IBM POWER8 processor core microarchitecture

Article *in* Ibm Journal of Research and Development · January 2015

DOI: 10.1147/JRD.2014.2376112

CITATIONS
120

READS
10,069

20 authors, including:

Hung Q le
IBM
17 PUBLICATIONS **1,066** CITATIONS
SEE PROFILE

D.Q. Nguyen
IBM
16 PUBLICATIONS **452** CITATIONS
SEE PROFILE

José E. Moreira
IBM
241 PUBLICATIONS **4,754** CITATIONS
SEE PROFILE

D. Levitan
IBM
9 PUBLICATIONS **189** CITATIONS
SEE PROFILE

IBM POWER8 processor core microarchitecture

B. Sinharoy
J. A. Van Norstrand
R. J. Eickemeyer
H. Q. Le
J. Leenstra
D. Q. Nguyen
B. Konigsburg
K. Ward
M. D. Brown
J. E. Moreira
D. Levitan
S. Tung
D. Hrusecky
J. W. Bishop
M. Gschwind
M. Boersma
M. Kroener
M. Kaltenbach
T. Karkhanis
K. M. Fernsler

The POWER8™ processor is the latest RISC (Reduced Instruction Set Computer) microprocessor from IBM. It is fabricated using the company's 22-nm Silicon on Insulator (SOI) technology with 15 layers of metal, and it has been designed to significantly improve both single-thread performance and single-core throughput over its predecessor, the POWER7® processor. The rate of increase in processor frequency enabled by new silicon technology advancements has decreased dramatically in recent generations, as compared to the historic trend. This has caused many processor designs in the industry to show very little improvement in either single-thread or single-core performance, and, instead, larger numbers of cores are primarily pursued in each generation. Going against this industry trend, the POWER8 processor relies on a much improved core and nest microarchitecture to achieve approximately one-and-a-half times the single-thread performance and twice the single-core throughput of the POWER7 processor in several commercial applications. Combined with a 50% increase in the number of cores (from 8 in the POWER7 processor to 12 in the POWER8 processor), the result is a processor that leads the industry in performance for enterprise workloads. This paper describes the core microarchitecture innovations made in the POWER8 processor that resulted in these significant performance benefits.

Introduction

Based on principles adopted in the POWER7* multi-core processor [1–4], the POWER8* processor continues to emphasize a balanced multi-core design, with significant improvements in both single-thread and core performance and modest increases in the core count per chip. This contrasts with other multi-core processor designs in the industry today, for which an increase in the core count is primarily pursued with little improvement in single-thread or core performance. In this eighth-generation POWER* processor, IBM continues to innovate its RISC (Reduced Instruction Set Computer) product line by introducing a twelve-core multi-chip design, with large on-chip eDRAM (embedded Dynamic Random Access Memory) caches, and high-performance eight-way multi-threaded cores, implementing the Power ISA (Instruction Set Architecture) version 2.07 [5].

Our goal for the POWER8 processor was to significantly improve the socket-level, core-level and thread-level performance in each of the multiple simultaneous multithreading (SMT) modes relative to the POWER7 processor. This was achieved by keeping the area and power requirement of each POWER8 processor core ("POWER8 core") sufficiently low to allow twelve such cores on the processor chip while maintaining its power at the same level as that of the POWER7 processor chip. An "at-a-glance" comparison between the POWER7 and the POWER8 processors can be seen in **Table 1**.

Because of the slowdown in frequency increases from silicon technology, thread and core performance were improved through microarchitectural enhancements such as an advanced branch prediction mechanism; extensive out-of-order execution; dual pipelines for instruction decode, dispatch, issue, and execution; advanced eight-way simultaneous multi-threading; advanced prefetching with more precise application software control over the prefetching mechanism; doubled bandwidth throughout the cache and memory hierarchy; and a significant

Digital Object Identifier: 10.1147/JRD.2014.2376112

Table 1 Summary of characteristics of the POWER7 and POWER8 processors.

	POWER7	*POWER8*
Cores/chip	8	12
Maximum threads/core	4	8
L1 instruction cache/core	32 KB	32 KB
L1 data cache/core	32 KB	64 KB
L2 cache/core	256 KB	512 KB
L3 cache/core	4 MB	8 MB
Instruction issue/cycle/core	8	10
Instruction completion/cycle/core	6	8

reduction in memory latency relative to the POWER7 processor design.

We enhanced the POWER8 core microarchitecture with support for fast access to unaligned and little-endian data. This simplifies application development and porting of applications across different processor architectures. It also facilitates data exchange between devices of different architectures, including system-level accelerators such as GPUs (graphical processing units).

The POWER8 core also supports several new differentiating features such as advanced security, dynamic compiler optimizations enablement, hybrid computation, cryptography acceleration, advanced SIMD (Single-Instruction, Multiple-Data) features, and business analytics optimizations enablement, among others.

The POWER* processor line [6–9] has continuously delivered high performance and reliability for commercial workloads and business applications. In addition to its performance in traditional commercial workloads, the POWER8 processor is highly optimized for cloud-based workloads, as well as big data, business analytics, and systems of engagement applications. These emerging workloads and applications have their own performance requirements and challenges, and we had to provide corresponding innovations in the POWER8 core.

Business analytics applications are increasingly important in the enterprise and are both data and compute intensive. For that reason, the POWER8 core delivers improved SIMD performance, through symmetric vector pipelines, doubled data buses, and larger caches. These same enhancements also improve the performance of the POWER8 core on scientific and technical computing, also known as HPC (high-performance computing). Similar to the POWER7+* processor core [10], a POWER8 core has a peak computational throughput of 8 double-precision (64-bit) and 16 single-precision (32-bit) floating-point operations per cycle (four double-precision and eight single-precision fused multiply-adds per cycle, respectively). We also expanded the repertoire of SIMD integer instructions in the POWER8 processor to include 64-bit integer operations and other fixed-point operations such as 32-bit vector multiply instructions. These instructions significantly improve the performance of data warehouse and analytics applications.

Many business analytics applications run in thread-rich configurations, to exploit the inherent parallelism in these computations. To accommodate them, the POWER8 core doubled the hardware thread parallelism to 8-way multithreading (referred to as SMT8). Because of the doubling in size of the L1 data cache and L2 and L3 caches, each thread in a POWER8 core can have as much resident memory footprint as a thread in a POWER7 core. In fact, it was a design requirement that at each common multithreading level—ST (single-thread), SMT2 (two-way multithreading), and SMT4 (four-way multithreading)—the individual thread performance on a POWER8 core should be better than on a POWER7 core. In single-thread mode, practically all of the core resources can be used by the single thread. At the same time, these core resources can efficiently support eight threads per core. The core can dynamically change mode among ST, SMT2, SMT4, and SMT8, depending on the number of active threads.

Cloud instances often do not have enough simultaneously active application threads to utilize all eight hardware threads on the POWER8 core. To instead exploit the parallelism across cloud instances, the POWER8 core can be put in a "split-core mode," so that four partitions can run on one core at the same time, with up to two hardware threads per partition.

Modern computing environments, and cloud systems in particular, require extra layers of security and protection in order to deliver a safe and usable solution to the end user. For that reason, the POWER8 processor includes several features that accelerate cryptographic codes. In particular, the POWER8 core includes a cryptographic unit supporting new Power ISA instructions for the computation of AES (Advanced Encryption Standard), SHA (Secure Hash Algorithm), and CRC (Cyclic Redundancy Check) codes.

Big data applications typically have a larger memory footprint and working set than traditional commercial applications. Correspondingly, compared to the POWER7 core, the POWER8 core has an L1 data cache that is twice as large, has twice as many ports from that data cache for higher read/write throughput, and has four times as many entries in its TLB (Translation Lookaside Buffer). In addition, POWER8 technology expands the addressing range of memory accesses using fusion to allow applications to access large data sets more quickly. As mentioned, the L2 and L3 caches in the POWER8 processor are also twice the size of the corresponding POWER7 processor caches, on a per core basis.

Dynamic scripting languages, such as Ruby, Python, and JavaScript**, are often dominant in the new systems of engagement being deployed. The POWER8 core improves

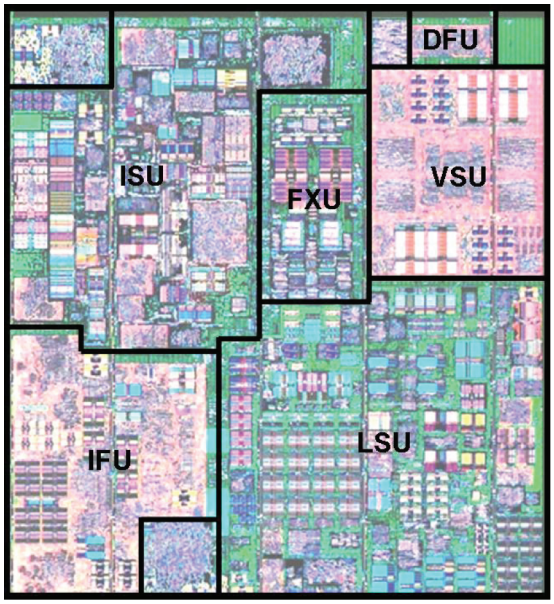

Figure 1

POWER8 processor core floorplan.

on the performance of the POWER7 core for these workloads through better branch prediction mechanisms, increased instruction level parallelism, efficient unaligned data access, and support for run-time monitoring and optimization. These microarchitectural improvements of the POWER8 core have resulted in significant single-thread and throughput gains over the POWER7 core.

Also supporting emerging workloads, the POWER8 processor includes an optimized implementation of hardware TM (Transactional Memory). This implementation has a low overhead to start a transaction and additional features that support the exploitation of transactions in Java and other programming languages, in many cases without any changes to the user code. For more information on TM, please see [11].

Organization of the POWER8 processor core

Figure 1 shows the POWER8 core floorplan. The core consists primarily of the following six units: instruction fetch unit (IFU), instruction sequencing unit (ISU), load-store unit (LSU), fixed-point unit (FXU), vector and scalar unit (VSU) and decimal floating point unit (DFU). The instruction fetch unit contains a 32 KB I-cache (instruction cache) and the load-store unit contains a 64 KB D-cache (data cache), which are both backed up by a tightly integrated 512 KB unified L2 cache.

In a given cycle, the core can fetch up to eight instructions, decode and dispatch up to eight instructions, issue and execute up to ten instructions, and commit up to eight instructions. There are sixteen execution pipelines within the core: two fixed-point pipelines, two load/store pipelines, two load pipelines, four double-precision floating-point pipelines (which can also act as eight single-precision floating-point pipelines), two fully symmetric vector pipelines that execute instructions from both the VMX (Vector eXtensions) and VSX (Vector-Scalar eXtensions) instruction categories in the Power ISA, one cryptographic pipeline, one branch execution pipeline, one condition register logical pipeline, and one decimal floating-point pipeline. The two load/store pipes and the two load pipes have the additional capability to execute simple fixed-point operations. The four floating-point pipelines are each capable of executing double precision multiply-add operations, accounting for eight double-precision, 16 single-precision, floating-point operations per cycle per core. In addition, these pipelines can also execute 64-bit integer SIMD operations. The decimal floating point unit, first introduced in the POWER6* processor [12], accelerates many commercial applications.

To satisfy the high bandwidth requirement of many commercial, big data, and HPC workloads, the POWER8 core has significantly higher load/store bandwidth capability compared to its predecessor. While the POWER7 processor can perform two load/store operations in a given cycle, the POWER8 processor can perform two load operations in the load pipes, in addition to two load or store operations in the load/store pipes in a given cycle.

For advanced virtualization capability, the POWER8 processor can be run in POWER7 or POWER6 compatibility modes. When put in these past processor modes, the POWER8 core will only recognize non-privileged instructions that were available in the past machines and generate an illegal instruction interrupt if newer instructions are attempted to be executed. This allows dynamic partition mobility between POWER6, POWER7, and POWER8 processor based systems. As was the case with the POWER7 processor, the large TLB of the POWER8 processor is not required to be invalidated on a partition swap. Instead, the TLB entries can persist across partition swapping, so that if a partition is swapped back again, some of its translation entries are likely to be found in the TLB. Additionally, the POWER8 processor introduces a "partition prefetch" capability, which restores the cache state when a partition is swapped back into a processor core. This feature is described in more detail in [13].

The POWER8 processor allows dynamic SMT mode switches among the various ST and SMT modes. The core supports the execution of up to eight hardware architected threads, named T0 through T7. Unlike the POWER7 core, where the ST mode required the thread to run on the T0 position, in the POWER8 core the single thread can run anywhere from T0 to T7. As long as it is the only thread

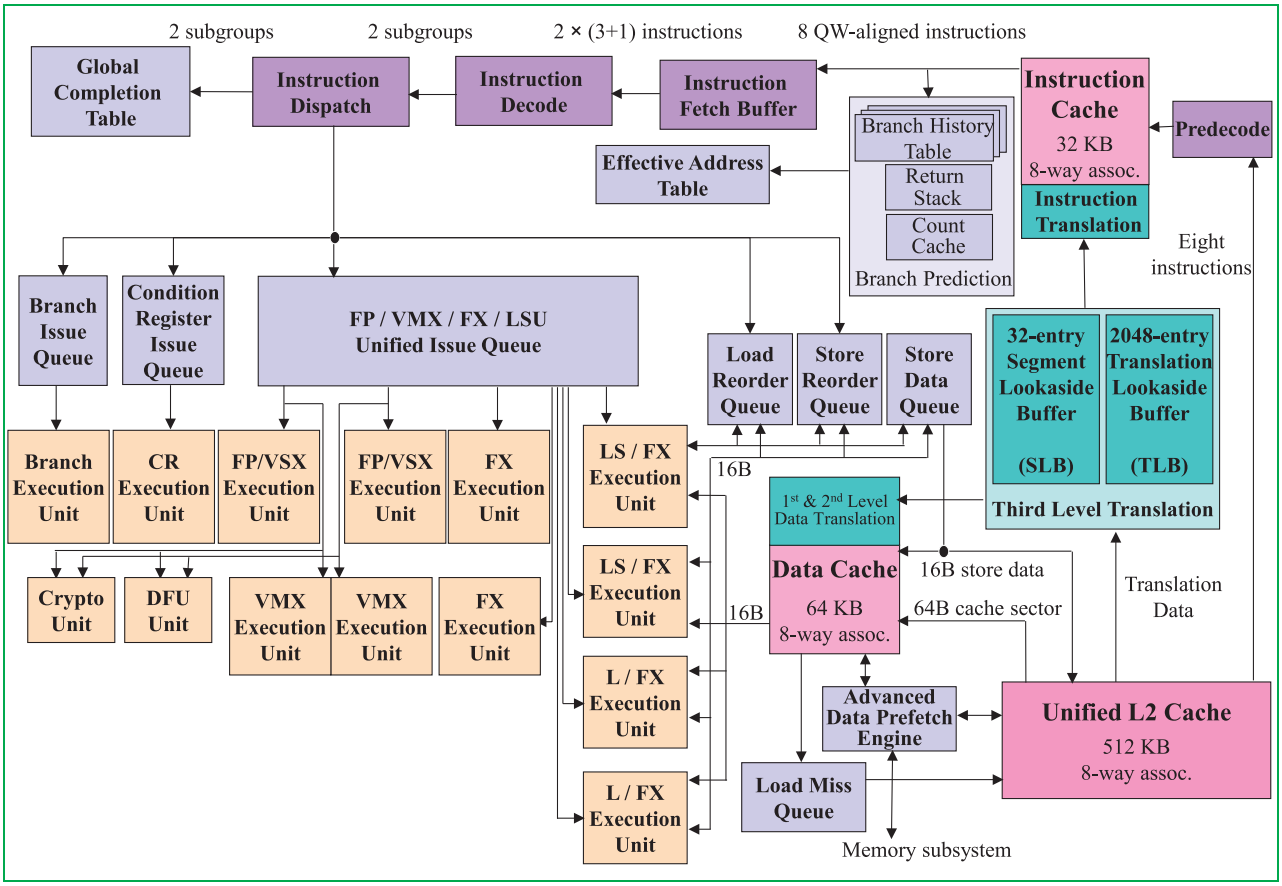

Figure 2

POWER8 processor core pipeline flow. QW-aligned refers to a quadword or 16-byte aligned address.

running, the core can execute in ST mode. Similarly, as long as only two threads are running, the core can execute in SMT2 mode, and it does not matter which hardware thread positions those two threads are running. This makes the SMT mode switch in the POWER8 core significantly easier and does not require software to invoke an expensive thread move operation to put the thread(s) in the right position to switch into the desired SMT mode. In addition, the performance difference of running one single thread on the core when the core is in ST mode versus in any of the SMT modes is significantly lower in the POWER8 processor than in the POWER7 processor.

The POWER8 processor implements robust RAS (reliability, availability, and serviceability) features. It can detect most soft-errors that occur during instruction execution. On soft-error detection, the core automatically uses its out-of-order execution features to flush the instructions in the pipeline and re-fetch and re-execute them, so that there is no loss of data integrity.

Figure 2 shows the instruction flow in POWER8 processor core. Instructions flow from the memory hierarchy through various issue queues and then are sent to the functional units for execution. Most instructions (except for branches and condition register logical instructions) are processed through the Unified Issue Queue (UniQueue), which consists of two symmetric halves (UQ0 and UQ1). There are also two copies (not shown) of the general-purpose (GPR0 and GPR1) and vector-scalar (VSR0 and VSR1) physical register files. One copy is used by instructions processed through UQ0 while the other copy is for instructions processed through UQ1. The fixed-point, floating-point, vector, load and load-store pipelines are similarly split into two sets (FX0, FP0, VSX0, VMX0, L0, LS0 in one set, and FX1, FP1, VSX1, VMX1, L1, LS1 in the other set) and each set is associated with one UniQueue half.

Which issue queue, physical register file, and functional unit are used by a given instruction depends on the simultaneous multi-threading mode of the processor core at run time. In ST mode, the two physical copies of the GPR and VSR have identical contents. Instructions from the thread can be dispatched to either one of the UniQueue halves (UQ0 or UQ1). Load balance across the two

UniQueue halves is maintained by dispatching alternate instructions of a given type to alternating UniQueue halves.

In the SMT modes (SMT2, SMT4, SMT8), the two copies of the GPR and VSR have different contents. The threads are split into two *thread sets* and each thread set is restricted to using only one UniQueue half and associated registers and execution pipelines. Fixed-point, floating-point, vector and load/store instructions from even threads (T0, T2, T4, T6) can only be placed in UQ0, can only access GPR0 and VSR0, and can only be issued to FX0, LS0, L0, FP0, VSX0, and VMX0 pipelines. Fixed-point, floating-point, vector and load/store instructions from odd threads (T1, T3, T5, T7) can only be placed in UQ1, can only access GPR1 and VSR1, and can only be issued to FX1, LS1, L1, FP1, VSX1, and VMX1 pipelines. Cryptographic and decimal floating-point instructions from a thread can only be placed in the corresponding UniQueue half, but since there is only one instance of each of these units, all instructions are issued to the same unit.

Branches and condition register logical instructions have their own dedicated issue queues and execution pipelines, which are shared by all threads.

Instruction Fetch Unit

The Instruction Fetch Unit (IFU) in the POWER8 processor (POWER8 IFU) is responsible for feeding the rest of the instruction pipeline with the most likely stream of instructions from each active hardware thread. It uses branch prediction mechanisms, described below, to produce this stream well ahead of the point of execution of the latest committed instruction. The IFU is also responsible for maintaining a balance of instruction execution rates from the active threads using software-specified thread priorities, decoding and forming groups of instructions for the rest of the instruction pipeline, and executing branch instructions. The normal flow of instructions through the IFU includes six fetch and five decode pipeline stages, as shown in **Figure 3**. (The last fetch and first decode stages overlap.)

The POWER8 IFU has several new features relative to the POWER7 processor IFU. Support for SMT8 and additional concurrent LPARs (logical partitions) required changes in sizes for many resources in the IFU. In addition, the following changes were made to improve the overall performance of the POWER8 core: First, instruction cache alignment improvements result in a higher average number of instructions fetched per fetch operation. Second, branch prediction mechanism improvements result in more accurate target and direction predictions. Third, group formation improvements allow more instructions per dispatch group, on average. Fourth, instruction address translation hit rates were improved. Fifth, instruction fusion is used to improve performance of certain common instruction sequences. Finally, better pipeline hazard avoidance mechanisms reduce pipeline flushes. These improvements are describes in detail in the following sections.

Instruction fetching and predecoding

The POWER8 core has a dedicated 32 KB, 8-way set associative L1 I-cache. It is based on a 16-way banked design to avoid read and write collisions. A 32 $\times$ 8-entry Instruction Effective Address Directory (IEAD) provides fast prediction for way selection to choose one fetch line from the eight ways. A traditional full I-cache directory (I-dir) is accessed in parallel to confirm the way selection prediction in the next cycle. The I-cache can be addressed on any 16-byte boundary within the 128-byte cache line.

Fast instruction address translation for instruction fetch is supported by a fully associative 64-entry Instruction Effective to Real Address translation Table (IERAT). The IERAT is shared among all threads. The IERAT directly supports 4 KB, 64 KB, and 16 MB page sizes. Other page sizes are supported by storing entries with the next smaller supported page size.

The IFU reads instructions into the I-cache from the L2 unified cache. Each read request for instructions from the L2 returns four sectors of 32 bytes each. These reads are either demand loads that result from I-cache misses or instruction prefetches. For each demand load request, the prefetch engine initiates additional prefetches for sequential cache lines following the demand load. Demand and prefetch requests are made for all instruction threads independently, and instructions may return in any order, including interleaving of sectors for different cache lines. Up to eight instruction read requests can be outstanding from the core to the L2 cache. Instruction prefetching is supported in ST, SMT2, and SMT4 modes only. Up to three sequential lines are prefetched in ST mode and one sequential line per thread in SMT2 and SMT4 modes. There is no instruction prefetching in SMT8 mode to save on memory bandwidth. Prefetches are not guaranteed to be fetched and depending on the congestion in the POWER8 processor nest, some prefetches may be dropped.

When instructions are read from the L2 cache, the IFU uses two cycles to create predecode and parity bits for each of the instructions, before they are written into the I-cache. The predecode bits are used to scan for taken branches, help group formation, and denote several exception cases. Branch instructions are modified in these stages to help generate target addresses during the branch scan process that happens during the instruction fetch stages of the pipeline. The modified branch instruction, with a partially computed target address, is stored in the I-cache. Three cycles after a 32-byte sector of instructions arrives on the I-cache/L2 interface, the sector is written into the I-cache. If the requesting thread is waiting for these instructions, they are bypassed around the I-cache to be delivered to the instruction buffers and the branch scan logic.

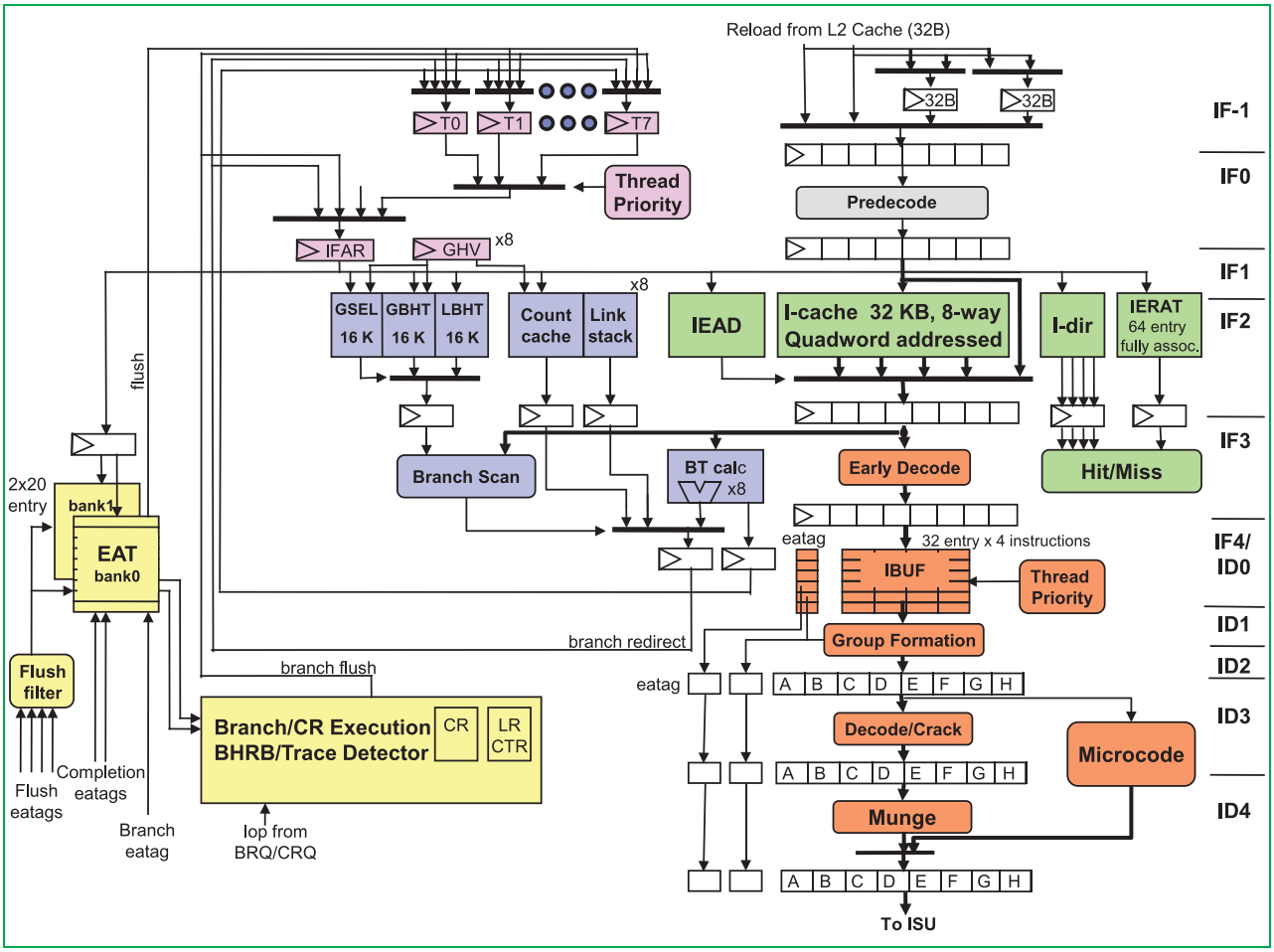

Figure 3

POWER8 instruction fetch unit logical flow. The labels on the right of the figure denote the instruction fetch (IF) and instruction decode (ID) stages. (EAT: effective address table, eatag: effective address tag; iop: internal operation.)

Instruction Fetch Address Registers (IFARs) track program counter addresses for each thread. On each cycle, the IFAR register for one of the threads is selected to provide the fetch address to the I-cache complex and the branch prediction arrays. The I-cache fetch process reads quad-word aligned block of up to eight instructions per cycle from the I-cache and writes them into the instruction buffers where they are later formed into dispatch groups. Quadword-aligned fetch ensures that for a non-sequential fetch at least one instruction from the first quadword and four instructions from the second quadword are fetched as long as there is a cache hit and both quadwords are within the cache line. Thread priority, pending cache misses, instruction buffer fullness, and thread balancing metrics are used to determine which thread is selected for instruction fetching in a given cycle.

The IFU allocates fetch cycles within threads of the same partition based on the priorities associated with each thread. The POWER8 IFU includes a new Relative Priority Register (RPR) allowing the software to optimize the weightings of each of the thread priorities for optimal performance. Thread priority is based on the Relative Priority Register and partition fairness. If a thread does not have space for new instructions in its instruction buffer, another thread can be chosen so no cycles are wasted. The RPR value determines the relative number of cycles that will be allocated to each thread. When there are multiple partitions running on the same core (as in the "split core mode" discussed in the Introduction) the fetch cycles are divided equally between the partitions. If one of the partitions does not have any threads that are ready to fetch, its fetch cycles are relinquished to the next partition that has threads that are ready to fetch.

Group formation

Fetched instructions are processed by the branch scan logic and are also stored in the instruction buffers (IBUF) for

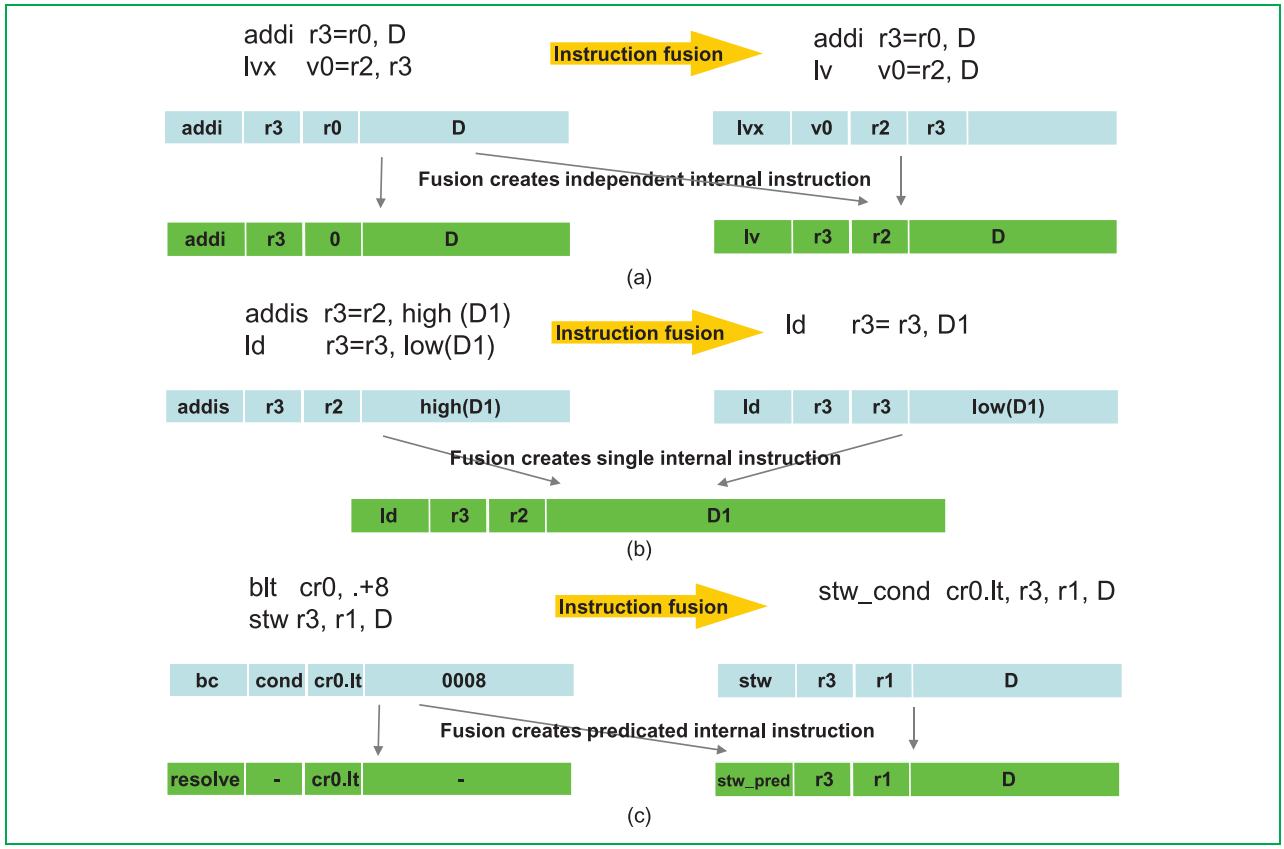

Figure 4

Instruction fusion in the POWER8 processor can be used for a variety of purposes that improve performance. (a) Two dependent instructions are transformed into two independent internal operations. (b) Two dependent instructions are transformed into a single internal operation. (c) A branch is transformed into predicated execution.

group formation. The IBUF can hold up to 32 entries, each four instructions wide. Each thread can have four entries in SMT8 mode, eight entries in SMT4 mode and 16 entries in SMT2 and ST modes. Instructions are retrieved from the IBUF and collected into groups. Thread priority logic selects one group of up to six non-branch and two branch instructions in ST mode or two groups (from two different threads) of up to three non-branch and one branch instructions in SMT modes per cycle for group formation.

Instruction decode

After group formation, the instructions are either decoded or routed to microcode hardware that breaks complex instructions into a series of simple internal operations. Simple instructions are decoded and sent to dispatch. Complex instructions that can be handled by two or three simple internal operations are cracked into multiple dispatch slots. Complex instructions requiring more than three simple internal operations are handled in the microcode engine using a series of simple internal operations. Microcode handling continues until the architected instruction is fully emulated. The decode and dispatch section of the IFU also handles illegal special-purpose register (SPR) detection, creation of execution route bits and marking of instructions for debugging and performance monitoring purposes.

Instruction fusion

For select combinations of instructions, the POWER8 core is capable of fusing two adjacent architected instructions into a single internal operation. Programs exploiting new features introduced via fusion execute correctly on older processors, thus enabling compatibility across the POWER processor line while exploiting the latest POWER8 features. In addition, fusion-based sequences can specify operations that cannot traditionally be encoded in the POWER 32-bit fixed-width RISC ISA. **Figure 4** shows examples of the three different types of fusion.

One class of fusable sequences accelerates address computations for data accesses: the instruction fetch unit decodes a sequence of add-immediate instructions together with certain load instructions and optimizes them. One

such sequence combines add immediate instructions used to load a displacement with register indexed load instructions to create register plus displacement internal instructions which allow the load instructions to execute in parallel with the add immediate instructions [Figure 4(a)]. Another form of fusion directed at address calculation merges add-immediate-shifted and load with 16-bit displacement into a single internal load instruction with a wider displacement. On POWER processors, this sequence is commonly used by software to extend the addressing range of memory access instructions beyond the 16-bit displacement that can be specified with register plus displacement address mode instructions. This optimization supports faster processing of load operations with displacements up to ± 1 MB (i.e., 21 bits), caused by the growing data sets of traditional and emerging applications [Figure 4(b)].

A second class of fusion is for conditional branches skipping over a single instruction. When a conditional branch is followed by certain fixed-point or store instructions, the second instruction can be converted into a predicated operation to eliminate branch processing, including any possible mispredictions [Figure 4(c)].

Branch prediction

The POWER8 core uses separate mechanisms to predict the branch direction (that is, whether a conditional branch is predicted to be taken, or not taken) and the branch target address. In the Power ISA, the target address for a conventional branch instruction can be computed from the instruction encoding and its address, as the displacement is an immediate field in the instruction. The target address of a branch-to-link or branch-to-count instruction is architecturally available in the link or count register. Since the link or count register can be updated anytime before the execution of these instructions, the target address of these branches cannot be computed ahead of time and hence it needs to be predicted to enable instruction fetch to stay well ahead of the point of execution.

The POWER8 IFU supports a three-cycle branch scan mechanism that fetches 32 bytes (corresponding to eight instructions) from the I-cache, scans the fetched instructions for branches that are predicted taken, computes their target addresses (or predicts the target address for a branch-to-link or branch-to-count instruction), determines if any of these branches (in the path of execution) is unconditional or predicted taken and if so, makes the target address of the first such branch available for next fetch for the thread.

It takes three cycles to obtain the next fetch address when there is a taken branch, and for two of these cycles there is no fetch for the thread. However, in SMT mode, those two cycles will normally be allocated to other active threads, and thus not lost. If the fetched instructions do not contain any branch that is unconditional or predicted taken, the next sequential address is used for the next fetch for that thread and no fetch cycles are lost.

The direction of a conditional branch is predicted using a complex of Branch History Tables (BHT), consisting of a 16K-entry local BHT array (LBHT), a 16K-entry global BHT array (GBHT) and a 16K-entry global selection array (GSEL). These arrays are shared by all active threads and provide branch direction predictions for all the instructions in a fetch sector in each cycle. A fetch sector can have up to eight instructions, all of which can be branches. The LBHT is directly indexed by 14 bits from the instruction fetch address. The GBHT and GSEL arrays are indexed by the instruction fetch address hashed with a 21-bit Global History Vector (GHV) folded down to 11 bits. The value in the GSEL entry is used to choose between the LBHT and GBHT, for the direction prediction of each individual branch. All BHT entries consist of two bits with the higher order bit determining direction (taken or not taken), and the lower order bit providing hysteresis. There is one GHV for every thread in the POWER8 core to track the past branch history for that particular thread.

If the effect of a conditional branch is only to conditionally skip over a subsequent fixed-point (FX) or load/store (LS) instruction (called a "bc+8" branch) and the branch is highly unpredictable, the POWER8 processor can often detect such a branch, remove it from the instruction pipeline and conditionally execute the FX/LS instruction. The conditional branch is converted to a "resolve" internal operation (iop) and the subsequent FX/LS instruction is made dependent on the resolve iop. When the condition is resolved, depending on the taken/not-taken determination of the condition, the FX/LS instruction is either executed or ignored. This may cause a delayed issue of the FX/LS instruction, but it prevents a potential pipeline flush due to a mispredicted branch.

Branch target addresses are predicted using two distinct mechanisms with a selection process to determine which mechanism should be used to make the final target prediction. Indirect branches that are not subroutine returns are predicted using a 256-entry local count cache and a 512-entry global count cache, shared by all active threads. The local count cache is indexed using eight bits from the instruction fetch address. The global count cache is indexed using an address obtained by doing an XOR of nine bits each from the instruction fetch address and the GHV. Each entry in the global count cache contains a 30-bit predicted address along with two confidence bits. The upper 32 address bits are assumed to match the current fetch address. Each entry in the local count cache contains a 62-bit predicted address along with a confidence bit. The confidence bits are used to determine when an entry should be replaced if an indirect branch prediction is incorrect. In addition, there is a 2-bit saturating counter in each entry of the local count cache which is used to determine which of the two count cache should be used for prediction. This 2-bit counter is

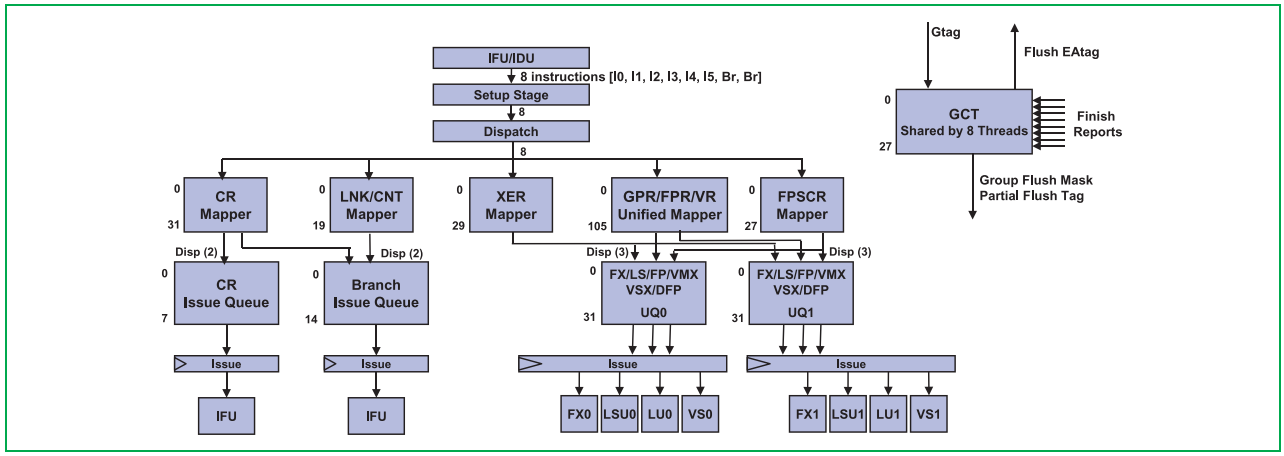

Figure 5

POWER8 instruction sequencing unit (logical flow).

incremented or decremented, based on which of the two mechanisms is successful in making the correct prediction.

Subroutine returns are predicted using a link stack, one per thread. Whenever a branch-and-link (BL) instruction is found in the path of fetched instructions by the branch scanning logic, the address of the next instruction is "pushed down" in the link stack for that thread. The link stack is "popped" whenever a branch-to-link (BLR) instruction is scanned. Entries are pushed and popped speculatively, as the instructions fetched depend on the previously predicted execution path, which is possibly wrong. The POWER8 link stack contains a graph of subroutine addresses. At any given time, only one path linking the root node and a leaf node is the valid link stack; other paths must be stored because control flow that created the multiple paths is not resolved yet. Because of this method, the POWER8 link stack supports subroutine return predictions for programs with complex nested subroutine call-return structures. In ST and SMT2 modes, each thread uses a 32-entry link stack. In SMT4 mode, each thread uses a 16-entry link stack. In SMT8, each thread uses an 8-entry link stack.

Pipeline hazards

The POWER8 IFU also implements mechanisms to mitigate performance degradation associated with pipeline hazards. A Store-Hit-Load (SHL) is an out-of-order pipeline hazard condition, where an older store executes after a younger overlapping load, thus signaling that the load received stale data. The POWER8 IFU has logic to detect when this condition exists and provide control to avoid the hazard by flushing the load instruction which received stale data (and any following instructions). When a load is flushed due to detection of a SHL, the fetch address of the load is saved and the load is marked on subsequent fetches allowing the downstream logic to prevent the hazard. When a marked load instruction is observed, the downstream logic introduces an explicit register dependency for the load to ensure that it is issued after the store operation.

Dynamic configuration of IFU mechanisms

The POWER8 processor also added a Workload Optimization Register for Thread control (WORT) to allow dynamic control over some of the IFU mechanisms. The mechanisms can be used to specifically tune the microarchitecture for a given workload. While these controls can generally provide improved performance for a specific workload, they can also degrade the performance for other workloads. WORT control includes the use of the global predictors in branch prediction, I-cache prefetch, instruction speculation, bc+8 conversion to predication, SHL avoidance and instruction group ending on backward branch.

Instruction Sequencing Unit

The Instruction Sequencing Unit (ISU) dispatches instructions to the various issue queues, renames registers in support of out-of-order execution, issues instructions from the various issues queues to the execution pipelines, completes executing instructions, and handles exception conditions. **Figure 5** illustrates the logical flow of instructions in the ISU.

The POWER8 processor dispatches instructions on a group basis. In ST mode, it can dispatch a group of up to eight instructions per cycle. In SMT mode, it can dispatch two groups per cycle from two different threads and each group can have up to four instructions. All resouces such as the renaming registers and various queue entries must be available for the instructions in a group before the group can be dispatched. Otherwise, the group will be held at the dispatch stage. An instruction group to be dispatched can

have at most two branch and six non-branch instructions from the same thread in ST mode. If there is a second branch, it will be the last instruction in the group. In SMT mode, each dispatch group can have at most one branch and three non-branch instructions.

The ISU employs a Global Completion Table (GCT) to track all in-flight instructions after dispatch. The GCT has 28 entries that are dynamically shared by all active threads. In ST mode, each GCT entry corresponds to one group of instructions. In SMT modes, each GCT entry can contain up to two dispatch groups, both from the same thread. This allows the GCT to track a maximum of 224 in-flight instructions after dispatch.

Each GCT entry contains finish bits for each instruction in the group. At dispatch, the finish bits are set to reflect the valid instructions. Instructions are issued out of order and executed speculatively. When an instruction has executed successfully (without a reject), it is marked as "finished." When all the instructions in a group are marked "finished," and the group is the oldest for a given thread, the group can "complete." When a group completes, the results of all its instructions are made architecturally visible and the resources held by its instructions are released. In SMT modes, the POWER8 core can complete one group per thread set per cycle, for a maximum total of two group completions per cycle. In ST mode, only one group, consisting of up to eight instructions, can complete per cycle. When a group is completed, a completion group tag (GTAG) is broadcast so that resources associated with the completing group can be released and reused by new instructions.

Flush generation for the core is also handled by the ISU. There are many reasons to flush speculative instructions from the instruction pipeline such as branch misprediction, load/store out-of-order execution hazard detection, execution of a context synchronizing instruction, and exception conditions. The GCT combines flushes for all groups to be discarded into a 28-bit mask, one bit for each group. The GCT also sends out the GTAG for partial-group flushes, which occurs when the first branch is not the last instruction in the group, and it mispredicts, causing a need to flush all subsequent instructions from the thread. A 6-bit slot mask accompanies the partial flush GTAG to point out which instructions in the group need to be partially flushed. In addition to discarding some of the operations in the partially flushed group, all the operations in all the flushed groups are also discarded.

Register renaming is performed using the Mapper logic before the instructions are placed in their respective issue queues. The following Power ISA registers are renamed in the POWER8 core: GPR (general-purpose registers), VSR (vector-and-scalar registers), XER (fixed-point exception register), CR (condition register), FPSCR (floating-point status and control register), LR (link register), CTR (count register), and TAR (target address register, new in Power ISA 2.07).

In all single thread and SMT modes, a total of 106 rename (non-architected) states are available across all the threads for GPR and VSR. Two levels of register files are used in POWER8 design to track all the renamed and architected states, as well as checkpointing the GPR/VSR architected states for transactional memory. The first level, GPR and VSR register files, contain all the renames and some of the architected states. The second level, Software Architected Registers (SAR), can contain the entire GPR and VSR architected states, along with the checkpointed architected states due to transactional memory.

GPR registers are mapped onto a pair of 124-entry GPR register files. In ST mode, the two register files have the same contents and 92 entries are available for renaming, while 32 entries are used for architected states. In SMT2 mode, each thread uses one GPR register file and 92 renames and 32 architected registers are available to each thread from its GPR register file. In SMT4 mode, each GPR register file supports two threads with 60 available for rename and 64 available for architected states. In SMT8 mode, each GPR register file supports four threads with 60 available for rename and 64 available for architected states. Since 64 is not enough to store the GPR archiected states for the four threads, SAR contains the entirety of the GPR architected states.

VSR registers are mapped onto a pair of 144-entry VSR register files. In ST mode the two register files have the same contents and 80 entries are available for renaming, while 64 entries are used for architected states. In SMT2 mode, each thread uses a separate VSR register file and 80 renames and 64 archtiected registers are available to each thread from its VSR register files. In SMT4 and SMT8 modes, each VSR register file supports half the threads with 60 available for rename and 64 available for architected states, while SAR contains the entirety of the VSR architected states.

There are two sets of SAR register files, one for GPR and the other for VSR. GPR/VSR SAR contains castouts of the architected states from the GPR/VSR register files. The GPR SAR register file has 72 entries per thread: 32 architected GPR, 4 eGPR that are used for micro-coded instructions and another 32+4 for checkpointing the architected states due to Transactional Memory. The VSR SAR register file has 128 entries per thread: 64 architected VSR, and another 64 for checkpointing the architected states due to Transactional Memory.

As mentioned earlier, the total number of GPR and VSR rename states is limited to 106 in all modes. Renames are available only in the GPR or VSR register files, while the architected states can be found either in the GPR/VSR register files or in their corresponding SAR register files. ISU maintains a tracking mechanism to determine where a given register's architected state can be found in a given cycle.

The CR registers (one per thread) are mapped onto a 32-entry rename mapper plus a 64-entry Architected Register File for the complete architected state (eight CR fields per

thread). The XER register is broken into four separately renamed fields, and one non-renamed field per thread. The renamed fields of the XER registers are mapped onto a 30-entry rename mapper plus 32-entry Architected Register File for the architected state of the (up to) eight threads. The LR, CTR and TAR registers are mapped onto a 20-entry rename mapper plus 24-entry Architected Register File (one LR, one CTR, and one TAR for each of eight threads). The FPSCR is renamed using a 28-entry buffer to keep the state of the FPSCR associated with the instructions in the corresponding entry in the GCT. Each of the above resources has a separate rename pool which can be accessed independently and shared by all active threads. Instructions that update more than one destination registers are broken into sub-instructions.

The ISU assigns Load Tags (LTAG) and Store Tags (STAG) to manage the flow of load and store instructions. An LTAG corresponds to a pointer to the Load Reorder Queue (LRQ) entry assigned to a load instruction. An STAG corresponds to a pointer to the Store Reorder Queue (SRQ) entry assigned to a store instruction. Store instructions are issued twice, once as a store data internal operation (to fetch the store data from a register) and once as a store address internal operation (to compute the address of the store). The STAG is used to match up the store data internal operation with the store address internal operation in the SRQ. A virtual STAG/LTAG scheme is used to minimize dispatch holds due to running out of physical SRQ/LRQ entries. When a physical entry in the LRQ is freed up, a virtual LTAG will be converted to a real LTAG. When a physical entry in the SRQ is freed up, a virtual STAG will be converted to a real STAG. Virtual STAG/LTAGs are not issued to the LSU until they are subsequently marked as being real in the UniQueue. The ISU can assign up to 128 virtual LTAGs and 128 virtual STAGs to each thread.

The POWER8 processor employs three separate issue queues: a 15-entry Branch Issue Queue (BRQ) for branch instructions, an 8-entry Condition Register Queue (CRQ) for condition register instructions, and a 64-entry UniQueue consisting of the UQ0 and UQ1 halves for all other instructions. Dispatched instructions are stored in the corresponding issue queues and then issued to the execution units one cycle after dispatch at the earliest for the BRQ/CRQ and three cycles after dispatch at the earliest for the UniQueue.

The BRQ and CRQ are shifting queues, where dispatched instructions are placed at the top of the queue and then trickle downward toward the bottom of the queue. To save power, the UniQueue is implemented as a non-shifting queue and managed by queue position pointers. The queue position pointers are shifted, but the UniQueue entries themselves do not move, which significantly reduces the switching power in the large UniQueue.

Instructions can issue in out-of-order from all of these queues, with higher priority given to the older ready instructions. An instruction in an issue queue is selected for isssuing when all source operands for that instruction are available. In addition, the LTAG or STAG must have a real entry for a load or store instruction respectively, before it can be issued. For the BRQ and CRQ, instruction dependencies are checked by comparing the destination physical pointer of the renamed resource against all outstanding source physical pointers. For the UniQueue, dependencies are tracked using queue pointers via a dependency matrix. The issue queues together can issue a total of ten intructions per cycle: one branch, one condition register logical, two fixed-point instructions to the FXU, two load, store or simple fixed-point instructions to the LSU, two load or simple fixed point instructions to the LU, and two vector-scalar instructions (floating-point, VMX or VSX) to the VSU. DFU and Crypto instructions can also be issued using the VSU slots (up to one of each type per cycle).

The BRQ contains only branch instructions and it receives up to two branches per cycle from the dispatch stage, and can issue one branch instruction per cycle for execution to the IFU. The CRQ contains the CR logical instructions and move from SPR instructions for the IFU, the ISU, and the Pervasive Control Unit (PCU). The CRQ can receive up to two instructions per cycle from dispatch, and can issue one instruction per cycle to the IFU.

The UniQueue is implemented as a 64-entry queue that is split into two halves (UQ0 and UQ1) of 32 entries each. Each half of the queue contains all instructions that are executed by the FXU, LSU, LU, VSU, DFU, or Crypto units. Instructions are steered at the dispatch stage to the appropriate UniQueue half. The UniQueue can receive up to three instructions from dispatch per cycle per UniQueue half. Relative age of the instructions in the UniQueue is determined by an age matrix, since it is a non-shifting queue, that is written at dispatch time.

Each half of the UniQueue can issue one fixed-point instruction to the FXU, one load, store or simple fixed-point instruction to the LSU, one load or simple fixed-point instruction to the LU, and one vector-and-scalar instruction to the VSU per cycle, for a total of four instructions per cycle. Simple fixed-point instructions may be selected for issue to the LSU and LU for improved fixed-point throughput, with the same latency as a load operation from L1 D-cache.

Instructions are issued speculatively, and hazards can occur, for example, when a fixed-point operation dependent on a load operation is issued before it is known that the l oad misses the D-cache or the DERAT (see the Load/Store Unit description below). On a mis-speculation, the instruction is rejected and re-issued a few cycles later.

Load/Store Unit

The Load/Store Unit (LSU) is responsible for executing all the load and store instructions, managing the interface of

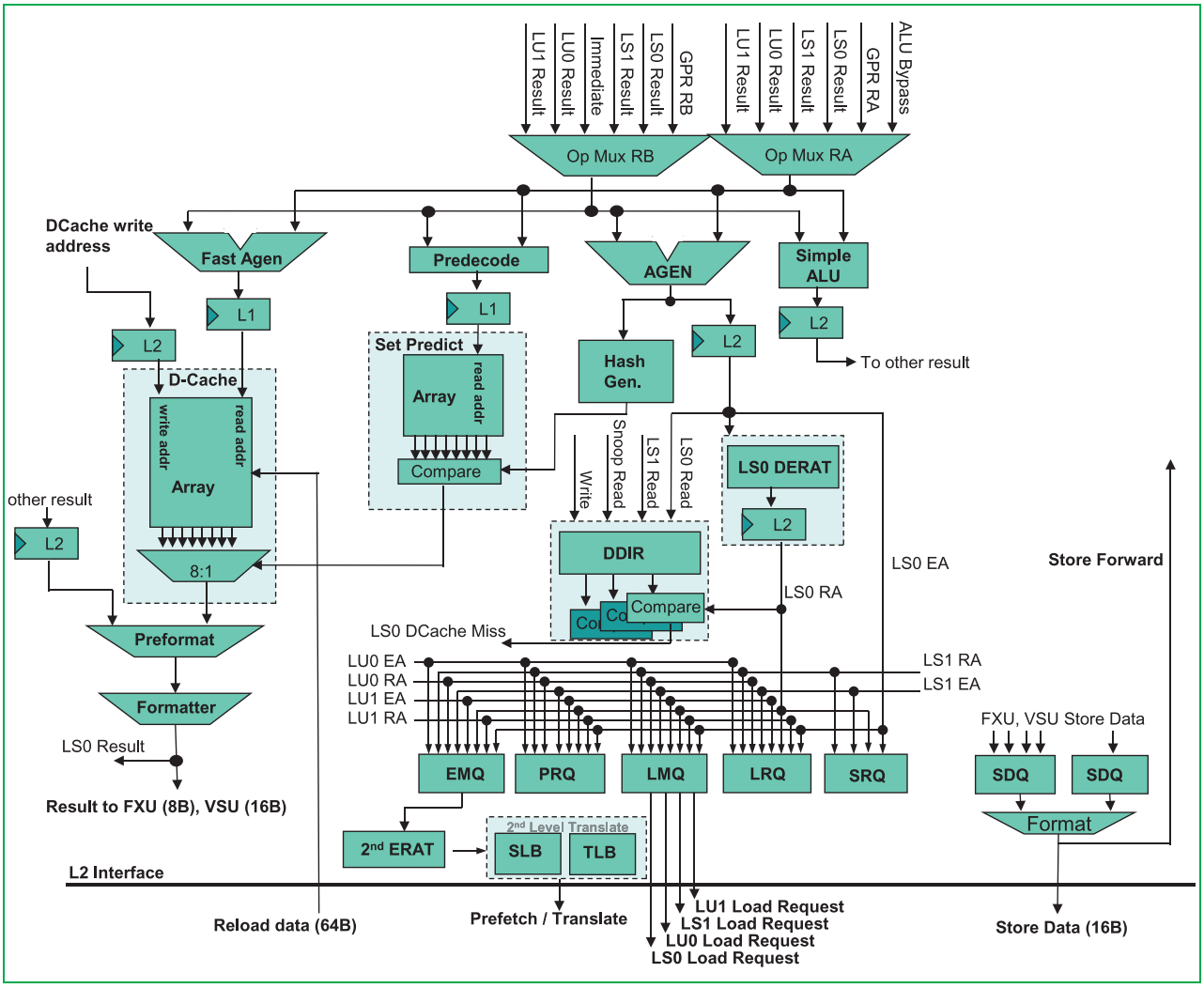

Figure 6

POWER8 LSU micro architecture (LS0 pipe shown).

the core with the rest of the systems through the unified L2 cache and the Non-Cacheable Unit (NCU), and implementing address translation as specified in the Power ISA. The POWER8 LSU contains two symmetric load pipelines (L0 and L1) and two symmetric load/store pipelines (LS0 and LS1). **Figure 6** illustrates the microarchitecture of the POWER8 LS0 pipeline.

Data fetching

Each of the LS0 and LS1 pipelines are capable of executing a load or a store operation in a cycle. Furthermore, each of L0 and L1 pipelines are capable of executing a load operation in a cycle. In addition, simple fixed-point operations can also be executed in each of the four pipelines in the LSU, with a latency of three cycles.

The LSU contains several subunits, including the load/store address generation (AGEN) and execution subunits, the store reorder queue (SRQ), the store data queue (SDQ), the load reorder queue (LRQ), the load miss queue (LMQ), and the L1 data cache array (D-cache) with its supporting set predict and directory arrays (DDIR), and the data prefetch engine (PRQ). The address translation mechanism in the LSU includes the Effective-to-Real Address Translation for data (DERAT), the Effective-to-Real Address Translation (ERAT) Miss Queue (EMQ), the Segment Lookaside Buffer (SLB), and TLB.

Load/store execution

In ST mode, a given load/store instruction can execute in any appropriate pipeline: LS0, LS1, L0 and L1 for loads, LS0 and LS1 for stores. In SMT2, SMT4, and SMT8 mode,

instructions from half of the threads execute in pipelines LS0 and L0, while instructions from the other half of the threads execute in pipelines LS1 and L1.

Instructions are issued to the load/store unit out-of-order, with a bias towards the oldest instructions first. Stores are issued twice; an address generation operation is issued to the LS0 or LS1 pipeline, while a data operation to retrieve the contents of the register being stored is issued to the L0 or L1 pipeline.

Main dataflow buses into and out of the LSU include a 64-byte reload data bus from and a 16-byte store data bus to the L2 cache, two 16-byte load data buses (one per execution pipeline) to and two 16-byte store data buses from the VSU, and two 8-byte store data buses (one per execution pipeline) from the FXU. The load data buses to the VSU have each a tap off of 8-byte load data to a corresponding FXU execution pipeline.

Fixed-point loads have a three-cycle load-to-use latency on a L1 D-cache hit. That is, two cycles of bubbles are introduced between a load and a dependent FXU operation. VSU loads have a five-cycle load-to-use latency on a L1 D-cache hit. That is, four cycles of bubbles are introduced between a load and a dependent VSU operation.

Each of the four LSU pipelines can also execute fixed-point add and logical instructions (simple fixed-point), allowing more fixed-point execution capability for the POWER8 core and greater flexibility to the ISU in the issuing of instructions.

Load/store ordering

The LSU must ensure the effect of architectural program order of execution of the load and store instructions, even though the instructions can be issued and executed out-of-order. To achieve that, the LSU employs two main queues: the store reorder queue (SRQ) and the load reorder queue (LRQ).

The SRQ is a 40-entry, real address based CAM structure. Whereas 128 virtual entries per thread are available to allow a total of 128 outstanding stores to be dispatched per thread, only a total of 40 outstanding stores may be issued, since a real, physical SRQ entry is required for the store to be issued. The SRQ is dynamically shared among the active threads. An SRQ entry is allocated at issue time and de-allocated after the completion point when the store is written to the L1 D-cache and/or sent to the L2 Cache. For each SRQ entry, there is a corresponding store data queue (SDQ) entry of 16 bytes. Up to 16 bytes of data for a store instruction can be sent to the L2 Cache (and also written to the L1 D-Cache on a hit) in every processor cycle. Store forwarding is supported, where data from an SRQ entry is forwarded to an inclusive, subsequent load, even if the store and load instructions are speculative.

Similar to the SRQ, the LRQ is a 44-entry, real address based, CAM structure. Again, 128 virtual entries per thread are available to allow a total of 128 outstanding loads to be dispatched per thread, but only a total of 44 outstanding loads may be issued, since a real, physical LRQ entry is required for the load to be issued. The LRQ is dynamically shared among the threads. The LRQ keeps track of out-of-order loads, watching for hazards. Hazards generally exist when a younger load instruction executes out-of-order before an older load or store instruction to the same address (in part or in whole). When such a hazard is detected, the LRQ initiates a flush of the younger load instruction and all its subsequent instructions from the thread, without impacting the instructions from other threads. The load is then re-fetched from the I-cache and re-executed, ensuring proper load/store ordering.

Address translation

In the Power ISA, programs execute in a 64-bit effective addresses space. (A 32-bit operating mode supports the execution of programs with 32-bit general purpose registers and 32-bit effective addresses.) During program execution, 64-bit effective addresses are translated by the first level translation into 50-bit real addresses that are used for all addressing in the cache and memory subsystem.

The first level translation consists of a primary Data Effective-to-Real Address Translation (DERAT), a secondary DERAT, and an Instruction Effective-to-Real Address Translation (IERAT, already discussed in the Instruction Fetch Unit Section). When a data reference misses the primary DERAT, it looks up the address translation in the secondary DERAT. If the translation is found in the secondary DERAT, it is then loaded into the primary DERAT.

If the translation is not found in either the primary or the secondary DERAT, the second-level translation process is invoked to generate the translation. When an instruction reference misses the IERAT, the second-level translation is also invoked to generate the translation. The second-level translation consists of a per-thread Segment Lookaside Buffer (SLB) and a TLB that is shared by all active threads.

Effective addresses are first translated into 78-bit virtual addresses using the segment table and the 78-bit virtual addresses are then translated into 50-bit real addresses using the page frame table. While the architected segment and page frame tables are large and reside in main memory, the SLB and TLB serve as caches of the recently used entries from the segment table and page frame table, respectively. The POWER8 processor supports two segment sizes, 256 MB and 1 TB, and four page sizes: 4 KB, 64 KB, 16 MB, and 16 GB.

The primary DERAT is a 48-entry, fully-associative, Content Addressed Memory (CAM) based cache. Physically, there are four identical copies of the primary DERAT, associated with the two load/store pipelines and two load pipelines. In ST mode, the four copies of the primary DERAT are kept synchronized with identical contents. So, in ST mode, logically there are a total of 48 entries available. In the SMT modes, two synchronized primary DERATs

(in LS0 and L0 pipes) contain translation entries for half of the active threads while the two other synchronized primary DERATs (in LS1 and L1 pipes) contain translation entries for the other half of the active threads. In the SMT modes, the first two paired primary DERATs contain addresses that can be different from the other two paired primary DERATs, for a total of 96 logical entries. Each Primary DERAT entry translates either 4 KB, 64 KB, or 16 MB pages. The 16 GB pages are broken into 16 MB pages in the primary DERAT. The primary DERAT employs a binary tree Least Recently Used (LRU) replacement policy.

The secondary DERAT is a 256-entry, fully associative, CAM-based cache. In single thread mode, all 256 entries are available for that thread. In SMT mode, the secondary DERAT is treated as two 128-entry arrays, one for each thread set. The secondary DERAT replacement policy is a simple First-In First-Out (FIFO) scheme.

The SLB is a 32-entry-per-thread, fully associative, CAM-based buffer. Each SLB entry can support 256 MB or 1 TB segment sizes. The Multiple Pages Per Segment (MPSS) extension of Power ISA is supported in the POWER8 processor. With MPSS, a segment with a base page size of 4 KB can have 4 KB, 64 KB, and 16 MB pages concurrently present in the segment. For a segment with a base page size of 64 KB, pages of size 64 KB and 16 MB are allowed concurrently. The SLB is managed by supervisor code, with the processor generating a data or instruction segment interrupt when an SLB entry needed for translation is not found.

The TLB is a 2,048-entry, 4-way set associative buffer. The TLB is managed by hardware, and employs a true LRU replacement policy. A miss in the TLB causes a table-walk operation, by which the TLB is reloaded from the page frame table in memory. There can be up to four concurrent outstanding table-walks for TLB misses. The TLB also provides a hit-under-miss function, where the TLB can be accessed and return translation information to the DERAT while a table-walk is in progress. In the POWER8 LSU, each TLB entry is tagged with the LPAR (logical partition) identity. For a TLB hit, the LPAR identity of the TLB entry must match the LPAR identity of the active partition running on the core. When a partition is swapped in, there is no need to explicitly invalidate the TLB entries. If a swapped-in partition has run previously on the same core, there is a chance that some of its TLB entries are still available which reduces TLB misses and improves performance.

L1 data cache organization

The POWER8 LSU contains a dedicated 64 KB, 8-way set-associative, banked L1 D-cache. The cache line size is 128 bytes, consisting of four sectors of 32 bytes each. There is a dedicated 64-byte reload data interface from the L2 cache, which can supply 64 or 32 bytes of data in every processor cycle. The cache line is validated on a sector basis as each 32-byte sector is returned from the memory subsystem. Loads can hit against a valid sector before the entire cache line is validated.

The L1 D-cache has five ports: four read ports and one write port. The four read ports are available for up to four load instructions per cycle. The one write port is available for a cache line reload or a cache store operation per cycle. The cache store operation is distinct from the issue and execution of a store instruction. The cache store operation drains in-order from the store queue (SRQ/SDQ) after the completion of the store instruction and sends the store data to both the L1 data cache and the L2 cache at the same time. A write has higher priority over a read, and a write for a cache line reload has higher priority than a write for a completed store instruction. Logic ahead of the L1 D-cache access can determine whether it is possible to access data for all four load instructions in a given cycle given the bank access restrictions described below. If all four loads cannot access data in a given cycle, two of them will execute and the other two will be rejected and re-issued.

The L1 D-cache consists of 16 physical macros organized by data bytes, each macro partitioned into 16 banks based on effective address bits, for a total of 256 banks. The physical cache macros are built with 6T cells with dual read or a single write capability. A bank can do one write or two read in a given cycle. The cache banking allows for one write and two reads to occur in the same cycle within one physical macro, so long as the reads are not to the same bank(s) as the write. If a read has a bank conflict with a write, the load instruction is rejected and re-issued. A 64 byte cache line reload spans 32 banks, and a 32 byte cache line reload spans 16 banks, while a completed store instruction spans from one to eight banks depending on data length and data alignment (nine banks for 16B unaligned data access).

A load is naturally aligned when the address of the load is an integer multiple of the size of the data being loaded. All aligned load operations perform at the three-cycle L1 D-cache access latency. Unaligned accesses that do not cross the 128B boundary also perform at the same latency, if there is a L1 D-cache hit. An unaligned data access that crosses the 128B boundary (that is, the cache line boundary) can incur additional latency compared to a normal data access.

The L1 D-cache is a store-through design. All stored data are sent to the L2 cache, and no L1 cast outs are required. The L1 D-cache and L2 cache of the POWER8 processor follow a strictly inclusive policy. The L1 D-cache is not allocated on a store miss. Instead, the store is just sent to the L2 cache. The L1 D-cache has byte write capability of up to 16 bytes in support of the different types of store instructions in the Power ISA.

The L1 D-cache is indexed with the effective address (EA) of a load or store operation. The L1 D-cache directory employs a hybrid true/binary tree LRU replacement policy. A capacity of 64 KB and 8-way set associativity results in

8 KB per way. Therefore, bits 51 through 56 of the effective address [EA(51:56)] are used to index into the L1 D-cache. [EA(0) is the most significant bit of the effective address. EA(57:63) are used to index within a cache line.] A way-predict array is used to reduce the L1 D-cache load hit latency. The way-predict array is based on EA and is used as a mini-directory to select which one of the eight L1 D-cache ways contains the load data. The way-predict array is organized as the L1 D-cache: indexed with EA(51:56) and 8-way set-associative. Each entry contains 11 hash bits obtained from hashing bits EA(32:50), a valid bit for each thread, and a parity bit.

When a load executes, the generated EA(51:56) is used to index into the way-predict array, and EA(32:50) is hashed and compared to the contents of the eight ways of the indexed entry. When an EA hash match occurs and the appropriate thread valid bit is active, the match signal is used as the way select for the L1 D-cache data. If there is no EA hash match, it indicates a cache miss. However, an EA hash match does not necessarily mean a L1 D-cache hit. For L1 D-cache hit determination, the effective address is used to look up the L1 data cache directory for the real address and then compare this real address with the real address obtained from the DERAT for the given EA.

When a cache line is first loaded into the L1 D-cache, the default is to enter it in a shared mode where all thread valid bits for the line are set. A non-shared mode is dynamically selected on an entry-by-entry basis to allow only one thread valid bit to be active. This is beneficial to avoid thrashing among the threads, allowing the same EA hash to exist for each thread separately at the same time when they are assigned to different real addresses (RA).

Storage alignment

The LSU performs most loads and stores that are unaligned with the same timing as naturally aligned loads and stores. If data referenced by a load instruction crosses a cache line boundary and both cache lines are in the L1 data cache, the access will have only a 5-cycle penalty over a normal L1 D-cache access. In the POWER8 processor all loads that are allowed to cross a cache line can get this treatment with any byte alignment. The implementation saves the bytes from the first cache line access and then 5 cycles later accesses the second cache line and merges data from the first cache line with the data from the second cache line.

Load miss handling

Loads that miss the L1 D-cache initiate a cache line reload request to the L2 Cache. The load releases the issue queue entry and creates an entry in the Load Miss Queue (LMQ) to track the loading of the cache line into the L1 D-cache and also to support the forwarding of the load data to the destination register. When the load data returns from the L2 Cache, it gets higher priority in the LSU pipeline and the data is transferred to the destination register. The LMQ is real address based, and consists of sixteen entries, dynamically shared among the active threads. The LMQ tracks all cache line misses that result in reload data to the L1 D-cache, which also includes hardware data prefetch requests and data touch (or software prefetch) instructions, in addition to load instructions. The LMQ can support multiple misses (up to eight) to a given L1 D-cache congruence class (also called a set).

Data prefetch

The purpose of the data prefetch mechanism is to reduce the negative performance impact of memory latencies, particularly for technical workloads. These programs often access memory in regular, sequential patterns. Their working sets are also so large that they often do not fit into the cache hierarchy used in the POWER8 processor.

Designed into the load-store unit, the prefetch engine can recognize streams of 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 the L1, L2, and L3 caches so that much or all of the load latency can be hidden. The most urgently needed lines are prefetched into the nearest cache levels.

During stream start up, several lines ahead of the current demand reference can be requested from the memory subsystem. After steady state is achieved, each stream confirmation causes the engine to bring one additional line into the L1 cache, one additional line into the L2 cache, and one additional line into the L3 cache. To effectively hide the latency of the memory access while minimizing the potentially detrimental effects of prefetching such as cache pollution, the requests are staged such that the line that is being brought into the L3 cache is typically several lines ahead of the one being brought into the L1 cache. Because the L3 cache is much larger than the L1 cache, it can tolerate the most speculative requests more easily than the L1 cache can.

Prefetch begins by saving the effective address of the L1 D-cache misses in a 16-entry queue, offset up or down by one line address. Prefetch streams are tracked by their effective addresses and are allowed to cross small (4 KB) and medium (64 KB) memory page boundaries, but will be invalidated when crossing a large (16 MB) memory page boundary. All prefetch requests must therefore go through address translation before being sent to the memory subsystem.

Fixed-Point Unit

The Fixed-Point Unit (FXU) is composed of two identical pipelines (FX0 and FX1). As shown in **Figure 7**, each FXU

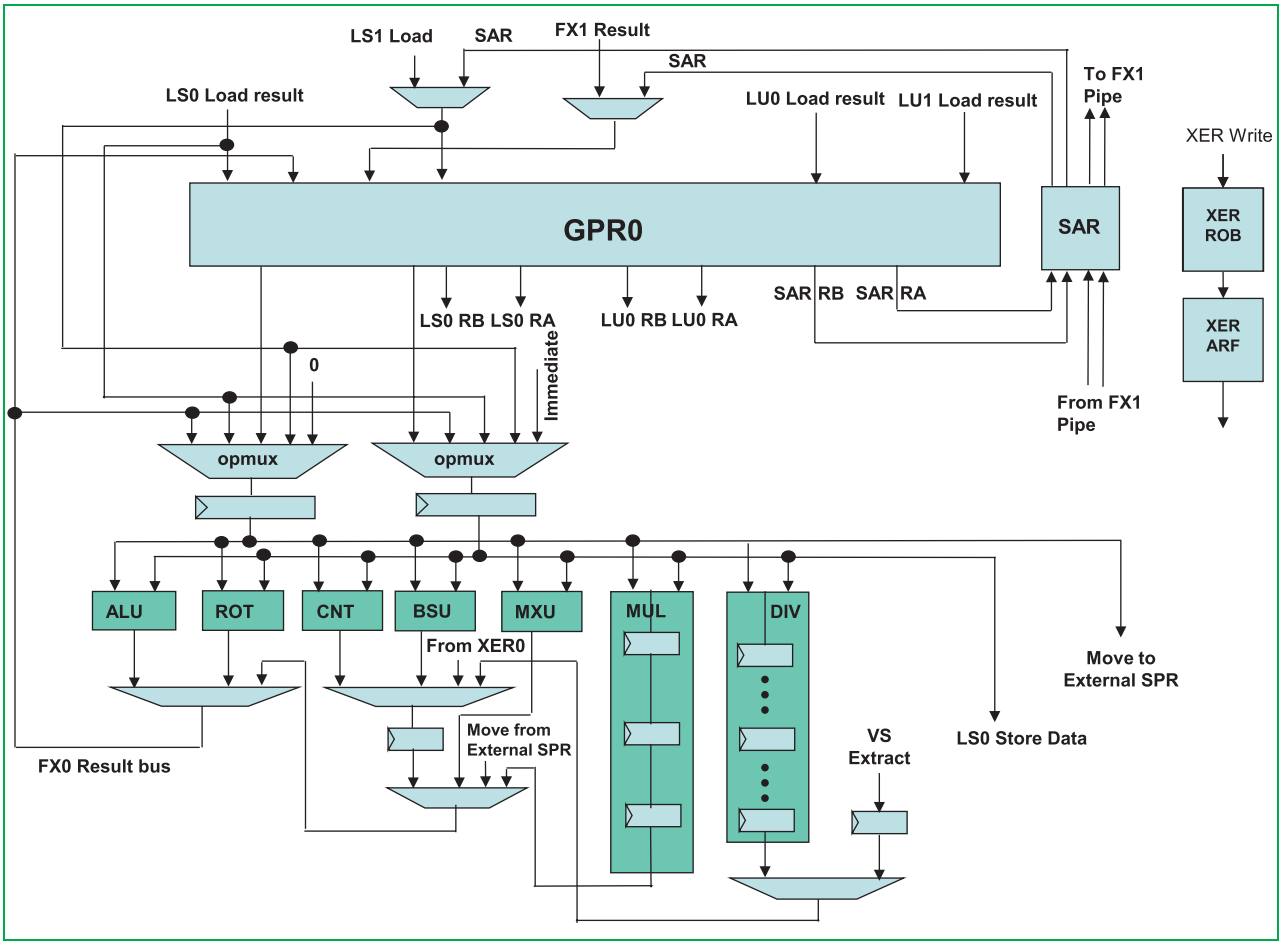

Figure 7

POWER8 FXU overview (FX0 pipe shown).

pipeline consists of a multiport General Purpose Register (GPR) file, an arithmetic and logic unit (ALU) to execute add, subtract, compares and trap instructions, a rotator (ROT) to execute rotate, shift and select instructions, a count unit (CNT) to execute count leading zeros instruction, a bit select unit (BSU) to execute bit permute instruction, a miscellaneous execution unit (MXU) to execute population count, parity and binary-coded decimal assist instructions, a multiplier (MUL), and a divider (DIV). Certain resources such as the Software Architected Register file (SAR) and Fixed-Point Exception Register (XER) file are shared between the two pipelines.

The most frequent fixed-point instructions are executed in one cycle and dependent operations may issue back to back to the same pipeline, if they are dispatched to the same UniQueue half (otherwise, a one-cycle bubble is introduced). Other instructions may take two, four, or a variable number of cycles.

At the heart of each FXU pipeline is a GPR file with 124 entries which holds all the rename and a subset of the architected registers for up to four threads. Additional architected registers are kept in the SAR register files. The GPR has eight read ports, two supplying operands for the fixed-point pipeline, two supplying operands to the load/store pipeline, two supplying operands to the load pipeline, and two supplying register data to the SAR. The GPR has six write ports: two for the fixed-point pipelines, two for the load/store pipelines, and two for the load pipelines. (Updates to a particular GPR can come from either set of fixed-point, load/store and load pipelines when the core is in ST mode.) The write ports from the remote fixed-point and load/store pipelines are shared with write ports from the SAR. In SMT modes, writes from remote pipelines are disabled and the ports can be used exclusively to load data from the SAR. The POWER8 core implements a VSU extract bus which is routed to the result multiplexer of each FXU pipe. The extract bus significantly reduces latency for VSR to GPR transfers.

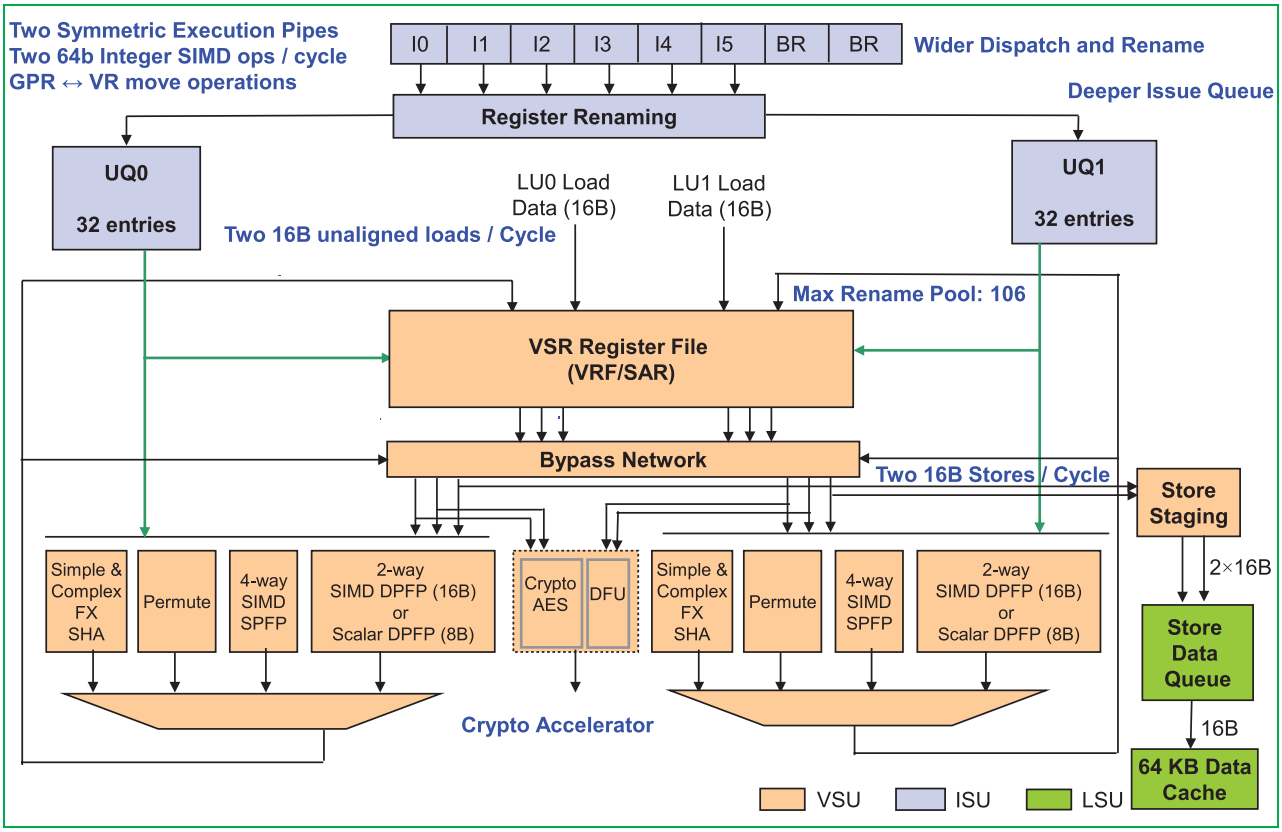

Figure 8

POWER8 fully symmetric VSU pipelines. (SPFP: single-precision floating-point; DPFP: double-precision floating-point.)

The contents of the two GPR register files in each pipeline are managed by the ISU to be identical in ST mode, but distinct in SMT2, SMT4, and SMT8 modes. That is, in SMT2, SMT4, or SMT8 mode the GPR in one pipeline contains the registers for one set of threads, while the GPR in the other pipeline contains the registers for the other set of threads.

Each of the two POWER8 FXU pipelines contain a 32- or 64-bit divider implemented using a Sweeney, Robertson, and Tocher (SRT) algorithm with a radix of $r = 16$. To improve throughput during the multi-cycle divide, instructions can be issued to other execution units (ALU, ROT, BSU, CNT, MXU, MUL) under a divide operation except when there is a result bus conflict.

The POWER8 FXU supports Transactional Memory (TM) by doubling the register space to hold a backup copy of all the architected registers. Rather than doubling the size of the GPR, the SAR was added to expand the state space of the architected GPR registers. The XER, which is the other architected register in the FXU, had to grow for TM support. The XER is implemented as a Reorder Buffer (ROB) and Architected Register File (ARF) structure to accommodate the increase in state space.

Vector-and-Scalar/Decimal Floating-Point Units

The POWER8 processor Vector-and-Scalar Unit (VSU), shown in **Figure 8**, has been completely redesigned from its initial implementation in the POWER7 processor [4] to support the growing computation and memory bandwidth requirements of business analytics and big data applications. The POWER8 VSU now supports dual issue of all scalar and vector instructions of the Power ISA defined in Chapter 4 "floating point facility" (FPU), Chapter 5 "Vector Facility" (VMX), and Chapter 7 "Vector-Scalar Floating Point Operations" (VSX) of Power ISA Book I [5]. Further improvements include a two-cycle VMX/VSX Permute (PM) pipeline latency, doubling of the store bandwidth to two 16-byte vectors/cycle to match the 32-byte/cycle load bandwidth, and execution of all floating-point compare instructions using the two-cycle Simple Unit (XS) pipeline to speedup branch execution. Other latencies remain unchanged from the POWER7 processor design point, supporting fast six-cycle bypass within the floating-point unit.

The POWER8 VSU implements full architected state storage for 8-way simultaneous multi-threaded (SMT8) execution, alongside additional checkpoint storage for transactional

memory (TM) support. The total number of 1,024 × 16-byte VSX registers is implemented as a two-level register space. The second level, namely the Software Architected Registers (SAR), maintains all 64 architected VSX registers plus up to 64 TM checkpointed registers per thread. Two copies of a 144-entry vector register file (VRF), one associated with each UniQueue, constitute the first level register space. Each VRF contains up to 64 recently used architected registers and up to 80 in-flight rename registers shared across all threads in the corresponding UniQueue half. In ST mode, the contents of both VRFs are kept synchronized. When running in SMT modes, the two VSU issue ports and VRFs work separately, thereby doubling the number of in-flight copies of architected and rename registers. The SAR space always appears as shared resource of the nine ports and all eight threads allowing for dynamic movement of threads or alternation of ST/SMT mode.

The VSU features a large number of new instructions and architectural refinements for applications like business analytics, big data, string processing, and security. The VSX pipelines now supports 2-way 64-bit vector and 128-bit scalar integer data types and new direct GPR-to/from-VSR move operations that provide a fixed-latency and high bandwidth data exchange between the vector and general purpose registers. The added VMX crypto instruction set is targeted towards AES, SHA2 and CRC computations and several instructions have been promoted into VSX to gain access to all 64 architected vector registers.

Also new vector 32-bit multiply instructions were introduced to speed-up applications like hashing in business analytics. By reusing the multiplier hardware in the floating point units, the new 32-bit multiply instructions could be added with little additional chip area, while still executing in a fully pipelined fashion. On signed and unsigned integer data types, the POWER8 VSU is capable of performing eight word (32-bit) or 16 halfword (16-bit) or 32 byte multiplies per cycle. If executing floating point vector instructions, the two POWER8 Floating Point Units on either VSU pipe operate pair wise. A single FPU can either operate on 64-bit double precision data, or on two 32-bit single precision data. As in the POWER7+ processor [10], the computation bandwidth of the processor is 16 single-precision floating-point operations (eight fused-multiply-add operations) per cycle per core (twice as high as in the POWER7 processor) or eight double-precision floating-point operations (four fused-multiply-add operations) per cycle per core. Finally the POWER8 floating point units support the new VSX Scalar Floating Point Single Precision instruction set to enable the use of all 64 VSX registers for single-precision data.

The Decimal Floating Point Unit (DFU) [12] in the POWER8 core allows fully pipelined execution of the Power ISA "Decimal Floating Point" instructions (described in Power ISA Book I, Chapter 6). The DFU attachment has greatly been improved to provide symmetrical, conflict-free access from both UniQueue ports, resulting in more predictable execution latencies. The issue-to-issue latency is 13 cycles for dependent instructions. The DFU is IEEE 754-2008 compliant and includes native support for signed decimal fixed-point add and fixed-point subtract with an operand length of up to 31 decimal digits, which speeds up the execution of business analytics applications such as DB2 BLU.

The new VSU microarchitecture doubles the number of VSX/VMX simple integer and permute units, supports many new instructions, adds a new crypto engine and greatly improves attachment of the redesigned DFU pipeline. With all these enhancements, the overall performance for many of the new computational intensive workloads is greatly improved in the POWER8 processor.

Performance monitoring and adaptive optimization

The POWER8 core was designed with support for adaptive workload optimization using a variety of technologies. It introduces a robust and versatile performance monitoring infrastructure as a key enabler of making such workload optimization decisions. The POWER8 performance monitoring unit (PMU) can be used to guide system level optimizations and settings (such as using the new WORT microarchitectural control), to collect program profiles for profile-directed feedback optimization and to support a range of dynamic compilation and optimization technologies, such as for Java** just-in-time (JIT) compilers, commonly used to support a range of business-critical enterprise applications.

To more efficiently integrate the performance monitoring unit with dynamic compilers, the POWER8 processor introduces a number of new architected interfaces, including application-level ones, for configuring and reading collected performance statistics. In addition, the POWER8 core also provides a low latency interface for reporting PMU events with the introduction of event-based branches as a user-level interrupt facility.

In addition to expanding the possible events available for execution sampling and summary statistics collection provided in previous designs, the POWER8 PMU also supports the collection of path histories. These capture control flow decisions over a sliding window of the last n conditional branches executed, giving dynamic compilers (such as Java JIT compilers) the opportunity to understand and optimize the dynamic control flow in applications.

Summary and conclusion

The POWER8 processor continues the tradition of innovation in the POWER line of processors. In addition to being the best-of-breed design for IBM's commercial workloads, the POWER8 processor design is also targeted for big data, analytics, and cloud application environments and provides the highest performance design in the industry. The POWER8 core is designed with high throughput

performance in mind and supports eight powerful threads per core. For many commercial workloads, each POWER8 core can provide about 1.5 times more single thread performance and twice the throughput performance over a POWER7 core.

Acknowledgments

A large number of people worked on the POWER8 processor core microarchitecture described in this paper. We acknowledge their efforts in developing the innovative ideas, conducting performance analysis, and working on the implementation and verification, thus enabling the POWER8 core to be the highest-performing core in the industry. We thank all of these IBM technologists for their dedication to this project.

References

1. R. Kalla and B. Sinharoy, "POWER7: IBM's next generation POWER microprocessor," in *Hot Chips 21*, Aug. 2009. [Online]. Available: http://www.hotchips.org/wp-content/uploads/hc_archives/hc21/3_tues/HC21.25.800.ServerSystemsII-Epub/HC21.25.829.Kalla-IBM-POWER7NextGenerationServerProcessorv7display.pdf
2. W. Starke, "POWER7: IBM's next generation, balanced POWER server chip," presented at the Hot Chips 21, Aug. 2009. [Online]. Available: http://www.hotchips.org/wp-content/uploads/hc_archives/hc21/3_tues/HC21.25.800.ServerSystemsII-Epub/HC21.25.835.Starke-IBM-POWER7SystemBalancev13_display.pdf
3. D. Wendel, R. Kalla, J. Warnock, R. Cargnoni, S. Chu, J. Clabes, D. Dreps, D. Hrusecky, J. Friedrich, S. Islam, J. Kahle, J. Leenstra, G. Mittal, J. Paredes, J. Pille, P. Restle, B. Sinharoy, G. Smith, W. Starke, S. Taylor, A. Van Norstrand, S. Weitzel, P. Williams, and V. Zyuban, "POWER7, a highly parallel, scalable multi-core high end server processor," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 145–161, Jan. 2011.
4. B. Sinharoy, R. Kalla, W. J. Starke, H. Q. Le, R. Cargoni, J. A. Van Norstrand, B. J. Ronchetti, J. Stuecheli, J. Leenstra, G. L. Guthrie, D. Q. Nguyen, B. Blaner, C. F. Marino, E. Retter, and P. Williams, "IBM POWER7 multicore server Processor," *IBM J. Res. & Dev.*, vol. 55, no. 3, Paper 1, pp. 1:1–1:29, May/Jun. 2011.
5. POWER ISA Version 2.07, May 10, 2013. [Online]. Available: https://www.power.org/documentation/
6. J. M. Tendler, J. S. Dodson, J. S. Fields, Jr., H. Le, and B. Sinharoy, "POWER4 system microarchitecture," *IBM J. Res. & Dev.*, vol. 46, no. 1, pp. 5–25, 2002.
7. B. Sinharoy, R. N. Kalla, J. M. Tendler, R. J. Eickemeyer, and J. B. Joyner, "POWER5 system microarchitecture," *IBM J. Res. & Dev.*, vol. 49, no. 4/5, pp. 505–521, 2005.
8. R. Kalla, B. Sinharoy, and J. Tendler, "IBM POWER5 chip: A dual-core multithreaded processor," *IEEE Micro*, vol. 24, no. 2, pp. 40–47, Mar./Apr. 2004.
9. H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden, "IBM POWER6 microarchitecture," *IBM J. Res. & Dev.*, vol. 51, no. 6, pp. 639–662, 2007.
10. S. Taylor, "POWER7+: IBM's next generation POWER microprocessor," in *Hot Chips 24*, 2012.
11. H. Q. Le, G. L. Guthrie, D. E. Williams, M. M. Michael, B. G. Frey, W. J. Starke, C. May, R. Odaira, and T. Nakaike, "Transactional memory support in the IBM POWER8 processor," *IBM J. Res. & Dev.*, vol. 59, no. 1, Paper 8, pp. 8:1–8:14, 2015.
12. L. Eisen, J. J. W. Ward, III, H. Tast, N. Mäding, J. Leenstra, S. M. Mueller, C. Jacobi, J. Preiss, E. M. Schwarz, and S. R. Carlough, "IBM POWER6 accelerators: VMX and DFU," *IBM J. Res. & Dev.*, vol. 51, no. 7, pp. 663–684, 2007.
13. B. Sinharoy, R. Swanberg, N. Nayar, B. Mealey, J. Stuecheli, B. Schiefer, J. Leenstra, J. Jann, P. Oehler, D. Levitan, S. Eisen, D. Sanner, T. Pflueger, C. Lichtenau, W. E. Hall, and T. Block, "Advanced features in IBM POWER8 systems," *IBM J. Res. & Dev.*, vol. 59, no. 1, Paper 1, pp. 1:1–1:18, 2015.

Received April 14, 2014; accepted for publication May 11, 2014

Balaram Sinharoy *IBM Systems and Technology Group, Poughkeepsie, NY 12601 USA (balaram@us.ibm.com).* Dr. Sinharoy is an IBM Fellow and the chief architect of the IBM POWER8 processor. Before his work on the POWER8 processor, he was the Chief Architect for the IBM POWER5 and POWER7 processors. Dr. Sinharoy has published numerous articles and authored approximately 135 issued or pending patents in many areas of computer architecture. Dr. Sinharoy also received several IBM Corporate Awards for his work in different generations of the IBM POWER processor. He is an IBM Master Inventor and an IEEE (Institute of Electrical and Electronics Engineers) Fellow.

James A. Van Norstrand *IBM Systems and Technology Group, Austin, TX 78758 USA (njvan@us.ibm.com).* Mr. Van Norstrand is a Distinguished Engineer in the IBM POWER development team. He graduated from Syracuse University in 1982 with a B.S.E.E. degree. He was the unit lead for the Instruction Fetch Unit on POWER7. Before POWER7, he was the core lead for the Cell Broadband Engine** chip, POWER4 lab manager, and IBM z System* designer for the IFU.

Richard J. Eickemeyer *IBM Systems and Technology Group, Rochester, MN 55901 USA (eick@us.ibm.com).* Dr. Eickemeyer received a B.S. degree in electrical engineering from Purdue University and M.S. and Ph.D. degrees from the University of Illinois at Urbana-Champaign. He is currently a Senior Technical Staff Member at IBM Corporation in Rochester, Minnesota, where he is the processor core performance team leader for IBM POWER servers and is working on future processor designs. Previously, he has worked on several different processor designs. His research interests are computer architecture and performance analysis. He has authored many papers and has been awarded 40 U.S. patents with others pending. He has been named an IBM Master Inventor. He has received several IBM awards including two IBM Corporate Awards.

Hung Q. Le *IBM Systems and Technology Group, Austin, TX 78758 USA (hung@us.ibm.com).* Mr. Le is an IBM Fellow in the POWER development team of the Systems and Technology Group. He joined IBM in 1979 after graduating from Clarkson University with a B.S. degree in electrical and computer engineering. He worked on the development of several IBM mainframe and POWER/PowerPC* processors and has been contributing to the technical advancement of IBM processor technology such as advanced high-frequency out-of-order instruction processing, simultaneous multithreading, and transactional memory. He led the POWER8 chip development and is developing the microarchitecture of the next Power processor. He holds more than 100 U.S. patents.

Jens Leenstra *IBM Systems and Technology Group, Boeblingen DE 71032 Germany (leenstra@de.ibm.com).* Mr. Leenstra is an IBM Senior Technical Staff Member and the lead for the IBM

POWER7 and POWER8 VSU. He worked on the design and verification of I/O chips, multiprocessor system verification of the IBM S/390* G2 and G3 mainframe computers, the Cell Broadband Engine processor SPEs (synergistic processor elements), and POWER6 processor VMX unit. He has 30 issued patents and is an IBM Master Inventor.

Dung Q. Nguyen *IBM Systems and Technology Group, Austin, TX 78758 USA (dqnguyen@us.ibm.com)* Mr. Nguyen is a Senior Engineer in the POWER development team of the Systems and Technology Group. He joined IBM in 1986 after graduating from the University of Michigan with an M.S. degree in materials engineering. He has worked on the development of many processors, including POWER3 through POWER8 processors. He is currently the unit lead for the Instruction Sequencing Unit on future POWER microprocessors. He has more than 80 issued patents and is an IBM Master Inventor.

Brian Konigsburg *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (brk@us.ibm.com)* Mr. Konigsburg earned a B.S.E.E. and a B.S.C.S. degree from the University of Florida. He is a Senior Technical Staff Member in IBM Research in the Design Automation area. He joined IBM in 1995 and has worked on several IBM POWER and IBM mainframe processor development teams as a processor core unit lead including instruction, load/store, and floating point units. He was also the performance lead for POWER7 and POWER8 processors. Mr. Konigsburg holds numerous patents in the area of instruction fetch and out-of-order instruction processing.

Kenneth Ward *IBM Systems and Technology Group, Austin, TX 78758 USA (wardk@us.ibm.com).* Mr. Ward earned a B.S. degree in mathematics and an M.S. degree in electrical engineering from the University of Florida. He is a Senior Engineer in the POWER development team of the Systems and Technology Group. He joined IBM in 1989 and has held a variety of positions in systems integration, systems development, card design, and processor development. He has worked in the areas of POWER5 Elastic I/O, POWER6 core recovery, POWER7 nest fabric, and recently as the unit lead for the POWER8 Fixed Point Unit (FXU). He is currently working on the POWER9 completion and flush implementation.

Mary D. Brown *IBM Systems and Technology Group, Austin, TX 78758 USA.* Dr. Brown received her B.S. degree in computer science from Florida State University, her M.S. degree in computer science and engineering from the University of Michigan, and her Ph.D. degree in computer engineering from the University of Texas at Austin. She started working at IBM in 2005 as a logic designer for the ISU for POWER7. On POWER8, she was the issue queue lead, and she was the Instruction Fetch Unit Lead starting in 2013.

José E. Moreira *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (jmoreira@us.ibm.com).* Dr. Moreira is a Distinguished Research Staff Member in the Commercial Systems department at the IBM T. J. Watson Research Center. He received a B.S. degree in physics and B.S. and M.S. degrees in electrical engineering from the University of Sao Paulo, Brazil, in 1987, 1988, and 1990, respectively. He also received a Ph.D. degree in electrical engineering from the University of Illinois at Urbana-Champaign in 1995. Since joining IBM at the T. J. Watson Research Center, he has worked on a variety of high-performance computing projects. He was system software architect for the Blue Gene*/L supercomputer, for which he received an IBM Corporate Award, and chief architect of the Commercial Scale Out project. He currently leads IBM Research work on the architecture of Power processor. He is author or coauthor of over 100 technical papers and ten patents. Dr. Moreira is a member of the Institute of Electrical and Electronics Engineers (IEEE) and a Distinguished Scientist of the Association for Computing Machinery (ACM).

David Levitan *IBM Systems and Technology Group, Austin, TX 78750 USA (levitan@us.ibm.com).* Mr. Levitan received his bachelor's degree in electrical engineering from McGill University in 1981, and his master's degree in computer engineering from Syracuse University in 1987. He is Senior Engineer and a Master Inventor who has reached the sixteenth invention achievement plateau at IBM. Mr. Levitan started work at IBM in Poughkeepsie, New York, in 1981. From 1981 until 1987, he worked in system simulation on various 3090 processors, and then from 1987 until 1990, he worked in the System Assurance Kernel Group. From 1990 until the present, Mr. Levitan has worked in PowerPC microprocessor development on various PowerPC microprocessors.

Steve Tung *IBM Systems and Technology Group, Austin, TX 78758 USA (stung@us.ibm.com).* Mr. Tung is a senior engineer in the POWER development team of the Systems and Technology Group. He has worked on the development of several POWER/PowerPC processors, particularly in load and store units. Mr. Tung received an M.S. degree in computer engineering from Syracuse University.

David Hrusecky *IBM Systems and Technology Group, Austin, TX 78758 USA (hrusecky@us.ibm.com).* Mr. Hrusecky is an advisory engineer in the POWER development team of the Systems and Technology Group. He has worked on core development L1 caches of several POWER processors, including POWER6, POWER7, and POWER8. He received a B.S. degree in computer engineering from Rochester Institute of Technology.

James W. Bishop *IBM Systems and Technology Group, Endicott, NY 13760 USA (bish@us.ibm.com).* Mr. Bishop is a Senior Engineer in the POWER development team of the Systems and Technology Group. He joined IBM in 1984 after graduating from the University of Cincinnati with a B.S. degree in electrical engineering. He subsequently earned an M.S. degree in computer engineering from Syracuse University in 1993. While at IBM, he has been a logic designer on memory and processor subsystems for System/390*, AS/400*, and Power. He has worked on the development of several POWER processors including POWER6, POWER7, and POWER8. Mr. Bishop is the author of 12 technical disclosures and 17 patents.

Michael Gschwind *IBM Systems and Technology Group, Poughkeepsie, NY 12601 USA (mkg@us.ibm.com).* Dr. Gschwind is a Senior Technical Staff Member and Senior Manager of the Systems Architecture team. In this role, Dr. Gschwind is responsible for the definition of the Power Systems and mainframe architecture. Previously, he was Chief Floating-Point Architect and Technical Lead for core reliability for Blue Gene/Q, was the architecture lead for the PERCS (Productive, Easy-to-use, Reliable Computing System) project defining the future POWER7 processor, and had key architecture and microarchitecture roles for the Cell Broadband Engine, Xbox 360**, and POWER7 processors. Dr. Gschwind also developed the first Cell compiler and served as technical lead and architect for the development of the Cell software-development environment. Dr. Gschwind has published numerous articles and received about 100 patents in the area of computer architecture. In 2006, Dr. Gschwind was recognized as IT Innovator and Influencer by *ComputerWeek*. Dr. Gschwind is a member of the ACM SIGMICRO Executive Board, a Member of the IBM Academy of Technology, an IBM Master Inventor, an ACM Distinguished Speaker, and an IEEE Fellow.

Maarten Boersma *IBM Systems and Technology Group, Boeblingen DE 71032 Germany (mboersma@de.ibm.com).* Mr. Boersma received his M.Sc. degree in electrical engineering from the University of Twente, the Netherlands. He joined IBM in 2005 to work on the design of high-performance floating point units for the PowerXCell* 8i, POWER7, POWER7+, and POWER8 microprocessors. His focus is on power-efficient design and formal verification techniques.

Michael Kroener *IBM Systems and Technology Group, Boeblingen DE 71032 Germany (mkroener@de.ibm.com).*
Mr. Kroener is the lead for the IBM POWER7 and POWER8 DFU unit. Since 1994, he worked in the floating point area, first on IBM mainframe z System processors and later on POWER6 processor. He has 26 issued patents.

Markus Kaltenbach *IBM Systems and Technology Group, Boeblingen DE 71032 Germany (kaltenba@de.ibm.com).*
Mr. Kaltenbach received his diploma degree in computer science from the University of Tuebingen, Germany. Joining IBM in 2005 working on the IBM z10* mainframe processor and designs for the POWER7 processor, he acts as logic design lead for the POWER8 VSU unit. His focus is on microarchitecture, accelerators, synthesis, and timing.

Tejas Karkhanis *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (tkarkha@us.ibm.com).* Dr. Karkhanis is a Research Staff Member at the IBM T. J. Watson Research Center since 2008. His research interests are in various aspects of enterprise-class and high-performance class computing systems. From 2006 to 2008, Dr. Karkhanis worked at Advanced Micro Devices, where he contributed to consumer-class microprocessors. Dr. Karkhanis received his B.S., M.S., and Ph.D. degrees in 2000, 2001, and 2006, respectively, all from University of Wisconsin-Madison. He has filed several patents and authored several papers in top conferences and journals.

Kimberly M. Fernsler *IBM Systems and Technology Group, Austin, TX 78758 USA (kimfern@us.ibm.com).* Ms. Fernsler is an Advisory Engineer in the POWER development team of the Systems and Technology Group. She has worked on the development of several POWER/PowerPC processors, particularly on load and store units. Ms. Fernsler joined IBM in 1999 after receiving an M.S. degree in computer engineering from Carnegie Mellon University.