4	2	2	1	-	-		3	-	D1P	-
stbu	Store Byte with Update	C2	1	ST				4	-	-	1	R-st	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	-	-
stbux	Store Byte with Update Indexed	C2	1	ST				4	-	-	1	R	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	-	-
stbx	Store Byte Indexed	-	-	ST				4	-	-	1	R	-			-	-	-
std	Store Dword	-	-	ST				4	-	-	1	R-st	-			-	-	-
stdat	Store Dword Atomic	X	1	LD				4	4	4	1	-	-			-	-	-
			2	ST				4	-	-	1	R	-			-	-	N
			3	LD				4	4	4	1	-	-			-	-	-
stdbrx	Store Dword Byte-Reverse Indexed	-	-	ST				4	-	-	1	R	-			-	-	-

Table A-1. Instruction Properties (Sheet 59 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
stdcix	Store Dword Caching Inhibited Indexed	-	-	ST				4	-	-	1	R	-			-	-	-
stdcx.	Store Dword Conditional Indexed and Record	C2	1	ST			fxcc	4	-	-	1	R	-			-	-	-
			2	ALU		CR		4	2	2	1	-	-		3	-	D1P	-
stdu	Store Dword with Update	C2	1	ST				4	-	-	1	R-st	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	-	-
stdux	Store Dword with Update Indexed	C2	1	ST				4	-	-	1	R	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	-	-
stdx	Store Dword Indexed	-	-	ST				4	-	-	1	R	-			-	-	-
stfd	Store Floating Double	-	-	ST				4	-	-	1	R	-			-	-	-
stfdp	Store Floating Double Pair	C2	1	ST				4	-	-	1	R	-			-	-	-
			2	ST				4	-	-	1	R	-			-	-	-
stfdpx	Store Floating Double Pair Indexed	C2	1	ST				4	-	-	1	R	-			-	-	-
			2	ST				4	-	-	1	R	-			-	-	-
stfdu	Store Floating Double with Update	C2	1	ST				4	-	-	1	R	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	-	-
stfdux	Store Floating Double with Update Indexed	C2	1	ST				4	-	-	1	R	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	-	-
stfdx	Store Floating Double Indexed	-	-	ST				4	-	-	1	R	-			-	-	-
stfiwx	Store Floating as Integer Word Indexed	-	-	ST				4	-	-	1	R	-			-	-	-
stfs	Store Floating Single	-	-	ST				4	-	-	1	R	-			-	-	-
stfsu	Store Floating Single with Update	C2	1	ST				4	-	-	1	R	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	-	-

Table A-1. Instruction Properties (Sheet 60 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
stfsux	Store Floating Single with Update Indexed	C2	1	ST				4	-	-	1	R	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	-	-
stfsx	Store Floating Single Indexed	-	-	ST				4	-	-	1	R	-			-	-	-
sth	Store Hword	-	-	ST				4	-	-	1	R-st	-			-	-	-
sthbrx	Store Hword Byte-Reverse Indexed	-	-	ST				4	-	-	1	R	-			-	-	-
sthcix	Store Hword Caching Inhibited Indexed	-	-	ST				4	-	-	1	R	-			-	-	-
sthcx.	Store Hword Conditional Indexed and Record	C2	1	ST			fxcc	4	-	-	1	R	-			-	-	-
			2	ALU		CR		4	2	2	1	-	-		3	-	D1P	-
sthu	Store Hword with Update	C2	1	ST				4	-	-	1	R-st	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	-	-
sthux	Store Hword with Update Indexed	C2	1	ST				4	-	-	1	R	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	-	-
sthx	Store Hword Indexed	-	-	ST				4	-	-	1	R	-			-	-	-
stmw	Store Multiple Word	X	1u	ST				4	-	-	1	R	-			-	-	-
stop	Stop	-	-	0				-	-	-	1	-	-			-	-	-
stq	Store Qword	C2	1	ST				4	-	-	1	R-st	-			-	-	-
			2	ST				4	-	-	1	R-st	-			-	-	-
stqcx.	Store Qword Conditional Indexed and Record	C3	1	ST				4	-	-	1	R	-		3	-	-	-
			2	ST			fxcc	4	-	-	1	R	-		3	-	-	-
			3	ALU		CR		4	2	2	1	-	-		3	-	D1P	-
stswi	Store String Word Immediate	X	1u	ST				4	-	-	1	R-st	-			-	-	-

Table A-1. Instruction Properties (Sheet 61 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
stswx	Store String Word Indexed	X	1	DIV				2	12	12	1	E	-		3	-	-	-
			2u	ST2				4	-	-	1	R	-			-	-	-
stvebx	Store Vector Element Byte Indexed	-	-	ST				4	-	-	1	V	-	1		S	-	-
stvehx	Store Vector Element Hword Indexed	-	-	ST				4	-	-	1	V	-	1		S	-	-
stvewx	Store Vector Element Word Indexed	-	-	ST				4	-	-	1	V	-	1		S	-	-
stvx	Store Vector Indexed	-	-	ST				4	-	-	1	V	-	1		S	-	-
stvxl	Store Vector Indexed Last	-	-	ST				4	-	-	1	V	-	1		S	-	-
stw	Store Word	-	-	ST				4	-	-	1	R-st	-			-	-	-
stwat	Store Word ATomic	X	1	LD				4	4	4	1	-	-			-	-	-
			2	ST				4	-	-	1	R	-			-	-	N
			3	LD				4	4	4	1	-	-			-	-	-
stwbrx	Store Word Byte-Reverse Indexed	-	-	ST				4	-	-	1	R	-			-	-	-
stwcix	Store Word Caching Inhibited Indexed	-	-	ST				4	-	-	1	R	-			-	-	-
stwcx.	Store Word Conditional Indexed and Record	C2	1	ST			fxcc	4	-	-	1	R	-			-	-	-
			2	ALU		CR		4	2	2	1	-	-		3	-	D1P	-
stwu	Store Word with Update	C2	1	ST				4	-	-	1	R-st	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	-	-
stwux	Store Word with Update Indexed	C2	1	ST				4	-	-	1	R	-			-	-	-
			2	ALU	GPR			4	2	2	1	-	-			-	-	-
stwx	Store Word Indexed	-	-	ST				4	-	-	1	R	-			-	-	-
stxsd	Store VSX Scalar Dword	-	-	ST				4	-	-	1	R-st	-			-	-	-

Table A-1. Instruction Properties (Sheet 62 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
stxsdx	Store VSX Scalar Dword Indexed	-	-	ST				4	-	-	1	R	-			-	-	-
stxsibx	Store VSX Scalar as Integer Byte Indexed	-	-	ST				4	-	-	1	R	-			-	-	-
stxsihx	Store VSX Scalar as Integer Hword Indexed	-	-	ST				4	-	-	1	R	-			-	-	-
stxsiwx	Store VSX Scalar as Integer Word Indexed	-	-	ST				4	-	-	1	R	-			-	-	-
stxssp	Store VSX Scalar SP	-	-	ST				4	-	-	1	R-st	-			-	-	-
stxsspx	Store VSX Scalar SP Indexed	-	-	ST				4	-	-	1	R	-			-	-	-
stxv	Store VSX Vector	-	-	ST				4	-	-	1	V	-	1		S	-	-
stxvb16x	Store VSX Vector Byte*16 Indexed	-	-	ST				4	-	-	1	V	-	1		S	-	-
stxvd2x	Store VSX Vector Dword*2 Indexed	-	-	ST				4	-	-	1	V	-	1		S	-	-
stxvh8x	Store VSX Vector Hword*8 Indexed	-	-	ST				4	-	-	1	V	-	1		S	-	-
stxvl	Store VSX Vector with Length	-	-	ST2				4	-	-	1	V	-	1		S	-	-
stxvll	Store VSX Vector Left-justified with Length	-	-	ST2				4	-	-	1	V	-	1		S	-	-
stxvw4x	Store VSX Vector Word*4 Indexed	-	-	ST				4	-	-	1	V	-	1		S	-	-
stxvx	Store VSX Vector Indexed	-	-	ST				4	-	-	1	V	-	1		S	-	-
subf	Subtract From	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
subf.	Subtract From and Record	-	-	ALU	GPR	CR	fxcc	4	2	2	1	-	-			-	-	-
subfc	Subtract From Carrying	-	-	ALU	GPR		caoc	4	2	2	1	-	-			-	-	-
subfc.	Subtract From Carrying and Record	-	-	ALU	GPR	CR	caoc,fxcc	4	2	2	1	-	-			-	-	-

Table A-1. Instruction Properties (Sheet 63 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
subfco	Subtract From Carrying and Record OV	-	-	ALU	GPR		caoc,ov	4	2	2	1	-	-			-	-	-
subfco.	Subtract From Carrying and Record OV and Record	-	-	ALU	GPR	CR	caoc,fxcc,ov	4	2	2	1	-	-			-	-	-
subfe	Subtract From Extended	-	-	ALU	GPR		caoc	4	2	2	1	-	-		3	-	-	-
subfe.	Subtract From Extended and Record	-	-	ALU	GPR	CR	caoc,fxcc	4	2	2	1	-	-		3	-	-	-
subfeo	Subtract From Extended and Record OV	-	-	ALU	GPR		caoc,ov	4	2	2	1	-	-		3	-	-	-
subfeo.	Subtract From Extended and Record OV and Record	-	-	ALU	GPR	CR	caoc,fxcc,ov	4	2	2	1	-	-		3	-	-	-
subfic	Subtract From Immediate Carrying	-	-	ALU	GPR		caoc	4	2	2	1	-	-			-	-	-
subfme	Subtract From Minus One Extended	-	-	ALU	GPR		caoc	4	2	2	1	-	-		3	-	-	-
subfme.	Subtract From Minus One Extended and Record	-	-	ALU	GPR	CR	caoc,fxcc	4	2	2	1	-	-		3	-	-	-
subfmeo	Subtract From Minus One Extended and Record OV	-	-	ALU	GPR		caoc,ov	4	2	2	1	-	-		3	-	-	-
subfmeo.	Subtract From Minus One Extended and Record OV and Record	-	-	ALU	GPR	CR	caoc,fxcc,ov	4	2	2	1	-	-		3	-	-	-
subfo	Subtract From and Record OV	-	-	ALU	GPR		ov	4	2	2	1	-	-			-	-	-
subfo.	Subtract From and Record OV and Record	-	-	ALU	GPR	CR	fxcc,ov	4	2	2	1	-	-			-	-	-
subfze	Subtract From Zero Extended	-	-	ALU	GPR		caoc	4	2	2	1	-	-		3	-	-	-
subfze.	Subtract From Zero Extended and Record	-	-	ALU	GPR	CR	caoc,fxcc	4	2	2	1	-	-		3	-	-	-
subfzeo	Subtract From Zero Extended and Record OV	-	-	ALU	GPR		caoc,ov	4	2	2	1	-	-		3	-	-	-

Table A-1. Instruction Properties (Sheet 64 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
subfzeo.	Subtract From Zero Extended and Record OV and Record	-	-	ALU	GPR	CR	caoc,fxcc,ov	4	2	2	1	-	-		3	-	-	-
sync	Synchronize	-	-	LD				4	4	4	1	-	-			-	-	-
tabort.	Transaction Abort and Record	-	-	ALU		CR	fxcc	4	2	2	1	R	-			-	-	-
tabortdc.	Transaction Abort Dword Conditional and Record	-	-	ALU2		CR	fxcc	4	3	3	1	R	-			-	-	-
tabortdci.	Transaction Abort Dword Conditional Immediate and Record	-	-	ALU2		CR	fxcc	4	3	3	1	R	-			-	-	-
tabortwc.	Transaction Abort Word Conditional and Record	-	-	ALU2		CR	fxcc	4	3	3	1	R	-			-	-	-
tabortwci.	Transaction Abort Word Conditional Immediate and Record	-	-	ALU2		CR	fxcc	4	3	3	1	R	-			-	-	-
tbegin.	Transaction Begin and Record	C2	1	NOP				6	0	0	1	-	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	-	-
tcheck	Transaction Check and Record	C2	1	LD			fxcc	4	4	4	1	-	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-		3	-	D1P	-
td_ti	Trap Dword	-	-	ALU2				4	3	3	1	R	-			-	-	-
tdi_ti	Trap Dword Immediate	-	-	ALU2				4	3	3	1	R	-			-	-	-
tend.	Transaction End and Record	C2	1	LD				4	4	4	1	-	-			-	-	-
			2	ALU		CR	fxcc	4	2	2	1	-	-			-	-	-
tlbie	TLB Invalidate Entry	-	-	ST				4	-	-	1	R	-			-	-	-
tlbie_h	TLB Invalidate Entry	-	-	ST				4	-	-	1	R	-			-	-	-
tlbiel	TLB Invalidate Entry Local	C2	1	LD				4	4	4	1	P	-			-	-	N
			2	LD				4	4	4	1	P	-			-	-	N

Table A-1. Instruction Properties (Sheet 65 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
tlbiel_h	TLB Invalidate Entry Local	C2	1	LD				4	4	4	1	P	-			-	-	N
			2	LD				4	4	4	1	P	-			-	-	N
tlbsync	TLB Synchronize	-	-	LD				4	4	4	1	-	-			-	-	-
trechkpt.	Transaction Recheckpoint and Record	-	-	ALU		CR	fxcc	4	2	2	1	-	-			-	-	N
treclaim.	Transaction Reclaim and Record	-	-	ALU		CR	fxcc	4	2	2	1	R	-			-	-	-
tsr.	Transaction Suspend or Resume and Record	-	-	ALU		CR	fxcc	4	2	2	1	R	-			-	-	-
tw	Trap Word	-	-	ALU2				4	3	3	1	R	-			-	-	-
twi	Trap Word Immediate	-	-	ALU2				4	3	3	1	R	-			-	-	-
vabsdub	Vector Absolute Difference Unsigned Byte	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vabsduh	Vector Absolute Difference Unsigned Hword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vabsduw	Vector Absolute Difference Unsigned Word	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vaddcuq	Vector Add and write Carry Unsigned Qword	-	-	DX	VR			2	3	3	1	V	-	1		S	-	-
vaddcuw	Vector Add and Write Carry-Out Unsigned Word	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vaddecuq	Vector Add Extended and write Carry Unsigned Qword	-	-	DX	VR			2	3	3	1	V	-	1		S	-	-
vaddeuqm	Vector Add Extended Unsigned Qword Modulo	-	-	DX	VR			2	3	3	1	V	-	1		S	-	-
vaddfp	Vector Add Floating-Point	-	-	DP	VR			2	5	7	1	V	-	1	3	S	-	-
vaddsbs	Vector Add Signed Byte Saturate	-	-	ALU2	VR		sat	2	3	3	1	V	-	1	3	S	-	-
vaddshs	Vector Add Signed Hword Saturate	-	-	ALU2	VR		sat	2	3	3	1	V	-	1	3	S	-	-

Table A-1. Instruction Properties (Sheet 66 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
vaddsws	Vector Add Signed Word Saturate	-	-	ALU2	VR		sat	2	3	3	1	V	-	1	3	S	-	-
vaddubm	Vector Add Unsigned Byte Modulo	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vaddubs	Vector Add Unsigned Byte Saturate	-	-	ALU2	VR		sat	2	3	3	1	V	-	1	3	S	-	-
vaddudm	Vector Add Unsigned Dword Modulo	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vadduhm	Vector Add Unsigned Hword Modulo	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vadduhs	Vector Add Unsigned Hword Saturate	-	-	ALU2	VR		sat	2	3	3	1	V	-	1	3	S	-	-
vadduqm	Vector Add Unsigned Qword Modulo	-	-	DX	VR			2	3	3	1	V	-	1		S	-	-
vadduwm	Vector Add Unsigned Word Modulo	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vadduws	Vector Add Unsigned Word Saturate	-	-	ALU2	VR		sat	2	3	3	1	V	-	1	3	S	-	-
vand	Vector Logical AND	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vandc	Vector Logical AND with Complement	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vavgsb	Vector Average Signed Byte	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vavgsh	Vector Average Signed Hword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vavgsw	Vector Average Signed Word	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vavgub	Vector Average Unsigned Byte	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vavguh	Vector Average Unsigned Hword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vavguw	Vector Average Unsigned Word	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vbpermd	Vector Bit Permute Dword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vbpermq	Vector Bit Permute Qword	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-

Table A-1. Instruction Properties (Sheet 67 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
vcfsx	Vector Convert From Signed Word	-	-	DP	VR			2	5	7	1	V	-	1		S	-	-
vcfux	Vector Convert From Unsigned Word	-	-	DP	VR			2	5	7	1	V	-	1		S	-	-
vcipher	Vector AES Cipher	-	-	CY	VR			1	6	6	1	V	-	1		S	-	-
vcipherlast	Vector AES Cipher Last	-	-	CY	VR			1	6	6	1	V	-	1		S	-	-
vclzb	Vector Count Leading Zeros Byte	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vclzd	Vector Count Leading Zeros Dword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vclzh	Vector Count Leading Zeros Hword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vclzlsbb	Vector Count Leading Zero Least-Significant Bits Byte	-	-	PM	GPR			4	3	3	1	V	-	1		S	-	-
vclzw	Vector Count Leading Zeros Word	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vcmpbfp	Vector Compare Bounds Floating-Point	-	-	ALU2	VR			2	3	3	1	V	-	1	3	S	-	-
vcmpbfp.	Vector Compare Bounds Floating-Point and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1	3	S	-	-
vcmpeqfp	Vector Compare Equal to Floating-Point	-	-	ALU2	VR			2	3	3	1	V	-	1	3	S	-	-
vcmpeqfp.	Vector Compare Equal to Floating-Point and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1	3	S	-	-
vcmpequb	Vector Compare Equal Unsigned Byte	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vcmpequb.	Vector Compare Equal Unsigned Byte and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1		S	-	-
vcmpequd	Vector Compare Equal Unsigned Dword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vcmpequd.	Vector Compare Equal Unsigned Dword and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1		S	-	-

Table A-1. Instruction Properties (Sheet 68 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
vcmpequh	Vector Compare Equal Unsigned Hword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vcmpequh.	Vector Compare Equal Unsigned Hword and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1		S	-	-
vcmpequw	Vector Compare Equal Unsigned Word	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vcmpequw.	Vector Compare Equal Unsigned Word and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1		S	-	-
vcmpgefp	Vector Compare Greater Than or Equal to Floating-Point	-	-	ALU2	VR			2	3	3	1	V	-	1	3	S	-	-
vcmpgefp.	Vector Compare Greater Than or Equal to Floating-Point and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1	3	S	-	-
vcmpgtfp	Vector Compare Greater Than Floating-Point	-	-	ALU2	VR			2	3	3	1	V	-	1	3	S	-	-
vcmpgtfp.	Vector Compare Greater Than Floating-Point and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1	3	S	-	-
vcmpgtsb	Vector Compare Greater Than Signed Byte	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vcmpgtsb.	Vector Compare Greater Than Signed Byte and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1		S	-	-
vcmpgtsd	Vector Compare Greater Than Signed Dword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vcmpgtsd.	Vector Compare Greater Than Signed Dword and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1		S	-	-
vcmpgtsh	Vector Compare Greater Than Signed Hword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vcmpgtsh.	Vector Compare Greater Than Signed Hword and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1		S	-	-
vcmpgtsw	Vector Compare Greater Than Signed Word	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-

Table A-1. Instruction Properties (Sheet 69 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
vcmpgtsw.	Vector Compare Greater Than Signed Word and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1		S	-	-
vcmpgtub	Vector Compare Greater Than Unsigned Byte	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vcmpgtub.	Vector Compare Greater Than Unsigned Byte and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1		S	-	-
vcmpgtud	Vector Compare Greater Than Unsigned Dword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vcmpgtud.	Vector Compare Greater Than Unsigned Dword and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1		S	-	-
vcmpgtuh	Vector Compare Greater Than Unsigned Hword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vcmpgtuh.	Vector Compare Greater Than Unsigned Hword and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1		S	-	-
vcmpgtuw	Vector Compare Greater Than Unsigned Word	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vcmpgtuw.	Vector Compare Greater Than Unsigned Word and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1		S	-	-
vcmpneb	Vector Compare Not Equal Byte	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vcmpneb.	Vector Compare Not Equal Byte and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1		S	-	-
vcmpneh	Vector Compare Not Equal Hword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vcmpneh.	Vector Compare Not Equal Hword and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1		S	-	-
vcmpnew	Vector Compare Not Equal Word	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vcmpnew.	Vector Compare Not Equal Word and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1		S	-	-
vcmpnezb	Vector Compare Not Equal or Zero Byte	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-

Table A-1. Instruction Properties (Sheet 70 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
vcmpnezb.	Vector Compare Not Equal or Zero Byte and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1		S	-	-
vcmpnezh	Vector Compare Not Equal or Zero Hword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vcmpnezh.	Vector Compare Not Equal or Zero Hword and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1		S	-	-
vcmpnezw	Vector Compare Not Equal or Zero Word	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vcmpnezw.	Vector Compare Not Equal or Zero Word and Record	-	-	ALU2	VR	CR		2	3	3	1	V	-	1		S	-	-
vctsxs	Vector Convert to Signed Word Saturate	-	-	DP	VR		sat	2	5	7	1	V	-	1	3	S	-	-
vctuxs	Vector Convert to Unsigned Word Saturate	-	-	DP	VR		sat	2	5	7	1	V	-	1	3	S	-	-
vctzb	Vector Count Trailing Zeros Byte	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vctzd	Vector Count Trailing Zeros Dword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vctzh	Vector Count Trailing Zeros Hword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vctzlsbb	Vector Count Trailing Zero Least-Significant Bits Byte	-	-	PM	GPR			4	3	3	1	V	-	1		S	-	-
vctzw	Vector Count Trailing Zeros Word	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
veqv	Vector Logical Equivalence	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vexptefp	Vector 2 Raised to the Exponent Estimate Floating-Point	-	-	DP	VR			2	5	7	1	V	-	1	3	S	-	-
vextractd	Vector Extract Dword	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vextractub	Vector Extract Unsigned Byte	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vextractuh	Vector Extract Unsigned Hword	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vextractuw	Vector Extract Unsigned Word	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-

Table A-1. Instruction Properties (Sheet 71 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
vextsb2d	Vector Extend Sign Byte to Dword	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vextsb2w	Vector Extend Sign Byte to Word	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vextsh2d	Vector Extend Sign Hword to Dword	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vextsh2w	Vector Extend Sign Hword to Word	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vextsw2d	Vector Extend Sign Word to Dword	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vextublx	Vector Extract Unsigned Byte Left-Indexed	-	-	PM	GPR			4	3	3	1	V	-	1		S	-	-
vextubrx	Vector Extract Unsigned Byte Right-Indexed	-	-	PM	GPR			4	3	3	1	V	-	1		S	-	-
vextuhlx	Vector Extract Unsigned Hword Left-Indexed	-	-	PM	GPR			4	3	3	1	V	-	1		S	-	-
vextuhrx	Vector Extract Unsigned Hword Right-Indexed	-	-	PM	GPR			4	3	3	1	V	-	1		S	-	-
vextuwlx	Vector Extract Unsigned Word Left-Indexed	-	-	PM	GPR			4	3	3	1	V	-	1		S	-	-
vextuwrx	Vector Extract Unsigned Word Right-Indexed	-	-	PM	GPR			4	3	3	1	V	-	1		S	-	-
vgbbd	Vector Gather Bits by Byte by Dword	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vinsertb	Vector Insert Byte	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vinsertd	Vector Insert Dword	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vinserth	Vector Insert Hword	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vinsertw	Vector Insert Word	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vlogefp	Vector Log Base 2 Estimate Floating-Point	-	-	DP	VR			2	5	7	1	V	-	1	3	S	-	-
vmaddfp	Vector Multiply-Add Floating-Point	-	-	DP	VR			2	5	7	1	V	-	1	3	S	-	-

Table A-1. Instruction Properties (Sheet 72 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
vmaxfp	Vector Maximum Floating-Point	-	-	ALU2	VR			2	3	3	1	V	-	1	3	S	-	-
vmaxsb	Vector Maximum Signed Byte	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vmaxsd	Vector Maximum Signed Dword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vmaxsh	Vector Maximum Signed Hword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vmaxsw	Vector Maximum Signed Word	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vmaxub	Vector Maximum Unsigned Byte	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vmaxud	Vector Maximum Unsigned Dword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vmaxuh	Vector Maximum Unsigned Hword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vmaxuw	Vector Maximum Unsigned Word	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vmhaddshs	Vector Multiply-High-Add Signed Hword Saturate	-	-	DP	VR		sat	2	5	7	1	V	-	1	3	S	-	-
vmhraddshs	Vector Multiply-High-Round-Add Signed Hword Saturate	-	-	DP-XC	VR		sat	2	7	7	1	V	-	1	3	S	-	-
vminfp	Vector Minimum Floating-Point	-	-	ALU2	VR			2	3	3	1	V	-	1	3	S	-	-
vminsb	Vector Minimum Signed Byte	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vminsd	Vector Minimum Signed Dword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vminsh	Vector Minimum Signed Hword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vminsw	Vector Minimum Signed Word	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vminub	Vector Minimum Unsigned Byte	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vminud	Vector Minimum Unsigned Dword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vminuh	Vector Minimum Unsigned Hword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vminuw	Vector Minimum Unsigned Word	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vmladduhm	Vector Multiply-Low-Add Unsigned Hword Modulo	-	-	DP-XC	VR			2	7	7	1	V	-	1		S	-	-

Table A-1. Instruction Properties (Sheet 73 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
vmrgew	Vector Merge Even Word	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vmrghb	Vector Merge High Byte	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vmrghh	Vector Merge High Hword	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vmrghw	Vector Merge High Word	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vmrglb	Vector Merge Low Byte	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vmrglh	Vector Merge Low Hword	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vmrglw	Vector Merge Low Word	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vmrgow	Vector Merge Odd Word	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vmsummbm	Vector Multiply-Sum Mixed Byte Modulo	-	-	DP-XC	VR			2	7	7	1	V	-	1		S	-	-
vmsumshm	Vector Multiply-Sum Signed Hword Modulo	-	-	DP-XC	VR			2	7	7	1	V	-	1		S	-	-
vmsumshs	Vector Multiply-Sum Signed Hword Saturate	-	-	DP-XC	VR		sat	2	7	7	1	V	-	1	3	S	-	-
vmsumubm	Vector Multiply-Sum Unsigned Byte Modulo	-	-	DP-XC	VR			2	7	7	1	V	-	1		S	-	-
vmsumudm	Vector Multiply-Sum Unsigned Doubleword Modulo	-	-	DP	VR			2	5	7	1	V	-	1		S	-	-
vmsumuhm	Vector Multiply-Sum Unsigned Hword Modulo	-	-	DP-XC	VR			2	7	7	1	V	-	1		S	-	-
vmsumuhs	Vector Multiply-Sum Unsigned Hword Saturate	-	-	DP-XC	VR		sat	2	7	7	1	V	-	1	3	S	-	-
vmul10cuq	Vector Multiply-by-10 and write Carry Unsigned Qword	-	-	DX	VR			2	3	3	1	V	-	1		S	-	-
vmul10ecuq	Vector Multiply-by-10 Extended and write Carry Unsigned Qword	-	-	DX	VR			2	3	3	1	V	-	1		S	-	-
vmul10euq	Vector Multiply-by-10 Extended Unsigned Qword	-	-	DX	VR			2	3	3	1	V	-	1		S	-	-

Table A-1. Instruction Properties (Sheet 74 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
vmul10uq	Vector Multiply-by-10 Unsigned Qword	-	-	DX	VR			2	3	3	1	V	-	1		S	-	-
vmulesb	Vector Multiply Even Signed Byte	-	-	DP-XC	VR			2	7	7	1	V	-	1		S	-	-
vmulesh	Vector Multiply Even Signed Hword	-	-	DP-XC	VR			2	7	7	1	V	-	1		S	-	-
vmulesw	Vector Multiply Even Signed Word	-	-	DP-XC	VR			2	7	7	1	V	-	1		S	-	-
vmuleub	Vector Multiply Even Unsigned Byte	-	-	DP-XC	VR			2	7	7	1	V	-	1		S	-	-
vmuleuh	Vector Multiply Even Unsigned Hword	-	-	DP-XC	VR			2	7	7	1	V	-	1		S	-	-
vmuleuw	Vector Multiply Even Unsigned Word	-	-	DP-XC	VR			2	7	7	1	V	-	1		S	-	-
vmulosb	Vector Multiply Odd Signed Byte	-	-	DP-XC	VR			2	7	7	1	V	-	1		S	-	-
vmulosh	Vector Multiply Odd Signed Hword	-	-	DP-XC	VR			2	7	7	1	V	-	1		S	-	-
vmulosw	Vector Multiply Odd Signed Word	-	-	DP-XC	VR			2	7	7	1	V	-	1		S	-	-
vmuloub	Vector Multiply Odd Unsigned Byte	-	-	DP-XC	VR			2	7	7	1	V	-	1		S	-	-
vmulouh	Vector Multiply Odd Unsigned Hword	-	-	DP-XC	VR			2	7	7	1	V	-	1		S	-	-
vmulouw	Vector Multiply Odd Unsigned Word	-	-	DP-XC	VR			2	7	7	1	V	-	1		S	-	-
vmuluwm	Vector Multiply Unsigned Word Modulo	-	-	DP-XC	VR			2	7	7	1	V	-	1		S	-	-
vnand	Vector Logical NAND	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vncipher	Vector AES Inverse Cipher	-	-	CY	VR			1	6	6	1	V	-	1		S	-	-
vncipherlast	Vector AES Inverse Cipher Last	-	-	CY	VR			1	6	6	1	V	-	1		S	-	-
vnegd	Vector Negate Dword	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-

Table A-1. Instruction Properties (Sheet 75 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
vnegw	Vector Negate Word	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vnmsubfp	Vector Negative Multiply-Subtract Floating-Point	-	-	DP	VR			2	5	7	1	V	-	1	3	S	-	-
vnor	Vector Logical NOR	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vor	Vector Logical OR	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vorc	Vector Logical OR with Complement	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vperm	Vector Permute	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vpermr	Vector Permute Right-indexed	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vpermxor	Vector Permute and Exclusive-OR	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vpkpx	Vector Pack Pixel	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vpksdss	Vector Pack Signed Dword Signed Saturate	-	-	PM	VR		sat	2	3	3	1	V	-	1	3	S	-	-
vpksdus	Vector Pack Signed Dword Unsigned Saturate	-	-	PM	VR		sat	2	3	3	1	V	-	1	3	S	-	-
vpkshss	Vector Pack Signed Hword Signed Saturate	-	-	PM	VR		sat	2	3	3	1	V	-	1	3	S	-	-
vpkshus	Vector Pack Signed Hword Unsigned Saturate	-	-	PM	VR		sat	2	3	3	1	V	-	1	3	S	-	-
vpkswss	Vector Pack Signed Word Signed Saturate	-	-	PM	VR		sat	2	3	3	1	V	-	1	3	S	-	-
vpkswus	Vector Pack Signed Word Unsigned Saturate	-	-	PM	VR		sat	2	3	3	1	V	-	1	3	S	-	-
vpkudum	Vector Pack Unsigned Dword Unsigned Modulo	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vpkudus	Vector Pack Unsigned Dword Unsigned Saturate	-	-	PM	VR		sat	2	3	3	1	V	-	1	3	S	-	-
vpkuhum	Vector Pack Unsigned Hword Unsigned Modulo	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-

Table A-1. Instruction Properties (Sheet 76 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
vpkuhus	Vector Pack Unsigned Hword Unsigned Saturate	-	-	PM	VR		sat	2	3	3	1	V	-	1	3	S	-	-
vpkuwum	Vector Pack Unsigned Word Unsigned Modulo	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vpkuwus	Vector Pack Unsigned Word Unsigned Saturate	-	-	PM	VR		sat	2	3	3	1	V	-	1	3	S	-	-
vpmsumb	Vector Polynomial Multiply-Sum Byte	-	-	CY	VR			1	6	6	1	V	-	1		S	-	-
vpmsumd	Vector Polynomial Multiply-Sum Dword	-	-	CY	VR			1	6	6	1	V	-	1		S	-	-
vpmsumh	Vector Polynomial Multiply-Sum Hword	-	-	CY	VR			1	6	6	1	V	-	1		S	-	-
vpmsumw	Vector Polynomial Multiply-Sum Word	-	-	CY	VR			1	6	6	1	V	-	1		S	-	-
vpopcntb	Vector Population Count Byte	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vpopcntd	Vector Population Count Dword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vpopcnth	Vector Population Count Hword	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vpopcntw	Vector Population Count Word	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vprtybd	Vector Parity Byte Dword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vprtybq	Vector Parity Byte Qword	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vprtybw	Vector Parity Byte Word	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vrefp	Vector Reciprocal Estimate Floating-Point	-	-	DP	VR			2	5	7	1	V	-	1	3	S	-	-
vrfim	Vector Round to Floating-Point Integral toward -Infinity	-	-	DP	VR			2	5	7	1	V	-	1	3	S	-	-
vrfin	Vector Round to Floating-Point Integral Nearest	-	-	DP	VR			2	5	7	1	V	-	1	3	S	-	-
vrfip	Vector Round to Floating-Point Integral toward +Infinity	-	-	DP	VR			2	5	7	1	V	-	1	3	S	-	-

Table A-1. Instruction Properties (Sheet 77 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
vrfiz	Vector Round to Floating-Point Integral toward Zero	-	-	DP	VR			2	5	7	1	V	-	1	3	S	-	-
vrlb	Vector Rotate Left Byte	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vrld	Vector Rotate Left Dword	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vrldmi	Vector Rotate Left Dword then Mask Insert	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vrldnm	Vector Rotate Left Dword then AND with Mask	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vrlh	Vector Rotate Left Hword	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vrlw	Vector Rotate Left Word	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vrlwmi	Vector Rotate Left Word then Mask Insert	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vrlwnm	Vector Rotate Left Word then AND with Mask	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vrsqrtefp	Vector Reciprocal Square Root Estimate Floating-Point	-	-	DP	VR			2	5	7	1	V	-	1	3	S	-	-
vsbox	Vector AES SubBytes	-	-	CY	VR			1	6	6	1	V	-	1		S	-	-
vsel	Vector Select	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vshasigmad	Vector SHA-512 Sigma Dword	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vshasigmaw	Vector SHA-256 Sigma Word	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vsl	Vector Shift Left	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vslb	Vector Shift Left Byte	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vsld	Vector Shift Left Dword	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vsldoi	Vector Shift Left Double by Octet Immediate	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vslh	Vector Shift Left Hword	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-

Table A-1. Instruction Properties (Sheet 78 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
vslo	Vector Shift Left by Octet	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vslv	Vector Shift Left Variable	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vslw	Vector Shift Left Word	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vspltb	Vector Splat Byte	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vsplth	Vector Splat Hword	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vspltisb	Vector Splat Immediate Signed Byte	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vspltish	Vector Splat Immediate Signed Hword	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vspltisw	Vector Splat Immediate Signed Word	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vspltw	Vector Splat Word	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vsr	Vector Shift Right	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vsrab	Vector Shift Right Algebraic Byte	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vsrad	Vector Shift Right Algebraic Dword	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vsrah	Vector Shift Right Algebraic Hword	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vsraw	Vector Shift Right Algebraic Word	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vsrb	Vector Shift Right Byte	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vsrd	Vector Shift Right Dword	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vsrh	Vector Shift Right Hword	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vsro	Vector Shift Right by Octet	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vsrv	Vector Shift Right Variable	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vsrw	Vector Shift Right Word	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-

Table A-1. Instruction Properties (Sheet 79 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
vsubcuq	Vector Subtract and write Carry Unsigned Qword	-	-	DX	VR			2	3	3	1	V	-	1		S	-	-
vsubcuw	Vector Subtract and Write Carry-Out Unsigned Word	-	-	ALU2	VR			2	3	3	1	V	-	1		S	-	-
vsubecuq	Vector Subtract Extended and write Carry Unsigned Qword	-	-	DX	VR			2	3	3	1	V	-	1		S	-	-
vsubeuqm	Vector Subtract Extended Unsigned Qword Modulo	-	-	DX	VR			2	3	3	1	V	-	1		S	-	-
vsubfp	Vector Subtract Floating-Point	-	-	DP	VR			2	5	7	1	V	-	1	3	S	-	-
vsubsbs	Vector Subtract Signed Byte Saturate	-	-	ALU2	VR		sat	2	3	3	1	V	-	1	3	S	-	-
vsubshs	Vector Subtract Signed Hword Saturate	-	-	ALU2	VR		sat	2	3	3	1	V	-	1	3	S	-	-
vsubsws	Vector Subtract Signed Word Saturate	-	-	ALU2	VR		sat	2	3	3	1	V	-	1	3	S	-	-
vsububm	Vector Subtract Unsigned Byte Modulo	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vsububs	Vector Subtract Unsigned Byte Saturate	-	-	ALU2	VR		sat	2	3	3	1	V	-	1	3	S	-	-
vsubudm	Vector Subtract Unsigned Dword Modulo	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vsubuhm	Vector Subtract Unsigned Hword Modulo	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vsubuhs	Vector Subtract Unsigned Hword Saturate	-	-	ALU2	VR		sat	2	3	3	1	V	-	1	3	S	-	-
vsubuqm	Vector Subtract Unsigned Qword Modulo	-	-	DX	VR			2	3	3	1	V	-	1		S	-	-
vsubuwm	Vector Subtract Unsigned Word Modulo	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
vsubuws	Vector Subtract Unsigned Word Saturate	-	-	ALU2	VR		sat	2	3	3	1	V	-	1	3	S	-	-

Table A-1. Instruction Properties (Sheet 80 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
vsum2sws	Vector Sum across Half Signed Word Saturate	-	-	DP-XC	VR		sat	2	7	7	1	V	-	1	3	S	-	-
vsum4sbs	Vector Sum across Quarter Signed Byte Saturate	-	-	DP-XC	VR		sat	2	7	7	1	V	-	1	3	S	-	-
vsum4shs	Vector Sum across Quarter Signed Hword Saturate	-	-	DP-XC	VR		sat	2	7	7	1	V	-	1	3	S	-	-
vsum4ubs	Vector Sum across Quarter Unsigned Byte Saturate	-	-	DP-XC	VR		sat	2	7	7	1	V	-	1	3	S	-	-
vsumsws	Vector Sum across Signed Word Saturate	-	-	DP-XC	VR		sat	2	7	7	1	V	-	1	3	S	-	-
vupkhpx	Vector Unpack High Pixel	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vupkhsb	Vector Unpack High Signed Byte	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vupkhsh	Vector Unpack High Signed Hword	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vupkhsw	Vector Unpack High Signed Word	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vupklpx	Vector Unpack Low Pixel	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vupklsb	Vector Unpack Low Signed Byte	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vupklsh	Vector Unpack Low Signed Hword	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vupklsw	Vector Unpack Low Signed Word	-	-	PM	VR			2	3	3	1	V	-	1		S	-	-
vxor	Vector Logical XOR	-	-	ALU	VR			2	2	2	1	V	-	1		S	-	-
wait	Wait for Interrupt	-	-	ALU				4	2	2	1	-	-			-	-	-
xor	XOR	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
xor.	XOR and Record	-	-	ALU	GPR	CR	fxcc	4	2	2	1	-	-			-	-	-
xori	XOR Immediate	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
xoris	XOR Immediate Shifted	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
xsabsdp	VSX Scalar Absolute DP	-	-	ALU	VSR			4	2	2	1	-	-			-	-	-

Table A-1. Instruction Properties (Sheet 81 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
xsabsqp	VSX Scalar Absolute QP	-	-	ALU	VR			4	2	2	1	V	-	1		S	-	-
xsadddp	VSX Scalar Add DP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xsaddqp	VSX Scalar Add QP	-	-	DFU	VR		fpcc,fric,excp	1	12	12	1	V	-	1	3	S	-	-
xsaddqpo	VSX Scalar Add QP and Record OV	-	-	DFU	VR		fpcc,fric,excp	1	12	12	1	V	-	1	3	S	-	-
xsaddsp	VSX Scalar Add SP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xscmpeqdp	VSX Scalar Compare Equal Double-Precision	-	-	ALU2	VSR		excp	4	3	3	1	-	-		3	-	-	-
xscmpexpdp	VSX Scalar Compare Exponents DP	-	-	ALU2		CR	fpcc	4	3	3	1	-	-			-	-	-
xscmpexpqp	VSX Scalar Compare Exponents QP	-	-	DX		CR	excp,fpcc	2	3	3	1	V	-	1	3	S	-	-
xscmpgedp	VSX Scalar Compare Greater Than or Equal Double-Precision	-	-	ALU2	VSR		excp	4	3	3	1	-	-		3	-	-	-
xscmpgtdp	VSX Scalar Compare Greater Than Double-Precision	-	-	ALU2	VSR		excp	4	3	3	1	-	-		3	-	-	-
xscmpodp	VSX Scalar Compare Ordered DP	-	-	ALU2		CR	excp,fpcc	4	3	3	1	-	-		3	-	-	-
xscmpoqp	VSX Scalar Compare Ordered QP	-	-	DX		CR	excp,fpcc	2	3	3	1	V	-	1	3	S	-	-
xscmpudp	VSX Scalar Compare Unordered DP	-	-	ALU2		CR	excp,fpcc	4	3	3	1	-	-		3	-	-	-
xscmpuqp	VSX Scalar Compare Unordered QP	-	-	DX		CR	excp,fpcc	2	3	3	1	V	-	1	3	S	-	-
xscpsgndp	VSX Scalar Copy Sign DP	-	-	ALU	VSR			4	2	2	1	-	-			-	-	-
xscpsgnqp	VSX Scalar Copy Sign QP	-	-	ALU	VR			4	2	2	1	V	-	1		S	-	-
xscvdphp	VSX Scalar Convert DP to HP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xscvdpqp	VSX Scalar Convert DP to QP	-	-	DFU	VR		fpcc,fric,excp	1	12	12	1	V	-	1	3	S	-	-
xscvdpsp	VSX Scalar Convert DP to SP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-

Table A-1. Instruction Properties (Sheet 82 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
xscvdpspn	VSX Scalar Convert DP to SP Non-signalling	-	-	DP	VSR			4	5	7	1	-	-			-	-	-
xscvdpsxds	VSX Scalar Convert DP to Signed Dword truncate	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xscvdpsxws	VSX Scalar Convert DP to Signed Word truncate	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xscvdpuxds	VSX Scalar Convert DP to Unsigned Dword truncate	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xscvdpuxws	VSX Scalar Convert DP to Unsigned Word truncate	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xscvhpdp	VSX Scalar Convert HP to DP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xscvqpdp	VSX Scalar Convert QP to DP	-	-	DFU	VR		fpcc,fric,excp	1	12	12	1	V	-	1	3	S	-	-
xscvqpdpo	VSX Scalar Convert QP to DP and Record OV	-	-	DFU	VR		fpcc,fric,excp	1	12	12	1	V	-	1	3	S	-	-
xscvqpsdz	VSX Scalar Convert QP to Signed Dword truncate	-	-	DFU	VR		fpcc,fric,excp	1	12	12	1	V	-	1	3	S	-	-
xscvqpswz	VSX Scalar Convert QP to Signed Word truncate	-	-	DFU	VR		fpcc,fric,excp	1	12	12	1	V	-	1	3	S	-	-
xscvqpudz	VSX Scalar Convert QP to Unsigned Dword truncate	-	-	DFU	VR		fpcc,fric,excp	1	12	12	1	V	-	1	3	S	-	-
xscvqpuwz	VSX Scalar Convert QP to Unsigned Word truncate	-	-	DFU	VR		fpcc,fric,excp	1	12	12	1	V	-	1	3	S	-	-
xscvsdqp	VSX Scalar Convert Signed Dword to QP	-	-	DFU	VR		fpcc,fric	1	12	12	1	V	-	1	3	S	-	-
xscvspdp	VSX Scalar Convert SP to DP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xscvspdpn	VSX Scalar Convert SP to DP Non-signalling	-	-	ALU2	VSR			4	3	3	1	-	-			-	-	-
xscvsxddp	VSX Scalar Convert Signed Dword to DP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-

Table A-1. Instruction Properties (Sheet 83 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
xscvsxdsp	VSX Scalar Convert Signed Dword to SP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xscvudqp	VSX Scalar Convert Unsigned Dword to QP	-	-	DFU	VR		fpcc,fric	1	12	12	1	V	-	1	3	S	-	-
xscvuxddp	VSX Scalar Convert Unsigned Dword to DP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xscvuxdsp	VSX Scalar Convert Unsigned Dword to SP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xsdivdp	VSX Scalar Divide DP	-	-	DP	VSR		fpcc,fric,excp	4/21	27	33	7-8	-	-		3	-	-	-
xsdivqp	VSX Scalar Divide QP	-	-	DFU	VR		fpcc,fric,excp	1/45	56	58	44	V	-	1	3	S	-	-
xsdivqpo	VSX Scalar Divide QP and Record OV	-	-	DFU	VR		fpcc,fric,excp	1/45	56	58	44	V	-	1	3	S	-	-
xsdivsp	VSX Scalar Divide SP	-	-	DP	VSR		fpcc,fric,excp	4/16	22	22	5	-	-		3	-	-	-
xsiexpdp	VSX Scalar Insert Exponent DP	-	-	ALU	VSR			4	2	2	1	R	-			-	-	-
xsiexpqp	VSX Scalar Insert Exponent QP	-	-	ALU	VR			4	2	2	1	V	-	1		S	-	-
xsmaddadp	VSX Scalar Multiply-Add Type-A DP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
xsmaddasp	VSX Scalar Multiply-Add Type-A SP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
xsmaddmdp	VSX Scalar Multiply-Add Type-M DP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
xsmaddmsp	VSX Scalar Multiply-Add Type-M SP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
xsmaddqp	VSX Scalar Multiply-Add QP	-	-	DFU	VR		fpcc,fric,excp	1/13	24	24	12	V	-	1	3	S	-	-
xsmaddqpo	VSX Scalar Multiply-Add QP and Record OV	-	-	DFU	VR		fpcc,fric,excp	1/13	24	24	12	V	-	1	3	S	-	-
xsmaxcdp	VSX Scalar Maximum Type-C Double-Precision	-	-	ALU2	VSR		excp	4	3	3	1	-	-		3	-	-	-
xsmaxdp	VSX Scalar Maximum DP	-	-	ALU2	VSR		excp	4	3	3	1	-	-		3	-	-	-

Table A-1. Instruction Properties (Sheet 84 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
xsmaxjdp	VSX Scalar Maximum Type-J Double-Precision	-	-	ALU2	VSR		excp	4	3	3	1	-	-		3	-	-	-
xsmincdp	VSX Scalar Minimum Type-C Double-Precision	-	-	ALU2	VSR		excp	4	3	3	1	-	-		3	-	-	-
xsmindp	VSX Scalar Minimum DP	-	-	ALU2	VSR		excp	4	3	3	1	-	-		3	-	-	-
xsminjdp	VSX Scalar Minimum Type-J Double-Precision	-	-	ALU2	VSR		excp	4	3	3	1	-	-		3	-	-	-
xsmsubadp	VSX Scalar Multiply-Subtract Type-A DP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
xsmsubasp	VSX Scalar Multiply-Subtract Type-A SP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
xsmsubmdp	VSX Scalar Multiply-Subtract Type-M DP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
xsmsubmsp	VSX Scalar Multiply-Subtract Type-M SP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
xsmsubqp	VSX Scalar Multiply-Subtract QP	-	-	DFU	VR		fpcc,fric,excp	1/13	24	24	12	V	-	1	3	S	-	-
xsmsubqpo	VSX Scalar Multiply-Subtract QP and Record OV	-	-	DFU	VR		fpcc,fric,excp	1/13	24	24	12	V	-	1	3	S	-	-
xsmuldp	VSX Scalar Multiply DP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
xsmulqp	VSX Scalar Multiply QP	-	-	DFU	VR		fpcc,fric,excp	1/13	24	24	12	V	-	1	3	S	-	-
xsmulqpo	VSX Scalar Multiply QP and Record OV	-	-	DFU	VR		fpcc,fric,excp	1/13	24	24	12	V	-	1	3	S	-	-
xsmulsp	VSX Scalar Multiply SP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
xsnabsdp	VSX Scalar Negative Absolute DP	-	-	ALU	VSR			4	2	2	1	-	-			-	-	-
xsnabsqp	VSX Scalar Negative Absolute QP	-	-	ALU	VR			4	2	2	1	V	-	1		S	-	-
xsnegdp	VSX Scalar Negate DP	-	-	ALU	VSR			4	2	2	1	-	-			-	-	-
xsnegqp	VSX Scalar Negate QP	-	-	ALU	VR			4	2	2	1	V	-	1		S	-	-

Table A-1. Instruction Properties (Sheet 85 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
xsnmaddadp	VSX Scalar Negative Multiply-Add Type-A DP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
xsnmaddasp	VSX Scalar Negative Multiply-Add Type-A SP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
xsnmaddmdp	VSX Scalar Negative Multiply-Add Type-M DP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
xsnmaddmsp	VSX Scalar Negative Multiply-Add Type-M SP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
xsnmaddqp	VSX Scalar Negative Multiply-Add QP	-	-	DFU	VR		fpcc,fric,excp	1/13	24	24	12	V	-	1	3	S	-	-
xsnmaddqpo	VSX Scalar Negative Multiply-Add QP and Record OV	-	-	DFU	VR		fpcc,fric,excp	1/13	24	24	12	V	-	1	3	S	-	-
xsnmsubadp	VSX Scalar Negative Multiply-Subtract Type-A DP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
xsnmsubasp	VSX Scalar Negative Multiply-Subtract Type-A SP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
xsnmsubmdp	VSX Scalar Negative Multiply-Subtract Type-M DP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
xsnmsubmsp	VSX Scalar Negative Multiply-Subtract Type-M SP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	R	-		3	-	-	-
xsnmsubqp	VSX Scalar Negative Multiply-Subtract QP	-	-	DFU	VR		fpcc,fric,excp	1/13	24	24	12	V	-	1	3	S	-	-
xsnmsubqpo	VSX Scalar Negative Multiply-Subtract QP and Record OV	-	-	DFU	VR		fpcc,fric,excp	1/13	24	24	12	V	-	1	3	S	-	-
xsrdpi	VSX Scalar Round DP to Integral to Nearest Away	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xsrdpic	VSX Scalar Round DP to Integral using Current rounding mode	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xsrdpim	VSX Scalar Round DP to Integral toward -Infinity	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-

Table A-1. Instruction Properties (Sheet 86 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
xsrdpip	VSX Scalar Round DP to Integral toward +Infinity	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xsrdpiz	VSX Scalar Round DP to Integral toward Zero	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xsredp	VSX Scalar Reciprocal Estimate DP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xsresp	VSX Scalar Reciprocal Estimate SP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xsrqpi	VSX Scalar Round QP to Integral	-	-	DFU	VR		fpcc,fric,excp	1	12	12	1	V	-	1	3	S	-	-
xsrqpxp	VSX Scalar Round QP to XP	-	-	DFU	VR		fpcc,fric,excp	1	12	12	1	V	-	1	3	S	-	-
xsrsp	VSX Scalar Round DP to SP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xsrsqrtedp	VSX Scalar Reciprocal Square Root Estimate DP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xsrsqrtesp	VSX Scalar Reciprocal Square Root Estimate SP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xssqrtdp	VSX Scalar Square Root DP	-	-	DP	VSR		fpcc,fric,excp	4/30	36	36	10	-	-		3	-	-	-
xssqrtqp	VSX Scalar Square Root QP	-	-	DFU	VR		fpcc,fric,excp	1/63	74	76	62	V	-	1	3	S	-	-
xssqrtqpo	VSX Scalar Square Root QP and Record OV	-	-	DFU	VR		fpcc,fric,excp	1/63	74	76	62	V	-	1	3	S	-	-
xssqrtsp	VSX Scalar Square Root SP	-	-	DP	VSR		fpcc,fric,excp	4/20	26	26	5	-	-		3	-	-	-
xssubdp	VSX Scalar Subtract DP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xssubqp	VSX Scalar Subtract QP	-	-	DFU	VR		fpcc,fric,excp	1	12	12	1	V	-	1	3	S	-	-
xssubqpo	VSX Scalar Subtract QP and Record OV	-	-	DFU	VR		fpcc,fric,excp	1	12	12	1	V	-	1	3	S	-	-
xssubsp	VSX Scalar Subtract SP	-	-	DP	VSR		fpcc,fric,excp	4	5	7	1	-	-		3	-	-	-
xstdivdp	VSX Scalar Test for software Divide DP	-	-	ALU2		CR		4	3	3	1	-	-			-	-	-

Table A-1. Instruction Properties (Sheet 87 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
xstsqrtdp	VSX Scalar Test for software Square Root DP	-	-	ALU2		CR		4	3	3	1	-	-			-	-	-
xststdcdp	VSX Scalar Test Data Class DP	-	-	ALU2		CR	fpcc	4	3	3	1	R	-			-	-	-
xststdcqp	VSX Scalar Test Data Class QP	-	-	DX		CR	fpcc	2	3	3	1	V	-	1		S	-	-
xststdcsp	VSX Scalar Test Data Class SP	-	-	ALU2		CR	fpcc	4	3	3	1	R	-			-	-	-
xsxexpdp	VSX Scalar Extract Exponent DP	-	-	ALU	GPR			4	2	2	1	-	-			-	-	-
xsxexpqp	VSX Scalar Extract Exponent QP	-	-	ALU	VR			4	2	2	1	V	-	1		S	-	-
xsxsigdp	VSX Scalar Extract Significand DP	-	-	ALU2	GPR			4	3	3	1	-	-			-	-	-
xsxsigqp	VSX Scalar Extract Significand QP	-	-	DX	VR			2	3	3	1	V	-	1		S	-	-
xvabsdp	VSX Vector Absolute DP	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xvabssp	VSX Vector Absolute SP	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xvadddp	VSX Vector Add DP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvaddsp	VSX Vector Add SP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvcmpeqdp	VSX Vector Compare Equal DP	-	-	ALU2	VSR		excp	2	3	3	1	V	-	1	3	S	-	-
xvcmpeqdp.	VSX Vector Compare Equal DP and Record	-	-	ALU2	VSR	CR	excp	2	3	3	1	V	-	1	3	S	-	-
xvcmpeqsp	VSX Vector Compare Equal SP	-	-	ALU2	VSR		excp	2	3	3	1	V	-	1	3	S	-	-
xvcmpeqsp.	VSX Vector Compare Equal SP and Record	-	-	ALU2	VSR	CR	excp	2	3	3	1	V	-	1	3	S	-	-
xvcmpgedp	VSX Vector Compare Greater Than or Equal DP	-	-	ALU2	VSR		excp	2	3	3	1	V	-	1	3	S	-	-
xvcmpgedp.	VSX Vector Compare Greater Than or Equal DP and Record	-	-	ALU2	VSR	CR	excp	2	3	3	1	V	-	1	3	S	-	-
xvcmpgesp	VSX Vector Compare Greater Than or Equal SP	-	-	ALU2	VSR		excp	2	3	3	1	V	-	1	3	S	-	-

Table A-1. Instruction Properties (Sheet 88 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
xvcmpgesp.	VSX Vector Compare Greater Than or Equal SP and Record	-	-	ALU2	VSR	CR	excp	2	3	3	1	V	-	1	3	S	-	-
xvcmpgtdp	VSX Vector Compare Greater Than DP	-	-	ALU2	VSR		excp	2	3	3	1	V	-	1	3	S	-	-
xvcmpgtdp.	VSX Vector Compare Greater Than DP and Record	-	-	ALU2	VSR	CR	excp	2	3	3	1	V	-	1	3	S	-	-
xvcmpgtsp	VSX Vector Compare Greater Than SP	-	-	ALU2	VSR		excp	2	3	3	1	V	-	1	3	S	-	-
xvcmpgtsp.	VSX Vector Compare Greater Than SP and Record	-	-	ALU2	VSR	CR	excp	2	3	3	1	V	-	1	3	S	-	-
xvcpsgndp	VSX Vector Copy Sign DP	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xvcpsgnsp	VSX Vector Copy Sign SP	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xvcvdpsp	VSX Vector Convert DP to SP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvcvdpsxds	VSX Vector Convert DP to Signed Dword truncate	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvcvdpsxws	VSX Vector Convert DP to Signed Word truncate	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvcvdpuxds	VSX Vector Convert DP to Unsigned Dword truncate	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvcvdpuxws	VSX Vector Convert DP to Unsigned Word truncate	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvcvhpsp	VSX Vector Convert HP to SP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvcvspdp	VSX Vector Convert SP to DP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvcvsphp	VSX Vector Convert SP to HP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvcvspsxds	VSX Vector Convert SP to Signed Dword truncate	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvcvspsxws	VSX Vector Convert SP to Signed Word truncate	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-

Table A-1. Instruction Properties (Sheet 89 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
xvcvspuxds	VSX Vector Convert SP to Unsigned Dword truncate	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvcvspuxws	VSX Vector Convert SP to Unsigned Word truncate	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvcvsxddp	VSX Vector Convert Signed Dword to DP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvcvsxdsp	VSX Vector Convert Signed Dword to SP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvcvsxwdp	VSX Vector Convert Signed Word to DP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvcvsxwsp	VSX Vector Convert Signed Word to SP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvcvuxddp	VSX Vector Convert Unsigned Dword to DP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvcvuxdsp	VSX Vector Convert Unsigned Dword to SP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvcvuxwdp	VSX Vector Convert Unsigned Word to DP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvcvuxwsp	VSX Vector Convert Unsigned Word to SP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvdivdp	VSX Vector Divide DP	-	-	DP	VSR		excp	2/21	27	33	7-8	V	-	1	3	S	-	-
xvdivsp	VSX Vector Divide SP	-	-	DP	VSR		excp	2/18	24	24	8	V	-	1	3	S	-	-
xviexpdp	VSX Vector Insert Exponent DP	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xviexpsp	VSX Vector Insert Exponent SP	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xvmaddadp	VSX Vector Multiply-Add Type-A DP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvmaddasp	VSX Vector Multiply-Add Type-A SP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvmaddmdp	VSX Vector Multiply-Add Type-M DP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-

Table A-1. Instruction Properties (Sheet 90 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
xvmaddmsp	VSX Vector Multiply-Add Type-M SP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvmaxdp	VSX Vector Maximum DP	-	-	ALU2	VSR		excp	2	3	3	1	V	-	1	3	S	-	-
xvmaxsp	VSX Vector Maximum SP	-	-	ALU2	VSR		excp	2	3	3	1	V	-	1	3	S	-	-
xvmindp	VSX Vector Minimum DP	-	-	ALU2	VSR		excp	2	3	3	1	V	-	1	3	S	-	-
xvminsp	VSX Vector Minimum SP	-	-	ALU2	VSR		excp	2	3	3	1	V	-	1	3	S	-	-
xvmsubadp	VSX Vector Multiply-Subtract Type-A DP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvmsubasp	VSX Vector Multiply-Subtract Type-A SP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvmsubmdp	VSX Vector Multiply-Subtract Type-M DP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvmsubmsp	VSX Vector Multiply-Subtract Type-M SP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvmuldp	VSX Vector Multiply DP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvmulsp	VSX Vector Multiply SP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvnabsdp	VSX Vector Negative Absolute DP	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xvnabssp	VSX Vector Negative Absolute SP	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xvnegdp	VSX Vector Negate DP	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xvnegsp	VSX Vector Negate SP	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xvnmaddadp	VSX Vector Negative Multiply-Add Type-A DP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvnmaddasp	VSX Vector Negative Multiply-Add Type-A SP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvnmaddmdp	VSX Vector Negative Multiply-Add Type-M DP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-

Table A-1. Instruction Properties (Sheet 91 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
xvnmaddmsp	VSX Vector Negative Multiply-Add Type-M SP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvnmsubadp	VSX Vector Negative Multiply-Subtract Type-A DP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvnmsubasp	VSX Vector Negative Multiply-Subtract Type-A SP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvnmsubmdp	VSX Vector Negative Multiply-Subtract Type-M DP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvnmsubmsp	VSX Vector Negative Multiply-Subtract Type-M SP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvrdpi	VSX Vector Round DP to Integral to Nearest Away	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvrdpic	VSX Vector Round DP to Integral using Current rounding mode	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvrdpim	VSX Vector Round DP to Integral toward -Infinity	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvrdpip	VSX Vector Round DP to Integral toward +Infinity	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvrdpiz	VSX Vector Round DP to Integral toward Zero	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvredp	VSX Vector Reciprocal Estimate DP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvresp	VSX Vector Reciprocal Estimate SP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvrspi	VSX Vector Round SP to Integral to Nearest Away	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvrspic	VSX Vector Round SP to Integral using Current rounding mode	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvrspim	VSX Vector Round SP to Integral toward -Infinity	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-

Table A-1. Instruction Properties (Sheet 92 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
xvrspip	VSX Vector Round SP to Integral toward +Infinity	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvrspiz	VSX Vector Round SP to Integral toward Zero	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvrsqrtedp	VSX Vector Reciprocal Square Root Estimate DP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvrsqrtesp	VSX Vector Reciprocal Square Root Estimate SP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvsqrtdp	VSX Vector Square Root DP	-	-	DP	VSR		excp	2/30	36	36	10	V	-	1	3	S	-	-
xvsqrtsp	VSX Vector Square Root SP	-	-	DP	VSR		excp	2/21	27	27	10	V	-	1	3	S	-	-
xvsubdp	VSX Vector Subtract DP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvsubsp	VSX Vector Subtract SP	-	-	DP	VSR		excp	2	5	7	1	V	-	1	3	S	-	-
xvtdivdp	VSX Vector Test for software Divide DP	-	-	ALU2		CR		4	3	3	1	V	-	1		S	-	-
xvtdivsp	VSX Vector Test for software Divide SP	-	-	ALU2		CR		4	3	3	1	V	-	1		S	-	-
xvtsqrtdp	VSX Vector Test for software Square Root DP	-	-	ALU2		CR		4	3	3	1	V	-	1		S	-	-
xvtsqrtsp	VSX Vector Test for software Square Root SP	-	-	ALU2		CR		4	3	3	1	V	-	1		S	-	-
xvtstdcdp	VSX Vector Test Data Class DP	-	-	ALU2	VSR			2	3	3	1	V	-	1		S	-	-
xvtstdcsp	VSX Vector Test Data Class SP	-	-	ALU2	VSR			2	3	3	1	V	-	1		S	-	-
xvxexpdp	VSX Vector Extract Exponent DP	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xvxexpsp	VSX Vector Extract Exponent SP	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xvxsigdp	VSX Vector Extract Significand DP	-	-	ALU2	VSR			2	3	3	1	V	-	1		S	-	-
xvxsigsp	VSX Vector Extract Significand SP	-	-	ALU2	VSR			2	3	3	1	V	-	1		S	-	-

Table A-1. Instruction Properties (Sheet 93 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source	Issue Synchronized	Issue Depend on Previous Iop	Issue Next-to-Complete
xxbrd	VSX Vector Byte-Reverse Dword	-	-	PM	VSR			2	3	3	1	V	-	1		S	-	-
xxbrh	VSX Vector Byte-Reverse Hword	-	-	PM	VSR			2	3	3	1	V	-	1		S	-	-
xxbrq	VSX Vector Byte-Reverse Qword	-	-	PM	VSR			2	3	3	1	V	-	1		S	-	-
xxbrw	VSX Vector Byte-Reverse Word	-	-	PM	VSR			2	3	3	1	V	-	1		S	-	-
xxextractuw	VSX Vector Extract Unsigned Word	-	-	PM	VSR			2	3	3	1	V	-	1		S	-	-
xxinsertw	VSX Vector Insert Word	-	-	PM	VSR			2	3	3	1	V	-	1		S	-	-
xxland	VSX Vector Logical AND	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xxlandc	VSX Vector Logical AND with Complement	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xxleqv	VSX Vector Logical Equivalence	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xxlnand	VSX Vector Logical NAND	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xxlnor	VSX Vector Logical NOR	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xxlor	VSX Vector Logical OR	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xxlorc	VSX Vector Logical OR with Complement	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xxlxor	VSX Vector Logical XOR	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-
xxmrghw	VSX Vector Merge Word High	-	-	PM	VSR			2	3	3	1	V	-	1		S	-	-
xxmrglw	VSX Vector Merge Word Low	-	-	PM	VSR			2	3	3	1	V	-	1		S	-	-
xxperm	VSX Vector Permute	-	-	PM	VSR			2	3	3	1	V	-	1		S	-	-
xxpermr	VSX Vector Permute Right-indexed	-	-	PM	VSR			2	3	3	1	V	-	1		S	-	-
xxsel	VSX Vector Select	-	-	ALU	VSR			2	2	2	1	V	-	1		S	-	-

Table A-1. Instruction Properties (Sheet 94 of 94)

Instruction Mnemonic and Name		Cracked/Expanded	Operation Number	Pipe Class	Main DST	CR DST	XER/FPSCR DST	Maximum Operations per Cycle	Latency (Minimum)	Latency (Maximum)	Pipe Busy Cycles (Minimum)	Dispatch Rule	Dispatch Interlock	Additional Dispatch to Issue Latency	Additional Latency for CR/XER/FPSCR/VSCR Source
xxsldwi	VSX Vector Shift Left Double by Word Immediate	-	-	PM	VSR			2	3	3	1	V	-	1	
xxspltib	VSX Vector Splat Immediate Byte	-	-	PM	VSR			2	3	3	1	V	-	1	
xxspltw	VSX Vector Splat Word	-	-	PM	VSR			2	3	3	1	V	-	1	

Appendix B. tlbie and tlbiel Encodings for Radix Translations

Table B-1. lists the **tlbie** encodings for Radix translations with R = '1' and GTSE = '1', and *Table B-2.* on page 476 lists the **tlbiel** encodings for Radix translations with R = '1'.

Note: The yellow rows result in a Machine Check interrupt (invalid form of **tlbie/tlbiel** instruction) and the red rows result in a privileged instruction interrupt.

Table B-1. ***tlbie*** *Encodings for Radix Translations (with R = '1' and GTSE = '1')* (Sheet 1 of 7)

IS	HV	PRS	RIC	EA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	Invalid Form Machine Check Case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
0	0	0	0									NO	YES
0	0	0	1									YES	YES
0	0	0	2									YES	YES
0	0	0	3									YES	YES
0	0	1	0	YES	YES	LPIDR	YES	Guest entry with gEA match	NO	YES - matching address, PID and LPID (by thread ID)	NO	NO	NO
0	0	1	1									YES	NO
0	0	1	2									YES	NO
0	0	1	3									YES	NO
0	1	0	0	YES	YES	RS	NO	Partition-scoped host entry with gRA/hEA match	NO	YES - matching address and LPID (by thread ID)	NO	NO	NO
0	1	0	1									YES	NO
0	1	0	2									YES	NO
0	1	0	3									YES	NO

*Table B-1. **tlbie** Encodings for Radix Translations (with R = '1' and GTSE = '1')* (Sheet 2 of 7)

IS	HV	PRS	RIC	EA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	Invalid Form Machine Check Case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
0	1	1	0	YES	YES	RS	YES	Guest entry with gEA match if LPID ≠ 0 Process-scoped host entry with hEA match if LPID = 0	NO	YES - matching address, PID and LPID (by thread ID)	NO	NO	NO
0	1	1	1									YES	NO
0	1	1	2									YES	NO
0	1	1	3									YES	NO
1	0	0	0									YES	YES
1	0	0	1									YES	YES
1	0	0	2									YES	YES
1	0	0	3									YES	YES
1	0	1	0	NO	YES	LPIDR	YES	Guest entries with matching PID	NO	YES - all with matching LPID and PID (by thread ID)	NO	NO	NO
1	0	1	1	NO	YES	LPIDR	YES	NONE	Guest PWC with matching PID (for a matching LPID)	NO	NO	NO	NO
1	0	1	2	NO	YES	LPIDR	YES	Guest entries with matching PID (for a matching LPID)	Guest PWC with matching PID (for a matching LPID)	YES - all with matching LPID and PID (by thread ID)	YES - process table regardless of LPID	NO	NO
1	0	1	3									YES	NO
1	1	0	0									YES	NO

*Table B-1. **tlbie** Encodings for Radix Translations (with R = '1' and GTSE = '1')* (Sheet 3 of 7)

IS	HV	PRS	RIC	EA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	Invalid Form Machine Check Case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
1	1	0	1									YES	NO
1	1	0	2									YES	NO
1	1	0	3									YES	NO
1	1	1	0	NO	YES	RS	YES	Guest entries with matching PID if LPID ≠ 0 Process-scoped host entries with matching PID if LPID = 0	NO	YES - all with matching LPID and PID (by thread ID)	NO	NO	NO
1	1	1	1	NO	YES	RS	YES	NONE	Guest PWC with matching PID if LPID ≠ 0 Process-scoped host PWC with matching PID if LPID = 0	NO	NO	NO	NO
1	1	1	2	NO	YES	RS	YES	Guest entries with matching PID if LPID ≠ 0 Process-scoped host entries with matching PID if LPID = 0	Guest PWC with matching PID if LPID ≠ 0 Process-scoped host PWC with matching PID if LPID = 0	YES - all with matching LPID and PID (by thread ID)	YES - process table regard-less of LPID	NO	NO
1	1	1	3									YES	NO
2	0	0	0									NO	YES
2	0	0	1									NO	YES
2	0	0	2									NO	YES
2	0	0	3									YES	YES

*Table B-1. **tlbie** Encodings for Radix Translations (with R = '1' and GTSE = '1')* (Sheet 4 of 7)

IS	HV	PRS	RIC	EA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	Invalid Form Machine Check Case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
2	0	1	0	NO	YES	LPIDR	NO	Guest entries with matching LPID	NO	YES - all with matching LPID (by thread ID)	NO	NO	NO
2	0	1	1	NO	YES	LPIDR	NO	NONE	Guest PWC with matching LPID	NO	NO	NO	NO
2	0	1	2	NO	YES	LPIDR	NO	Guest entries with matching LPID	Guest PWC with matching LPID	YES - all with matching LPID (by thread ID)	YES - process table regard-less of LPID	NO	NO
2	0	1	3									YES	NO
2	1	0	0	NO	YES	RS	NO	Partition-scoped host entries with matching LPID	NO	YES - all with matching LPID (by thread ID)	NO	NO	NO
2	1	0	1	NO	YES	RS	NO	NONE	Partition-scoped host PWC with matching LPID	NO	NO	NO	NO
2	1	0	2	NO	YES	RS	NO	Partition-scoped host entries with matching LPID	Partition-scoped host PWC with matching LPID	YES - all with matching LPID (by thread ID)	YES - all pro-cess and parti-tion table caching regardless of LPID	NO	NO
2	1	0	3									YES	NO
2	1	1	0	NO	YES	RS	NO	Guest entries with matching LPID if LPID ≠ 0 Process-scoped host entries with matching LPID if LPID = 0	NO	YES - all with matching LPID (by thread ID)	NO	NO	NO

*Table B-1. **tlbie** Encodings for Radix Translations (with R = '1' and GTSE = '1')* (Sheet 5 of 7)

IS	HV	PRS	RIC	EA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	Invalid Form Machine Check Case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
2	1	1	1	NO	YES	RS	NO	NONE	Guest PWC with matching LPID if LPID ≠ 0 Process-scoped host PWC with matching LPID if LPID = 0	NO	NO	NO	NO
2	1	1	2	NO	YES	RS	NO	Guest entries with matching LPID if LPID ≠ 0 Process-scoped host entries with matching LPID if LPID = 0	Guest PWC with matching LPID if LPID ≠ 0 Process-scoped host PWC with matching LPID if LPID = 0	YES - all with matching LPID (by thread ID)	YES - process table regardless of LPID	NO	NO
2	1	1	3									YES	NO
3	0	0	0									NO	YES
3	0	0	1									NO	YES
3	0	0	2									NO	YES
3	0	0	3									YES	YES
3	0	1	0	NO	YES	LPIDR	NO	Guest entries with matching LPID	NO	YES - all with matching LPID (by thread ID)	NO	NO	NO
3	0	1	1	NO	YES	LPIDR	NO	NONE	Guest PWC with matching LPID	NO	NO	NO	NO
3	0	1	2	NO	YES	LPIDR	NO	Guest entries with matching LPID	Guest PWC with matching LPID	YES - all with matching LPID (by thread ID)	YES - process table regard-less of LPID	NO	NO
3	0	1	3									YES	NO

*Table B-1. **tlbie** Encodings for Radix Translations (with R = '1' and GTSE = '1')* (Sheet 6 of 7)

IS	HV	PRS	RIC	EA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	Invalid Form Machine Check Case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
3	1	0	0	NO	NO	NA	NO	All partition-scoped host entries regardless of LPID	NO	YES - regardless of LPID (thread)	NO	NO	NO
3	1	0	1	NO	NO	NA	NO	NONE	All partition-scoped host PWC entries regardless of LPID	NO	NO	NO	NO
3	1	0	2	NO	NO	NA	NO	All partition-scoped host entries regardless of LPID	All partition-scoped host PWC entries regardless of LPID	YES - regardless of LPID (thread)	YES - all process and partition table regardless of LPID	NO	NO
3	1	0	3									YES	NO
3	1	1	0	NO	NO	NA	NO	Guest entries regardless of LPID if LPID ≠ 0 Process-scoped host entries with matching LPID if LPID = 0	NO	YES - regardless of LPID (thread)	NO	NO	NO

*Table B-1. **tlbie** Encodings for Radix Translations (with R = '1' and GTSE = '1')* (Sheet 7 of 7)

IS	HV	PRS	RIC	EA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	Invalid Form Machine Check Case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
3	1	1	1	NO	NO	NA	NO	NONE	All guest PWC regardless of LPID if LPID ≠ 0 Process-scoped host PWC regard-less of LPID if LPID = 0	NO	NO	NO	NO
3	1	1	2	NO	NO	NA	NO	Guest entries regardless of LPID if LPID ≠ 0 Process-scoped host entries with matching LPID if LPID = 0	All guest PWC regardless of LPID if LPID ≠ 0 Process-scoped host PWC regard-less of LPID if LPID = 0	YES - regard-less of LPID (thread)	YES - process table regardless of LPID	NO	NO
3	1	1	3									YES	NO

*Table B-2. **tlbiel** Encodings for Radix Translations (with R = '1')* (Sheet 1 of 9)

IS	HV	PRS	RIC	EA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	Invalid Form Machine Check case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
0	0	0	0									NO	YES
0	0	0	1									YES	YES
0	0	0	2									YES	YES
0	0	0	3									YES	YES
0	0	1	0	YES	YES	LPIDR	YES	Guest entry with gEA match	NO	YES - matching address for that thread only	NO	NO	NO
0	0	1	1									YES	NO
0	0	1	2									YES	NO
0	0	1	3									YES	NO
0	1	0	0	YES	YES	LPIDR	NO	Partition-scoped host entry with gRA/hEA match	NO	YES - matching address for that thread only	NO	NO	NO
0	1	0	1									YES	NO
0	1	0	2									YES	NO
0	1	0	3									YES	NO
0	1	1	0	YES	YES	LPIDR	YES	Guest entry with gEA match if LPID ≠ 0 Process-Scoped Host entry with hEA match if LPID = 0	NO	YES - matching address for that thread only	NO	NO	NO
0	1	1	1									YES	NO
0	1	1	2									YES	NO
1	0	0	0									YES	YES

*Table B-2. **tlbiel** Encodings for Radix Translations (with R = '1')* (Sheet 2 of 9)

IS	HV	PRS	RIC	EA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	Invalid Form Machine Check case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
1	0	0	1									YES	YES
1	0	0	2									YES	YES
1	0	0	3									YES	YES
1	0	1	0	NO	YES	LPIDR	YES	Guest entries with matching PID in set specified by RB(40:51)	NO	YES - all for that thread only	NO	NO	NO
1	0	1	1	NO	YES	LPIDR	YES	NONE	Guest PWC with matching PID (for a matching LPID)	NO	NO	NO	NO
1	0	1	2	NO	YES	LPIDR	YES	Guest entries with matching PID in set specified by RB(40:51)	Guest PWC with matching PID (for a matching LPID)	YES - all for that thread only	YES - all process table caching: 1. effPID = PIDR and effLPID = 0 2. effPID = PIDR and effLPID = LPIDR 3. effPID = 0 and effLPID = LPIDR 4. effPID = 0 and effLPID = 0	NO	NO
1	0	1	3									YES	NO
1	1	0	0									YES	NO
1	1	0	1									YES	NO
1	1	0	2									YES	NO
1	1	0	3									YES	NO

*Table B-2. **tlbiel** Encodings for Radix Translations (with R = '1')* (Sheet 3 of 9)

IS	HV	PRS	RIC	EA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	Invalid Form Machine Check case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
1	1	1	0	NO	YES	LPIDR	YES	Guest entries with matching PID in set specified by RB(40:51) if LPID ≠ 0 Process-scoped host entries with matching PID in set specified by RB(40:51) if LPID = 0	NO	YES - all for that thread only	NO	NO	NO
1	1	1	1	NO	YES	LPIDR	YES	NONE	Guest PWC with matching PID if LPID ≠ 0 Process-scoped host PWC with matching PID if LPID = 0	NO	NO	NO	NO
1	1	1	2	NO	YES	LPIDR	YES	Guest entries with matching PID in set specified by RB(40:51) if LPID ≠ 0 Process-scoped host entries with matching PID in set specified by RB(40:51) if LPID = 0	Guest PWC with matching PID if LPID ≠ 0 Process-scoped host PWC with matching PID if LPID = 0	YES - All for that thread only	YES - all process table caching: 1. effPID = PIDR and effLPID = 0 2. effPID=PIDR and effLPID = LPIDR 3. effPID = 0 and effLPID = LPIDR 4. effPID = 0 and effLPID = 0	NO	NO
1	1	1	3									YES	NO
2	0	0	0									NO	YES
2	0	0	1									NO	YES
2	0	0	2									NO	YES
2	0	0	3									YES	YES

*Table B-2. **tlbiel** Encodings for Radix Translations (with R = '1')* (Sheet 4 of 9)

IS	HV	PRS	RIC	EA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	Invalid Form Machine Check case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
2	0	1	0	NO	YES	LPIDR	NO	Guest entries with matching LPID in set specified by RB(40:51)	NO	YES - All for that thread only	NO	NO	NO
2	0	1	1	NO	YES	LPIDR	NO	NONE	Guest PWC with matching LPID	NO	NO	NO	NO
2	0	1	2	NO	YES	LPIDR	NO	Guest entries with matching LPID in set specified by RB(40:51)	Guest PWC with matching LPID	YES - all for that thread only	YES - all process table caching: 1. effPID = PIDR and effLPID = 0 2. effPID = PIDR and effLPID=LPIDR 3. effPID = 0 and effLPID = LPIDR 4. effPID = 0 and effLPID = 0	NO	NO
2	0	1	3									YES	NO
2	1	0	0	NO	YES	LPIDR	NO	All partition-scoped host entries regardless of LPID in set specified by RB(40:51)	NO	YES - all for that thread only	NO	NO	NO
2	1	0	1	NO	YES	LPIDR	NO	NONE	Partition-scoped host PWC with matching LPID	NO	NO	NO	NO

*Table B-2. **tlbiel** Encodings for Radix Translations (with R = '1')* (Sheet 5 of 9)

IS	HV	PRS	RIC	EA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	Invalid Form Machine Check case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
2	1	0	2	NO	YES	LPIDR	NO	All partition-scoped host entries regard-less of LPID in set specified by RB(40:51)	Partition-scoped host PWC with matching LPID	YES - all for that thread only	YES - all process table caching: 1. effPID = PIDR and effLPID = 0 2. effPID = PIDR and effLPID = LPIDR 3. effPID = 0 and effLPID = LPIDR 4. effPID = 0 and effLPID = 0 AND All partition table caching: 1. effLPID = 0 and 2. effLPID = LPIDR	NO	NO
2	1	0	3									YES	NO
2	1	1	0	NO	YES	LPIDR	NO	Guest entries with matching LPID in set specified by RB(40:51) if LPID ≠ 0 Process-scoped host entries with matching LPID in set specified by RB(40:51) if LPID = 0	NO	YES - all for that thread only	NO	NO	NO
2	1	1	1	NO	YES	LPIDR	NO	NONE	Guest PWC with matching LPID if LPID ≠ 0 Process-scoped host PWC with matching LPID if LPID = 0	NO	NO	NO	NO

*Table B-2. **tlbiel** Encodings for Radix Translations (with R = '1')* (Sheet 6 of 9)

IS	HV	PRS	RIC	EA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	Invalid Form Machine Check case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
2	1	1	2	NO	YES	LPIDR	NO	Guest entries with matching LPID in set specified by RB(40:51) if LPID ≠ 0 Process-scoped host entries with matching LPID in set specified by RB(40:51) if LPID = 0	Guest PWC with matching LPID if LPID ≠ 0 Process-scoped host PWC with matching LPID if LPID = 0	YES - all with matching thread	YES - all process table caching: 1. effPID = PIDR and effLPID = 0 2. effPID = PIDR and effLPID = LPIDR 3. effPID = 0 and effLPID = LPIDR 4. effPID = 0 and effLPID = 0	NO	NO
2	1	1	3									YES	NO
3	0	0	0									NO	YES
3	0	0	1									NO	YES
3	0	0	2									NO	YES
3	0	0	3									YES	YES
3	0	1	0	NO	YES	LPIDR	NO	Guest entries with matching LPID in set specified by RB(40:51)	NO	YES - All for that thread only	NO	NO	NO
3	0	1	1	NO	YES	LPIDR	NO	NONE	Guest PWC with matching LPID	NO	NO	NO	NO

*Table B-2. **tlbiel** Encodings for Radix Translations (with R = '1')* (Sheet 7 of 9)

IS	HV	PRS	RIC	EA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	Invalid Form Machine Check case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
3	0	1	2	NO	YES	LPIDR	NO	Guest entries with matching LPID in set specified by RB(40:51)	Guest PWC with matching LPID	YES - all for that thread only	YES - all process table caching: 1. effPID = PIDR and effLPID = 0 2. effPID = PIDR and effLPID = LPIDR 3. effPID = 0 and effLPID = LPIDR 4. effPID = 0 and effLPID = 0	NO	NO
3	0	1	3									YES	NO
3	1	0	0	NO	NO	NA	NO	All partition-scoped host entries regard-less of LPID in set specified by RB(40:51)	NO	YES - all for that thread only	NO	NO	NO
3	1	0	1	NO	NO	NA	NO	NONE	All partition-scoped host PWC entries regardless of LPID	NO	NO	NO	NO

*Table B-2. **tlbiel** Encodings for Radix Translations (with R = '1')* (Sheet 8 of 9)

IS	HV	PRS	RIC	EA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	Invalid Form Machine Check case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
3	1	0	2	NO	NO	NA	NO	All partition-scoped host entries regard-less of LPID in set specified by RB(40:51)	All partition-scoped host PWC entries regardless of LPID	YES - all for that thread only	YES - all process table caching: 1. effPID=PIDR and effLPID = 0 2. effPID=PIDR and effLPID=LPIDR 3. effPID = 0 and effLPID=LPIDR 4. effPID = 0 and effLPID = 0 AND All Partition Table caching: 1. effLPID = 0 and 2. effLPID=LPIDR	NO	NO
3	1	0	3									YES	NO
3	1	1	0	NO	NO	NA	NO	Guest entries regardless of LPID if LPID ≠ 0 in set specified by RB(40:51) Process-scoped host entries with matching LPID if LPID = 0 in set specified by RB(40:51)	NO	YES - all for that thread only	NO	NO	NO

*Table B-2. **tlbiel** Encodings for Radix Translations (with R = '1')* (Sheet 9 of 9)

IS	HV	PRS	RIC	EA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	Invalid Form Machine Check case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
3	1	1	1	NO	NO	NA	NO	NONE	All guest PWC regardless of LPID if LPID ≠ 0 Process-scoped host PWC regardless of LPID if LPID = 0	NO	NO	NO	NO
3	1	1	2	NO	NO	NA	NO	Guest entries regardless of LPID if LPID ≠ 0 in set specified by RB(40:51) Process-scoped host entries with matching LPID if LPID = 0 in set specified by RB(40:51)	All guest PWC regardless of LPID if LPID ≠ 0 Process-scoped host PWC regardless of LPID if LPID = 0	YES - all for that thread only	YES - all process table caching: 1. effPID = PIDR and effLPID = 0 2. effPID = PIDR and effLPID = LPIDR 3. effPID = 0 and effLPID = LPIDR 4. effPID = 0 and effLPID = 0	NO	NO
3	1	1	3									YES	NO

Appendix C. tlbie and tlbiel Encodings for HPT Translations

Table C-1. lists the **tlbie** encodings for HPT translation (with R = '0' and GTSE = '1') and *Table C-2.* on page 490 lists the **tlbiel** encodings for HPT translation (with R = '0').

Note: The yellow rows result in a Machine Check interrupt (invalid form of **tlbie/tlbiel** instruction) and the red rows result in a privileged instruction interrupt.

*Table C-1. **tlbie** Encodings for HPT Translations (with R = '0' and GTSE = '1')* (Sheet 1 of 5)

IS	HV	PRS	RIC	VA Match Required?	LPID Match Required?	LPID Taken From RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	RIC = 3 TLB Invalidations	Invalid Form Machine Check Case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
0	0	0	0	YES	YES	LPIDR	NO	VA match	NO	YES - matching address and LPID (by thread ID)	NO	NA	NO	NO
0	0	0	1										YES	NO
0	0	0	2										YES	NO
0	0	0	3	YES	YES	LPIDR	NO	Series of eight consecutive pages (see column RIC = 3)	NO	YES - matching address and LPID (by thread ID)	NO	AP = 110:8 consecutive 4 KB pages aligned on 32 KB boundary AP = 111:8 consecutive 64 KB pages aligned on 512 KB boundary	NO	NO
0	0	1	0										YES	NO
0	0	1	1										YES	NO
0	0	1	2										YES	NO
0	0	1	3										YES	NO
0	1	0	0	YES	YES	RS	NO	VA match	NO	YES - matching address and LPID (by thread ID)	NO	NA	NO	NO

*Table C-1. **tlbie** Encodings for HPT Translations (with R = '0' and GTSE = '1')* (Sheet 2 of 5)

IS	HV	PRS	RIC	VA Match Required?	LPID Match Required?	LPID Taken From RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	RIC = 3 TLB Invalidations	Invalid Form Machine Check Case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
0	1	0	1										YES	NO
0	1	0	2										YES	NO
0	1	0	3	YES	YES	RS	NO	Series of eight consecutive pages (see column RIC = 3)	NO	YES - matching address and LPID (by thread ID)	NO	AP = 110:8 consecutive 4 KB pages aligned on 32 KB boundary AP = 111:8 consecutive 64 KB pages aligned on 512 KB boundary	NO	NO
0	1	1	0										YES	NO
0	1	1	1										YES	NO
0	1	1	2										YES	NO
0	1	1	3										YES	NO
1	0	0	0										YES	NO
1	0	0	1										YES	NO
1	0	0	2										YES	NO
1	0	0	3										YES	NO
1	0	1	0										YES	NO
1	0	1	1										YES	NO
1	0	1	2	NO	YES	LPIDR	YES	NONE	NO	YES - all with matching LPID and PID (by thread ID)	YES - process table regardless of LPID	NA	NO	NO
1	0	1	3										YES	NO

*Table C-1. **tlbie** Encodings for HPT Translations (with R = '0' and GTSE = '1')* (Sheet 3 of 5)

IS	HV	PRS	RIC	VA Match Required?	LPID Match Required?	LPID Taken From RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	RIC = 3 TLB Invalidations	Invalid Form Machine Check Case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
1	1	0	0										YES	NO
1	1	0	1										YES	NO
1	1	0	2										YES	NO
1	1	0	3										YES	NO
1	1	1	0										YES	NO
1	1	1	1										YES	NO
1	1	1	2	NO	YES	RS	YES	NONE	NO	YES - all with matching LPID and PID (by thread ID)	YES - process table regardless of LPID	NA	NO	NO
1	1	1	3										YES	NO
2	0	0	0	NO	YES	LPIDR	NO	Host entries with matching LPID	NO	YES - all with matching LPID (by thread ID)	NO	NA	NO	NO
2	0	0	1										YES	NO
2	0	0	2										YES	NO
2	0	0	3										YES	NO
2	0	1	0										YES	NO
2	0	1	1										YES	NO
2	0	1	2	NO	YES	LPIDR	NO	NONE	NO	YES - all with matching LPID (by thread ID)	YES - process table regardless of LPID	NA	NO	NO
2	0	1	3										YES	NO

*Table C-1. **tlbie** Encodings for HPT Translations (with R = '0' and GTSE = '1')* (Sheet 4 of 5)

IS	HV	PRS	RIC	VA Match Required?	LPID Match Required?	LPID Taken From RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	RIC = 3 TLB Invalidations	Invalid Form Machine Check Case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
2	1	0	0	NO	YES	RS	NO	Host entries with matching LPID	NO	YES - all with matching LPID (by thread ID)	NO	NA	NO	NO
2	1	0	1								NO		YES	NO
2	1	0	2	NO	YES	RS	NO	Host entries with matching LPID	NO	YES - all with matching LPID (by thread ID)	YES - partition table with matching LPID	NA	NO	NO
2	1	0	3										YES	NO
2	1	1	0										YES	NO
2	1	1	1										YES	NO
2	1	1	2	NO	YES	RS	NO	NONE	NO	YES - all with matching LPID (by thread ID)	YES - process table regardless of LPID	NA	NO	NO
2	1	1	3										YES	NO
3	0	0	0	NO	YES	LPIDR	NO	Host entries with matching LPID	NO	YES - all with matching LPID (by thread ID)	NO	NA	NO	NO
3	0	0	1										YES	NO
3	0	0	2										YES	NO
3	0	0	3										YES	NO
3	0	1	0										YES	NO
3	0	1	1										YES	NO

*Table C-1. **tlbie** Encodings for HPT Translations (with R = '0' and GTSE = '1')* (Sheet 5 of 5)

IS	HV	PRS	RIC	VA Match Required?	LPID Match Required?	LPID Taken From RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invalidated?	ERATs Invalidated?	Process/ Partition Table Caching Invalidated?	RIC = 3 TLB Invalidations	Invalid Form Machine Check Case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
3	0	1	2	NO	YES	LPIDR	NO	NONE	Guest PWC with matching LPID	YES - all with matching LPID (by thread ID)	YES - process table regardless of LPID	NA	NO	NO
3	0	1	3										YES	NO
3	1	0	0	NO	NO	NA	NO	All host entries regardless of LPID	NO	YES - regardless of LPID (thread)	NO	NO	NO	NO
3	1	0	1										YES	NO
3	1	0	2	NO	NO	NA	NO	All host entries regardless of LPID	All partition-scoped host PWC entries regardless of LPID	YES - regardless of LPID (thread)	YES - partition table regardless of LPID	NA	NO	NO
3	1	0	3											NO
3	1	1	0										YES	NO
3	1	1	1										YES	NO
3	1	1	2	NO	NO	NA	NO	All host entries regardless of LPID	All guest PWC regardless of LPID if LPID ≠ 0 Process-scoped host PWC regardless of LPID if LPID = 0	YES - regardless of LPID (thread)	YES - process table regardless of LPID	NA		NO
3	1	1	3										YES	NO

Table C-2. ***tlbiel*** *Encodings for HPT Translation (with R = '0')* (Sheet 1 of 5)

IS	HV	PRS	RIC	VA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invali-dated?	ERATs Invali-dated?	Process/ Partition Table Caching Invalidated?	RIC = 3 TLB Invalidations	Invalid Form Machine Check Case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
0	0	0	0	YES	YES	LPIDR	NO	VA match	NO	YES - matching address for that thread only	NO	NA	NO	NO
0	0	0	1										YES	NO
0	0	0	2										YES	NO
0	0	0	3								NO	NA	YES	NO
0	0	1	0										YES	NO
0	0	1	1										YES	NO
0	0	1	2										YES	NO
0	0	1	3										YES	NO
0	1	0	0	YES	YES	LPIDR	NO	VA match	NO	YES - matching address for that thread only	NO	NA	NO	NO
0	1	0	1										YES	NO
0	1	0	2										YES	NO
0	1	0	3										YES	NO
0	1	1	0										YES	NO
0	1	1	1										YES	NO
0	1	1	2										YES	NO
0	1	1	3										YES	NO
1	0	0	0										YES	NO
1	0	0	1										YES	NO

Table C-2. ***tlbiel*** *Encodings for HPT Translation (with R = '0')* (Sheet 2 of 5)

IS	HV	PRS	RIC	VA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invali-dated?	ERATs Invali-dated?	Process/ Partition Table Caching Invalidated?	RIC = 3 TLB Invalidations	Invalid Form Machine Check Case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
1	0	0	2										YES	NO
1	0	0	3										YES	NO
1	0	1	0										YES	NO
1	0	1	1										YES	NO
1	0	1	2	NO	YES	LPIDR	YES	NONE	NO	YES - all for that thread only	YES - process table (regardless of LPID) for issuing thread	NA	NO	NO
1	0	1	3										YES	NO
1	1	0	0										YES	NO
1	1	0	1										YES	NO
1	1	0	2										YES	NO
1	1	0	3										YES	NO
1	1	1	0										YES	NO
1	1	1	1										YES	NO
1	1	1	2	NO	YES	LPIDR	YES	NONE	NO	YES - all for that thread only	YES - process table (regardless of LPID) for issuing thread	NA	NO	NO
1	1	1	3										YES	NO
2	0	0	0	NO	YES	LPIDR	NO	Host entries with matching LPID in set specified by RB(40:51)	NO	YES - all for that thread only	NO	NA	NO	NO

*Table C-2. **tlbiel** Encodings for HPT Translation (with R = '0')* (Sheet 3 of 5)

IS	HV	PRS	RIC	VA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invali-dated?	ERATs Invali-dated?	Process/ Partition Table Caching Invalidated?	RIC = 3 TLB Invalidations	Invalid Form Machine Check Case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
2	0	0	1										YES	NO
2	0	0	2										YES	NO
2	0	0	3										YES	NO
2	0	1	0										YES	NO
2	0	1	1										YES	NO
2	0	1	2	NO	YES	LPIDR	NO	NONE	NO	YES - all for that thread only	YES - process table (regardless of LPID) for issuing thread	NA	NO	NO
2	0	1	3										YES	NO
2	1	0	0	NO	YES	LPIDR	NO	Host entries with matching LPID in set specified by RB(40:51)	NO	YES - all for that thread only	NO	NA	NO	NO
2	1	0	1								NO		YES	NO
2	1	0	2	NO	YES	LPIDR	NO	Host entries with matching LPID in set specified by RB(40:51)	NO	YES - all for that thread only	YES - partition table for issuing thread	NA	NO	NO
2	1	0	3										YES	NO
2	1	1	0										YES	NO
2	1	1	1										YES	NO

*Table C-2. **tlbiel** Encodings for HPT Translation (with R = '0')* (Sheet 4 of 5)

IS	HV	PRS	RIC	VA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invali-dated?	ERATs Invali-dated?	Process/ Partition Table Caching Invalidated?	RIC = 3 TLB Invalidations	Invalid Form Machine Check Case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
2	1	1	2	NO	YES	LPIDR	NO	NONE	NO	YES - all for that thread only	YES - pro-cess table regardless of LPID for issuing thread	NA	NO	NO
2	1	1	3										YES	NO
3	0	0	0	NO	YES	LPIDR	NO	Host entries with matching LPID in set specified by RB(40:51)	NO	YES - all for that thread only	NO	NA	NO	NO
3	0	0	1										YES	NO
3	0	0	2										YES	NO
3	0	0	3										YES	NO
3	0	1	0										YES	NO
3	0	1	1										YES	NO
3	0	1	2	NO	YES	LPIDR	NO	NONE	Guest PWC with matching LPID	YES - all for that thread only	YES - pro-cess table (regardless of LPID) for issuing thread	NA	NO	NO
3	0	1	3										YES	NO

*Table C-2. **tlbiel** Encodings for HPT Translation (with R = '0')* (Sheet 5 of 5)

IS	HV	PRS	RIC	VA Match Required?	LPID Match Required?	LPID Taken from RS or LPIDR	PID Match Required?	TLB Entry/ Entries Invalidated	PWC Invali-dated?	ERATs Invali-dated?	Process/ Partition Table Caching Invalidated?	RIC = 3 TLB Invalidations	Invalid Form Machine Check Case? (yellow)	Privileged Instruction Interrupt? HV = 0, PRS = 0 (red)
3	1	0	0	NO	NO	NA	NO	All host entries regardless of LPID in set speci-fied by RB(40:51)	NO	YES - all for that thread only	NO	NO	NO	NO
3	1	0	1										YES	NO
3	1	0	2	NO	NO	NA	NO	All host entries regardless of LPID in set speci-fied by RB(40:51)	All parti-tion-scoped host PWC entries regardless of LPID	YES - all for that thread only	YES - parti-tion table regardless of LPID for issueing thread	NA	NO	NO
3	1	0	3											NO
3	1	1	0										YES	NO
3	1	1	1										YES	NO
3	1	1	2	NO	NO	NA	NO	All host entries regardless of LPID in set speci-fied by RB(40:51)	All guest PWC regardless of LPID if LPID ≠ 0 Process-scoped host PWC regardless of LPID if LPID = 0	YES - all for that thread only	YES - pro-cess table (regardless of LPID) for issuing thread	NA		NO
3	1	1	3										YES	NO

Glossary

ABIST	Array built-in self test
AC	Authenticated code
ACAM	Adress content-addressable memory
AES	Advanced Encryption Standard
AIB	ASIC interface bus
ALI	Alignment interrupt
ALU	Arithmetic logic unit
AMC	Architected mapper cache
AMO	Atomic memory operation
ARF	Architected register file
ASIC	Application-specific integrated circuit
ASST	At-speed structure-test
ATS	Address translation services
AVA	Abbreviated vrtual address
BAR	Base Address Register
BCD	Binary coded decimal
BER	Bit error ratio
BFP	Binary floating-point
BFU	Binary floating-point unit
BHT	Branch history table
BIST	Built-in self-test
BMC	Baseboard management control
BR	Branch register unit
BTAC	Branch target address cache
CAM	Content-addressable memory
CAPI	Coherent accelerator processor interface
CBC	Cipher-block chaining
CC	Completion code

CCM	Counter with CBC-MAC
CCS	Configured command sequencer
CEC	Central electronics complex
cgc	Congruence class
CI	Cast-in
CIABR	Current Instruction Address Breakpoint Register
CIR	Chip information register
CIU	Core interface unit
CLB	Cache load buffer (IBuffer)
CME	Core management engine
CMOS	Complementary metal–oxide–semiconductor
CO	Cast-out
CPB	Coprocessor parameter block
CPU	Central processing unit
CR	Condition Register
CRB	Coprocessor request block
CRC	Cyclic redundancy check
CRN	Conditioned random numbers
CT	Coprocessor type
CTLE	Continuous time linear equalizer
darn	Deliver a random number instruction
DARQ	Data and address recirculation queue
DAWR	Data Address Watch Register
dcbt	Data cache block touch
dcbtst	Data cache block touch for store
dcbz	Data cache block zero
dcbst	Data cache block store
dcbst	Data cache block store
dcbf	Data cache block flush

dcbfl	Data cache block flush local
dcbflp	Data cache block flush local primary
DDL	Data descriptor list
DDR	Double data rate
DDR4	Double data rate memory interface, 4th generation
DECFP	Decimal floating-point unit
DFE	Decision feedback equalizer
DFP	Decimal floating-point
DFU	Decimal floating-point unit
DIMM	Dual in-line memory module
DLL	Delay-locked loop
DMA	Direct memory attach
DMI	Differential memory interface
DPC	DIMMs per channel or dynamic peaking control
DPLL	Digital phase-locked loop
DRAM	Dynamic random access memory
DRTM	Dynamic root of trust for measurement
DSI	Data storage interrupt
dss	Data stream stop
dst	Data stream touch
dstst	Data stream touch for store
DTS	Digital thermal sensor
EA	Effective address
EADIR	Effective address directory
EAE	Event assignment entry
EAS	Event assignment structure
EASC	Event assignment structure cache
EAST	Event assignment structure table
EAT	Effective address translation

EAT	Event assignment table
EBB	Event-based branch
EBB	Event-based branch
ECB	Electronic codebook
ECC	Error correcting code
ECID	Electronic chip identification
ECO	Extended cache option
ECRC	End-to-end cyclic redundancy check
EDI	Elastic differential I/O
EDRAM	Enhanced dynamic random access memory
EEH	Enhance error handling
EEPROM	Electrically erasable programmable read-only memory
EMC	Extended memory controller
EMQ	ERAT miss queue
END	Event notification descriptor
ENDC	Event notification descriptor cache
ENDE	Event notification descriptor entry
ENDT	Event notification descriptor table
EOI	End of interrupt
EQ	Event queue
EQD	Event queue descriptor
EQDT	Event queue descriptor table
EQOC	Event queue page offset counter
ERAT	Effective-to-real address translation
ESB	Event state buffer
ESBC	Event state buffer cache
ESID	Effective segment identifier
FBC	SMP interconnect controller
FC	Function code

FFE	Feed-forward equalizer
FIFO	First-in, first-out
FIR	Fault Isolation Register
FLIT	Flow control digit
FLOPs	Floating-point operations per second
FPGA	Field-programmable gate array
FPR	Floating-point register
FPSCR	Floating-Point Status and Control Register
FPU	Floating-point unit
FSP	Flexible service processor
FXU	Fixed-point units
GCM	Galois counter mode
GCT	Global completion table
GFW	Global firmware
GHV	Global history vector
GPE	General purpose engine
GPR	General purpose register
GPS	Global Pstate
GPST	Global Pstates table
GPU	Graphics processor unit
GR	Guest Radix
GTps	Gigatransfers per second
HDEC	Hypervisor decrementer
HDSI	Hypervisor data storage interrupt
hEA	Host effective address
HISI	Hypervisor instruction storage interrupt
HMAC	Hash message authentication code
HMER	Hypervisor Maintenance Exception Register
HMI	Hypervisor maintenance interrupt

HPT	Hashed page table
HR	Host Radix
hRA	Host real address
HRMOR	Hypervisor Real Mode Offset Register
HSS	High-speed serial
HTM	Hardware trace monitor
hVA	Host virtual address
I^2C	Inter-integrated circuit
IBUF	Instruction buffer
ICA	Instruction cache access
icbi	Instruction cache block invalidate
icbt	Instruction cache block touch
isync	Instruction cache synchronize
ICP	Interrupt control presenter
ICS	Interrupt controller source
ICT	Instruction completion table
IEEE	Institute of Electrical and Elctronics Engineers
IFAR	Instruction fetch address register
IFB	Instruction fetch buffer
IFU	Instruction fetch and decode unit
IMA	In memory accumulate
IOO	OpenPOWER interface
IOP	Internal operation
IPB	Interrupt Pending Buffer
IPC	Instruction per cycle
IPL	Interrupt presenter layer; orinitial program load
IRL	Interrupt routing layer
IRR	Instruction retry and recovery
ISU	Instruction sequencing unit

ISV	Independent software vendor
IVE	Interrupt vector entry
IVE	Interrupt Virtualization Entry
IVPE	Interrupt Virtualization Presentation Engine
IVRE	Interrupt Virtualization Routing Engine
iVRM	Internal voltage regulation
IVSE	Interrupt Virtualization Source Engine
IVT	Interrupt Virtualization Table
JEDEC	Joint Electron Device Engineering Council
JTAG	Joint Test Action Group
KVM	Kernal-based virtual machine
LBIST	Logic built-in self test
LCO	Later castout or lateral castout (cast out to another cache rather than memory)
LDS	Load station
LE	Little-endian
LFSR	Linear Feedback Shift Register
LGA	Land grid array
LHR	Load-hit-reload
LMQ	Load-miss queue
LPAR	Logical partition
LPC	Low-pin count
LPID	Logical partition ID
LPIDR	Logical Partition ID Register
LPST	Local Pstate table
LRDIMM	Load-reduced dual in-line memory module
LRQ	Load reorder queue
LRU	Least-recently used
LSAQ	Load store address queue
LSI	Level signaled interrupt

LSMFB	Logical Server Most Favored Backlog
LSSD	Level-sensitive scan design
LSU	Load store units
LTE	Long-tail equalizer
LU	Load-only unit
MBA	Memory buffer asynchronous
MCA	Memory controller asynchronous
MCBIST	Memory card built in self-test
MCD	Memory cache-line domain
MCE	Machine check exception
MCS	Memory controller synchronous
MD5	Message Digest 5
MDI	Memory domain indicator
MDS	Memory domain status
MHCRO	Model hardware correlation ring oscillator
MMIO	Memory-mapped input/output
MMU	Memory nanagement unit
MPG	Multi-protocol gateway
MPSS	Multiple page sizes per segment
MRS	Mode register set
MSI	Message signalled interrupt
NaN	Not a number
NCU	Noncacheable unit
NDL	NVLink Datalink layer
NMMU	Nest memory nanagement unit
NPCQ	NPU common queue
NPS	Nap Pstate
NPU	NVLink processing unit
NTC	Next-to-complete

NVC	Notification virtual descriptor cache
NVT	Notification Virtual Target
NVTS	Notfication Virtual Target Structure
NVTT	Notification Virtual Target Table
OCC	On-chip controller
OCTS	On-chip thermal sensor
OEM	Original equipment manufacturer
OHA	On-chiplet hardware assist
OpenCAPI	Open Coherent Accelerator Processor Interface
ODL	OpenCAPI datalink layer
OTL	OpenCAPI transaction layer
P3CQ	POWER9 fabric bus interface common queue
P3PC	Presentation Controller
P3SC	Source Controller
P3VC	Virtualization Controller
PAPR	Power Architecture Platform Reference
PATB	Partition Table Base field
PATS	Partition Table Size field
PB	Processor bus
PBL	Packet buffer layer
PC	Pervasive core unit
PCIe	Peripheral component interconnect express
PCR	Processor Compatibility Register or Platform Configuration Register
PCS	Physical coding sublayer
PDE	Page directory entry
PE	Partitionable endpoints
PEC	PCI Express controller
PEF	Protected execution facility
PF	Prefetch machine

PFD	Phase-frequency detector
PFWI	Prefetch write inject
PHB	PCI host bridge
PHY	Physical layer
PHYP	Power hypervisor
PID	Process ID
PIDR	Process ID Register
PIPR	Pending Interrupt Priority Register
PLL	Phase-locked loop
PMA	Physical media access
PMC	Power management control
PMU	Performance monitor unit
POR	Power-on reset
PPE	Programmable PowerPC-lite engine
PRI	Private register interface
PRQ	Prefetch request queue
PSI	Processor serial interface
PSPB	Problem-state priority boost
PSRO	Performance sort-ring oscillator
Pstate	Performance state
PTCR	Partition Table Control Register
PTE	Page table entry
PTEG	Page table entry group
PTER	Physical Thread Enable Register
PURR	Processor Utilization Resource Register
PVR	Processor Version Register
PWC	Page-walk cache
QNaN	Quiet Not a number
QoS	Quality of service

qpos	Queue position
RA	Read address
RAIM	Redundant array of independent memory
RAM	Random access memory
RAS	Reliability, availability, and serviceability
RAW	Read after write
RC	Root complex or read claim
RCD	Register clock driver
RCMD	Remote command
RDIMM	Registered dual in-line memory module
RMA	Remote memory access
RMSC	Real mode storage control
RNG	Random number generator
ROB	Re-order buffer
ROT	Rollback-only transaction
RPT	Radix page table
RRN	Raw random numbers
S2Q	Store drain queue
SAM	Store address machine
SAO	Strong access ordering
SAR	Second-level Architected Register
SBE	Self-boot engine
SBE	State bit entry
Sc	Store clean (transactional memory value before a speculative store)
SCM	Single-chip module
SCOM	Scan communications
SDM	Store data machine
SDQ	Store data queue
SEC/DED	Single-error correction, double-error detection

SEEPROM	Serial electrically erasable programmable read-only memory
SERDES	Serializer/Deserializer
SETP	Set prediction directory
SHA	Secure hash algorithm
SHR	Store-hit-reload
SICQ	SMP interconnect common queue
SIMD	Single-instruction, multiple-data
SIU	SMP interconnect unit
SLB	Segment lookaside buffer
SMF	Secure memory facility
SMP	Symmetric multiprocessing
SMPI	SMP interconnect
SMT	Simultaneous multithreading
SN	Snoop machine
SOI	Silicon-on-insulator
SP	Single-precision
SPI	Serial peripheral interconnect
SPIVID	Serial Peripheral Interface - Voltage ID
SPR	Special purpose register
SPS	Sleep Pstate
SPURR	Scaled Processor Utilization Resource Register
SRAM	Static random access memory
SRQ	Store reorder queue
ST	Single thread
STAG	Storage tag
STE	Send Window Table Entry (STE) or segment table entry
STEG	Segment table entry group
SVIC	Slave VME interface controller
SVM	Secure virtual machine

TCE	Translation control entry
TCTXT	Thread context
TDP	Thermal design point
TEXASRU	Transaction Exception And Summary Register Upper
TID	Thread ID
TLB	Translation lookaside buffer
TLBI	translation look-aside buffer invalidate
TLDLP	Transaction and data link layer
TLE	Transaction lock elision
TLP	Translation layer packet
TM	Transactional memory
TOD	Time of day
TPM	Trusted platform module
UE	Uncorrectable error
UI	Unit interval
UILE	Ultravisor Interrupt Little Endian
UMAC	User mode access control
UniQ	Unified issue queues
VAS	Virtual Accelerator Switchboard
VCO	Voltage-controlled oscillator
VID	Voltage identification
VLE	Variable length encoding
VMX	Virtual machine extensions
VPD	Vital product data
VPD	Virtual Processor Descriptor
VPDT	Virtual Processor Descriptor Table
VPN	Virtual page number
VRF	Vector scalar register file
VRM	Voltage regulator module

VRMA	Virtualized real mode area
VS	Vector scalar
VSCR	Vector Status and Control Register
VSD	Virtualization Structure Descriptor
VST	Virtual Structure Table
VSU	Vector and scalar unit
VSX	Vector-scalar extension
WAW	Write after write
WI	Write inject
WOF	Workload optimized frequency
WPS	Winkle Pstate
X bus	An X bus is the socket-to-socket SMP interconnect between 2 POWER9 processors. **Note:** This is not really an acronym but a name (X bus).
XER	Fixed-Point Exception Register
XIVE	External Interrupt Virtualization Engine
XMAC	XCBC-MAC-96
XSL	Adress translation block
XTS	Extended translation services (NVlink protocol over 25G Link address translation services)