IBM DeveloperWorks AIX Virtual User Group
February 18th, 2016 webinar

The IBM Power8 Processor Core Microarchitecture

(a white paper summary with thoughts and considerations)

Prepared and Edited by Earl Jew (not an author of the whitepaper)
earlj@us.ibm.com; (310) 251-2907; Los Angeles, California, USA
Senior IT Consultant for IBM Power Systems and System Storage
IBM STG Lab Services Power Systems Delivery Practice

ABSTRACT

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.

Based on the article by the same title in

IBM JOURNAL OF RESEARCH & DEVELOPMENT VOL. 59 NO. 1 PAPER 2 JANUARY/FEBRUARY 2015

Written by 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

Presenter Commentary **Caveat**

- This presentation is an intermix of excerpts/figures from the original whitepaper and presenter commentary.
- The blue borders designate Presenter Commentary (like this slide).
- The presenter's comments are based solely on the words of the whitepaper, and do not offer any insider's insight beyond the content of this whitepaper.
- Please judge the presenter's thoughts and considerations accordingly.
- *That is, I don't know anything more than what we all can read together here.*

This paper describes the core microarchitecture innovations made in the POWER8 processor that resulted in these significant performance benefits.

Based on principles adopted in the POWER7 multi-core processor, 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.

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.

- Regarding "to significantly improve the socket-level, core-level and thread-level performance in each of the multiple simultaneous multithreading (SMT) modes", because there are as many as 12 CPUcores per socket *(which is a tremendous amount of processing capacity)*, configuring an LPAR to reside on only one socket that accesses only local DIMMs – should be a pervasive POWER8 imperative – given its ideal performance advantages.
- **Tight&Fat:** Configure fewer vCPUs, grant 0.7-0.9 eCPU per vCPU, and drive the core-level harder with SMT-2/4/8 thread-level workloads on POWER8.
- View/confirm any configuration with `AIX: lssrad -av` output.
- Study&practice using the Dynamic Platform Optimizer (DPO) utility.

An at-a-glance comparison between the POWER7 and the POWER8 processors can be seen in Table 1.

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

Because of the slowdown in frequency increases from silicon technology, thread and core performance were improved through micro-architectural 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
- a significant reduction in memory latency relative to the POWER7 processor design

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**.

- This knocked my socks off!! This means a vCPU in SMT-2 mode from four different LPARs can run **concurrently** on a given POWER8 core.
- In other words, four LPARs can have a vCPU in SMT-2 mode running on the same POWER8 core at the same time.
- For tuning, it means vCPU time-slice fragmentation (the 4th Dimension of Affinity) could eventually be partially mitigated by use of split-core mode.
- This POWER8 capability is also called "micro-threading".
- I believe this is only offered with PowerKVM, i.e. redbook:IBM PowerKVM Configuration and Use (sg248231.pdf Oct 2014)

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.**

The IBM 4764 PCI-X Cryptographic Coprocessor was withdrawn in 2011.

Overview

The IBM PCI-X Cryptographic Coprocessor provides a high-security, high-throughput cryptographic subsystem. The tamper-responding hardware is validated at the highest level under the stringent FIPS PUB (Federal Information Processing Standards Publication) 140-2 standard. Specialized hardware performs AES, DES, TDES, RSA, and SHA-1 cryptographic processes, relieving the main processor from these tasks. The coprocessor design protects your cryptographic keys and sensitive custom applications. The software running in the coprocessor can be customized to meet special requirements.

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.

- The trend of POWER engineering is clearly about keeping more data closer with 2 times larger **L1 data cache** and 2 times larger **L2 and L3 caches**.
- Having **twice as many ports from that data cache for higher read/write throughput** means moving more data faster to the 16 execution pipelines.
- Having **four times as many entries in its TLB (Translation Lookaside Buffer)** means cache-speed address translations. The TLB is a cache of address translations. Any TLB miss means searching the hardware page table (HPT; residing on slower main memory) for the missed address translation.
- **Keeping Tight&Fat on one socket substantially improves all of the above.**

Optimally exploiting the POWER8 core microarchitecture

- The best way to exploit POWER8/AIX is to first understand your workload with meaningful POWER/AIX tactical monitoring, i.e. “knowing it by the numbers”.
- Open invitation: Write to me for my script and I promise to assist in this regard.

- Optimal exploitation doesn’t always mean the fastest performance/throughput
- It may mean the most productivity-per-CPUcore or productivity-per-license
- It may mean the quickest responsiveness at the expense of wasted CPUcycles

- Sometimes you just care to understand what your workload is doing better
- Sometimes you want to learn what else can be done to improve a situation

- Other times there is a crisis and you urgently need another willing set-of-eyes
- Try me: Call or write and I will do what I can to help. **Really truly, I’m real.**
- I work for IBM Lab Services and Training; we’re a different delivery practice

IBM Systems Lab Services, U.S. Power AIX & LoP Offerings

Stephen Brandenburg – sbranden@us.ibm.com -OR- Linda Hoben – hoben@us.ibm.com -OR- Michael Gordon – mgordo@us.ibm.com

Power Workshops

- Power8 Transition Workshop
- NEW! Power8 Provisioning Assurance
- Power/AIX Monitoring and Tuning (not using NMON)
- IBM Power & Storage Planning for Disaster Recovery Workshop
- Data Center Availability Assessment

Power (VM) Virtual Management

- PowerVM HealthCheck / Best Practices Review
- PowerVM Customized Training (NPIV, LPM, AMS/AME, etc.)
- PowerVM Provisioning Toolkit (with NEW! "Capture" capability)
- IBM Proactive Monitoring for AIX & VIOS ("ProMon")
- NEW! PowerVM LPM Automation Tool (from China Labs team)
- Power Enterprise Pools Enablement
- WPAR (Workload Partition) Implementation and/or Migration

Power (HA) High Availability

- PowerHA Customized Training
- PowerHA SE Implementation / HealthCheck
- PowerHA & Storage Implementation for Disaster Recovery
- PowerHA EE Implementation for DR (incl. Toolkit "Capture")

Power Performance

- Power Virtualization Performance (PowerVP)
- Performance Optimization Assessment (POA)
- Oracle on AIX Performance Assessment Services
- Capacity Planning Tool (CPT) installation & configuration
- DB2 on AIX Application Performance Assessment (DB2 BLU)
- IBM Tivoli Monitoring (ITM) support for clients with AIX EE

Big Data Enablement

- IBM Power Analytics Infrastructure Enablement (DNS, DHCP, email, virus scanning, file sharing, etc.)
- BigData Assessment & Jumpstart Services
- Linux on Power BigData - InfoSphere BigInsights
- Linux on Power BigData - InfoSphere InfoStreams
- Linux on Power BigData - Executive Infrastructure Review

Linux

- Power IFL Implementation
- Linux on Power (LoP) Customized Training
- IBM PowerKVM
- Linux on PowerVM
- Linux on PowerVM Performance / HealthCheck
- Linux Workloads Assessment Workshop
- NEW! Linux on Power education offerings to expand customer training beyond the 1-day jump-starts from ATS
- Field Programmable Gate Array (FPGA) Development Platform RPQ

Cloud

- Cloud Design Workshop for custom cloud enablement
- PowerVC Implementation as a Pre-req to CMO
- Cloud Manager with OpenStack (CMO) Implementation
- Cloud IT Optimization Assessment
- Advanced Cloud on Power Services
- IBM Smart Analytics Optimizer Enablement
- SAP Landscape Virtualization Management Design & Planning Workshop
- SAP Landscape Virtualization Management Implementation
- Other "Open Stack" Consulting Services

SAP HANA on Power

- Installation and POC
- Health Check Assessment

Security

- IBM AIX Security Assessment
- LDAP Integration including Pass-through Authentication (PTA)
- IBM PowerSC™ Security & Automated Compliance Workshop
- PowerSC – Trusted Firewall Workshop (TWR)
- PowerSC – Trusted Surveyor Workshop (TS)
- AIX Auditing Workshop
- AIX Hacking Prevention Workshop
- Encrypted File System (EFS) Workshop
- Role Based Access Control (RBAC) Workshop/Support Services

Power/AIX Performance and Tuning Workshop (4.0-days on-site)

Overview:

This offering aims to grow and exercise the Power/AIX tactical skills of its attendees through lectures and lab sessions on their live-running AIX servers. The lectures describe the AIX Virtual Memory Manager, Power7 and Power8 Affinity, tactics for indicating performance issues, and remedial tactics to resolve these issues.

Throughout each lecture, the workshop illustrates its topics and tactics on the attendee's live-running Power/AIX LPARs as lab session exercises. As such, an incidental list of directly-observed and empirically-justified remedial tactics can be accumulated by each attendee as a by-product of the workshop.

The workshop is intended as a decidedly interactive venue. The attendee's questions are addressed immediately.

WHO benefits from this workshop and WHY ?

- Clients with Power6/7/8 servers with AIX 6.1-7.2 LPARs housing workloads.
- Clients who care to monitor their Power/AIX workloads by the numbers.
- Clients with workloads they suspect are not executing optimally but have been unable to determine what and why.
- Historically, Power/AIX system administrators, database administrators, application administrators, storage administrators, and IT architects have all learned more than they could imagine in a 4.0-day workshop.

Duration

- 28 to 32 hours (depending on the ability to absorb rigorous content)
- 6 to 8 hours per day (does not include an hour for lunch)
- Request a conference room with a PC projector
- Request an authorized staffer to "putty" into your Power/AIX LPARs

Delivery Details:

This is a customer onsite offering consisting of standup lectures and highly interactive lab sessions to your live-running LPARs. Presentation handouts are provided in PowerPoint format.

Lecture/Lab Session titles:

Part One: A Tactical Overview of Power/AIX Virtual Memory Manager mechanisms

Part Two: The Four Dimensions of Power7/Power8 Affinity

Part Three: How to use Power/AIX Historical/Cumulative Statistics to Indicate Performance Issues

Part Four: How to use Power/AXI Real-time Statistics to Indicate Performance Issues

Part Five: Remedial Tactics for Performance Tuning the Indicated Issues of Power/AIX Workloads

Part Six: IBM Power8 Processor Core Microarchitecture: Thoughts and Considerations

Terms and Conditions: Actual Tasks, Deliverables, Service Estimates,,and travel requirements vary with each client's environment. When we have reached a final agreement on the scope of your initiative and our level of assistance, a formal document describing our proposed work effort, costs, etc, will be presented for your approval and signature.

Linda Hoben – *Opportunity Manager* hoben@us.ibm.com 1-720-395-0556
Stephen Brandenburg – *Opportunity Manager* sbranden@us.ibm.com 1-301-240-2182

Organization of the POWER8 processor core

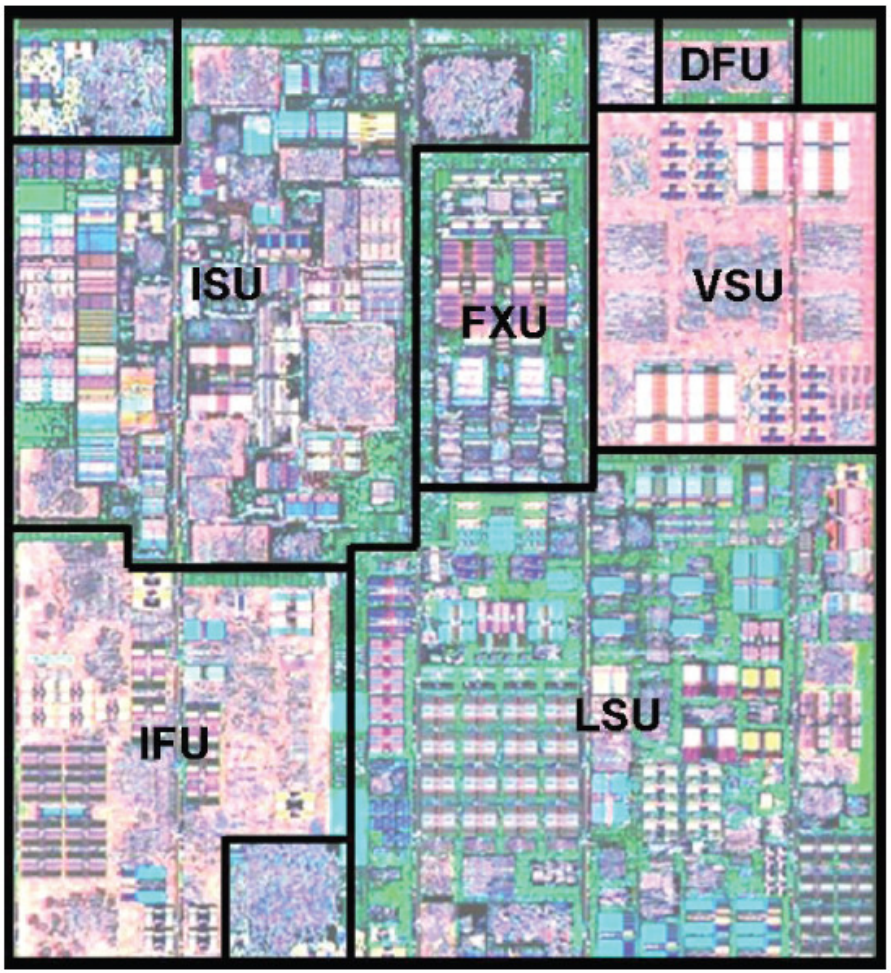

Figure 1

POWER8 processor core floorplan.

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
- one decimal floating-point pipeline

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.**

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.

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 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.

Figure 2 shows the instruction flow in POWER8 processor core.

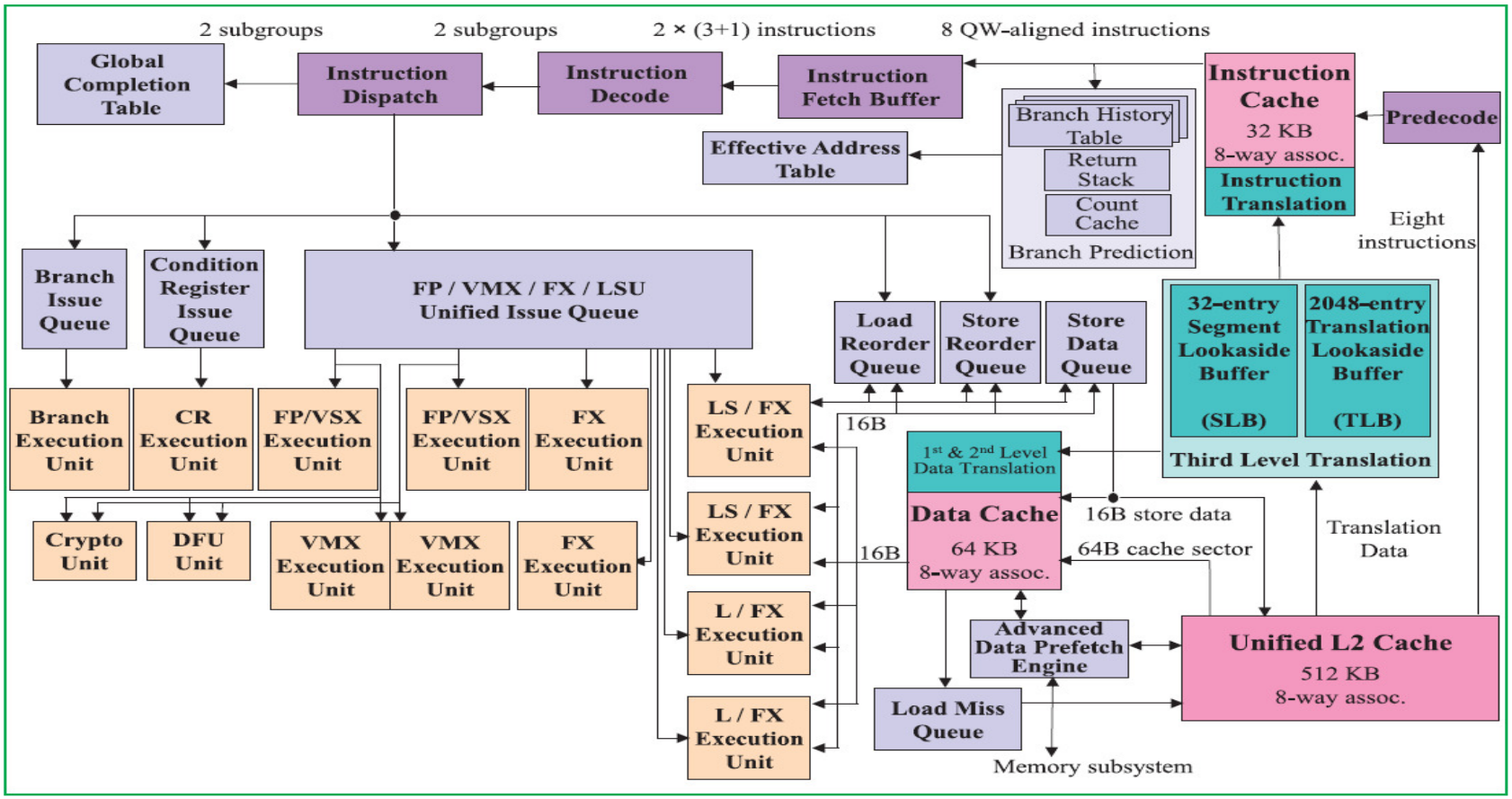

Figure 2

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

Instructions flow **from the memory hierarchy** through various issue queues and then are sent to the functional units for execution.

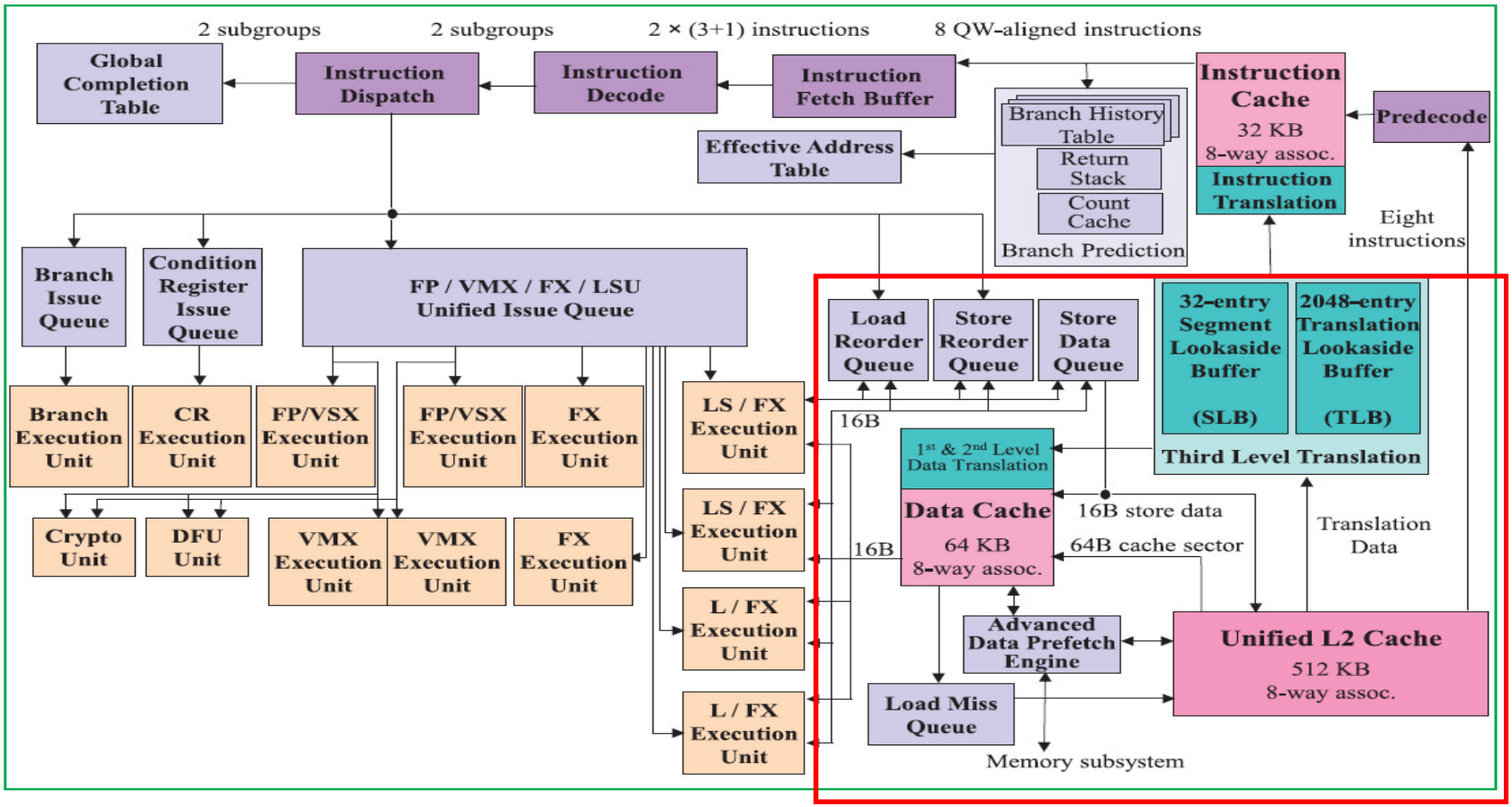

Figure 2

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

Instructions flow from the memory hierarchy **through various issue queues** and then are sent to the functional units for execution.

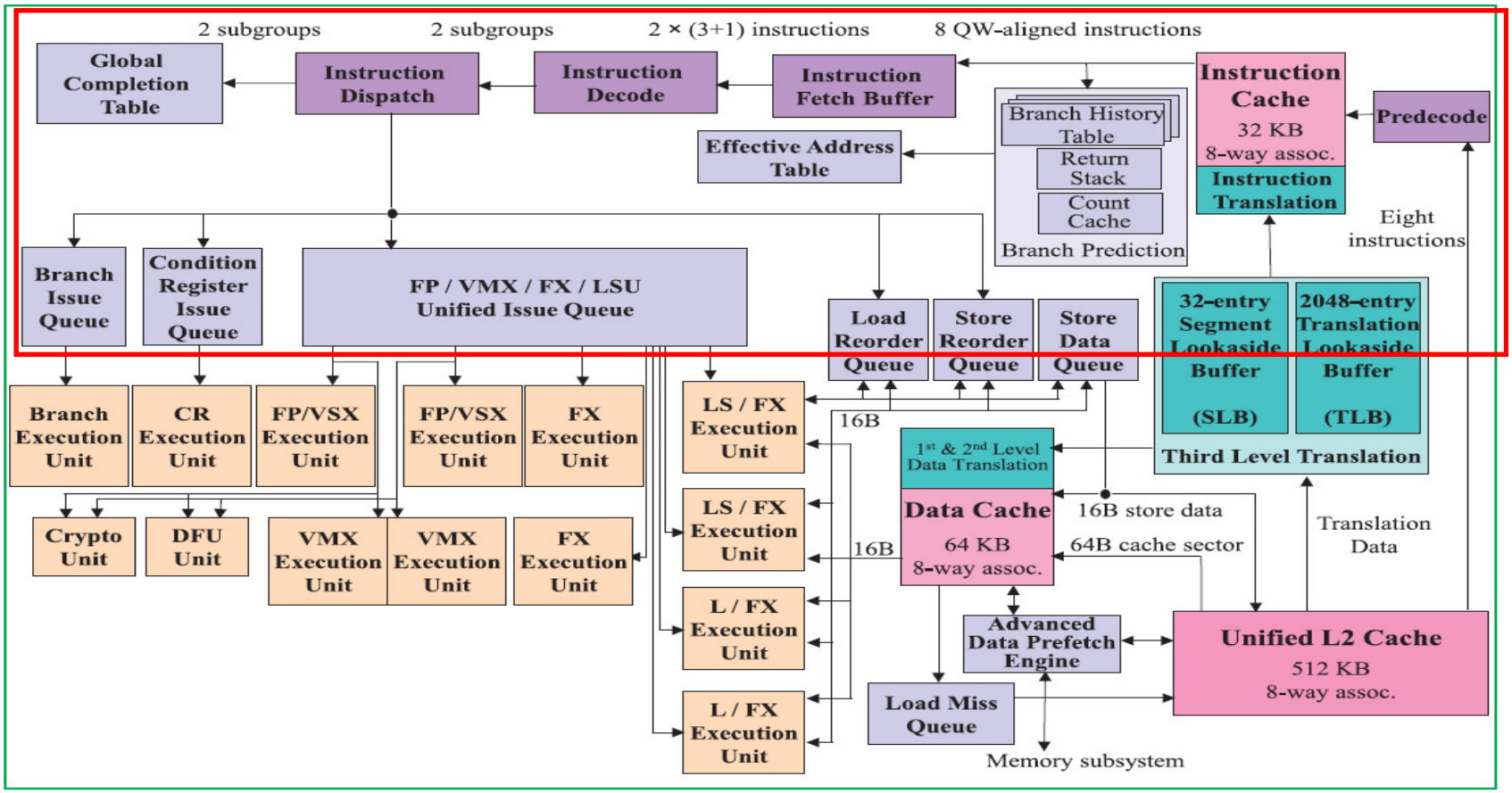

Figure 2

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

Instructions flow from the memory hierarchy through various issue queues and **then are sent to the functional units for execution**.

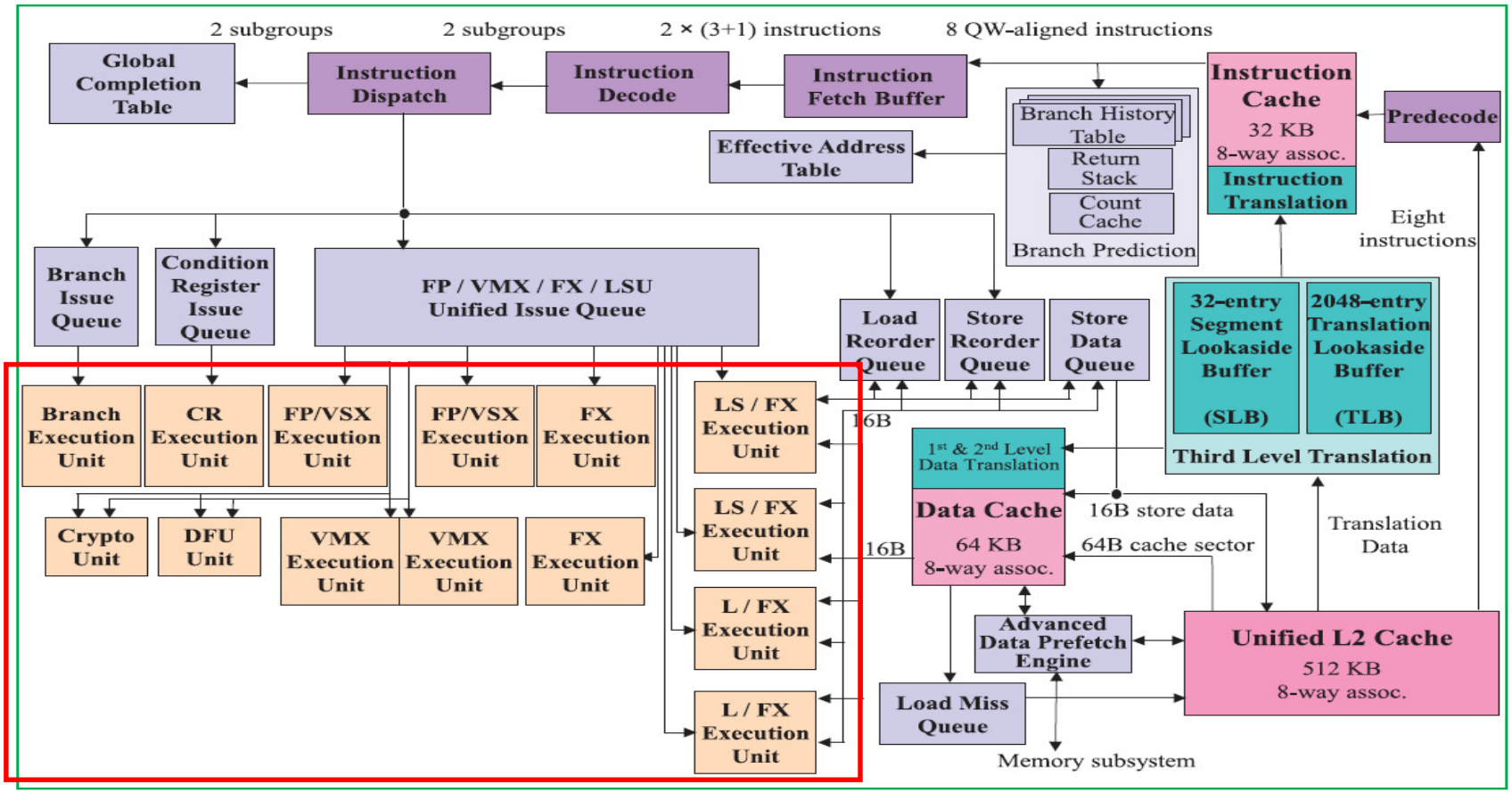

Figure 2

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

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).

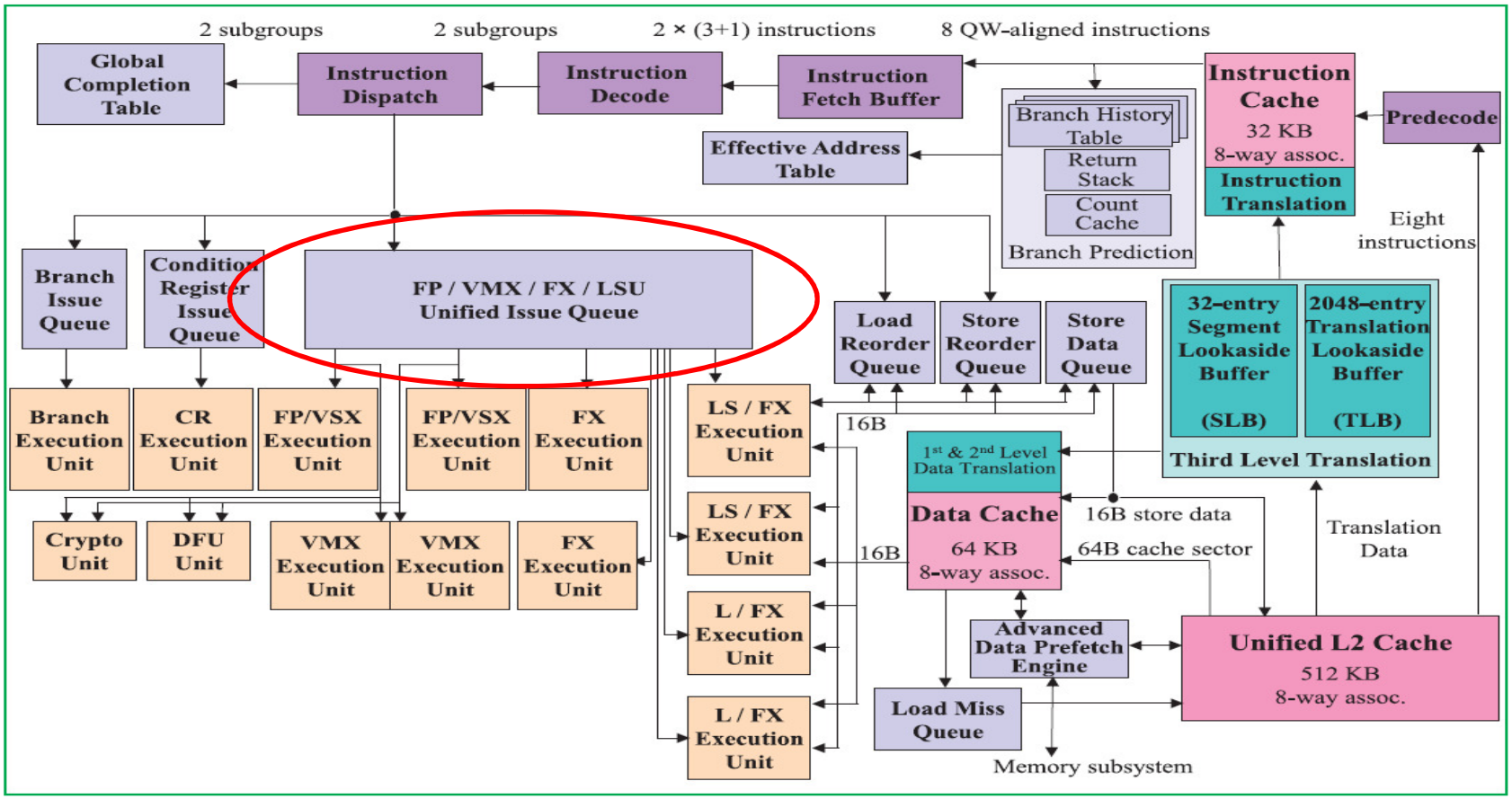

Figure 2

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

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.

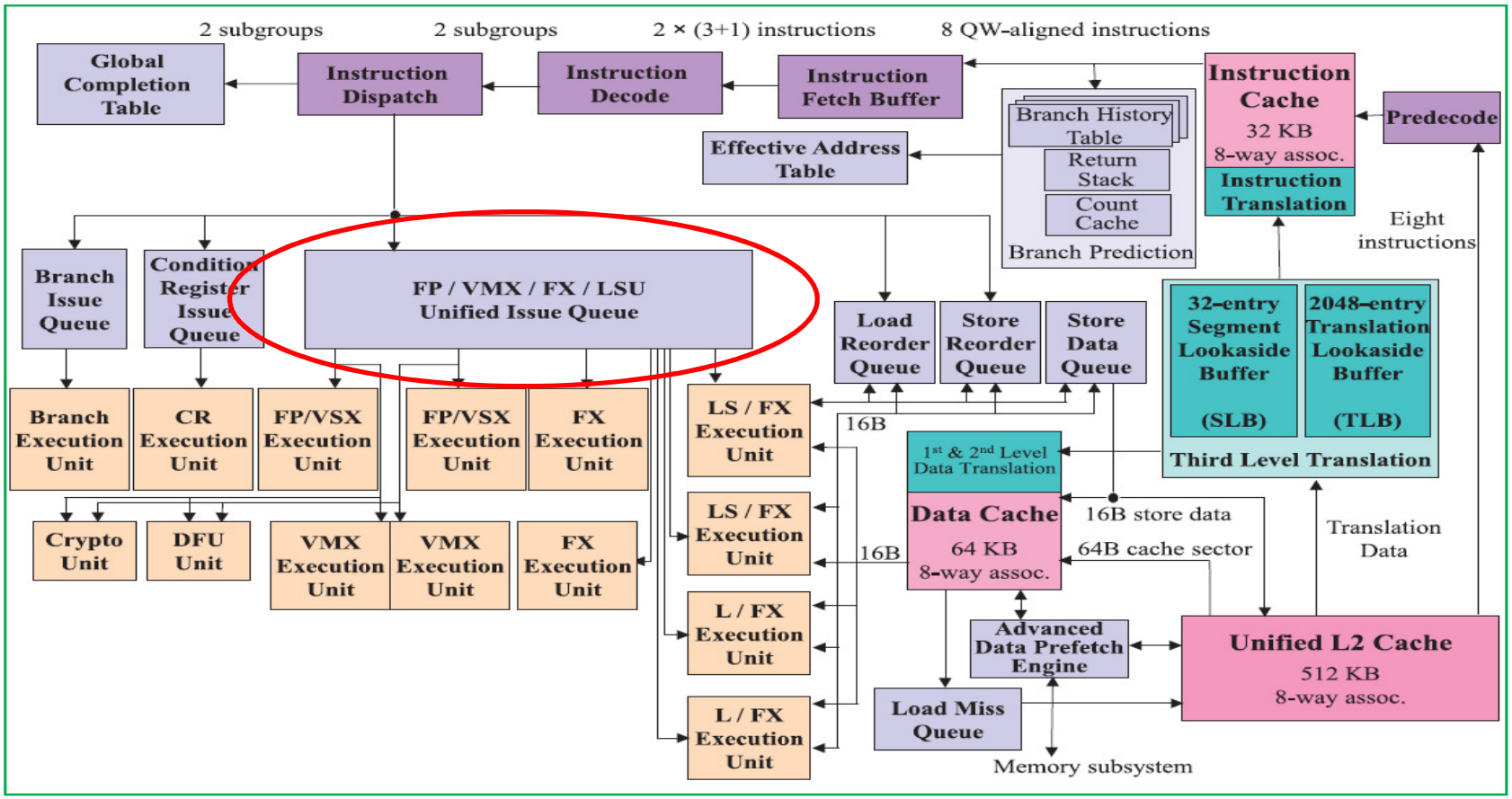

Figure 2

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

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.**

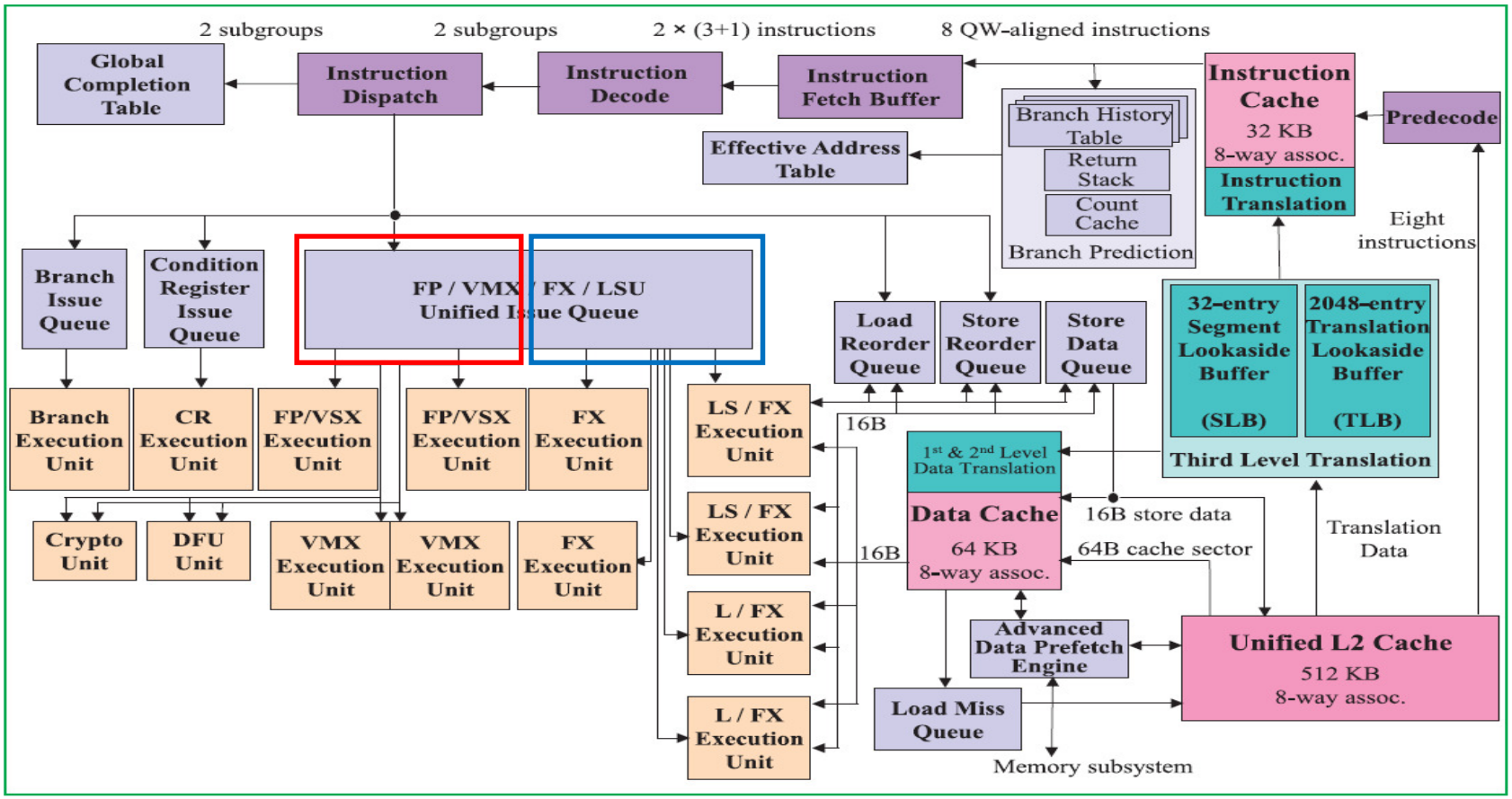

Figure 2

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

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.

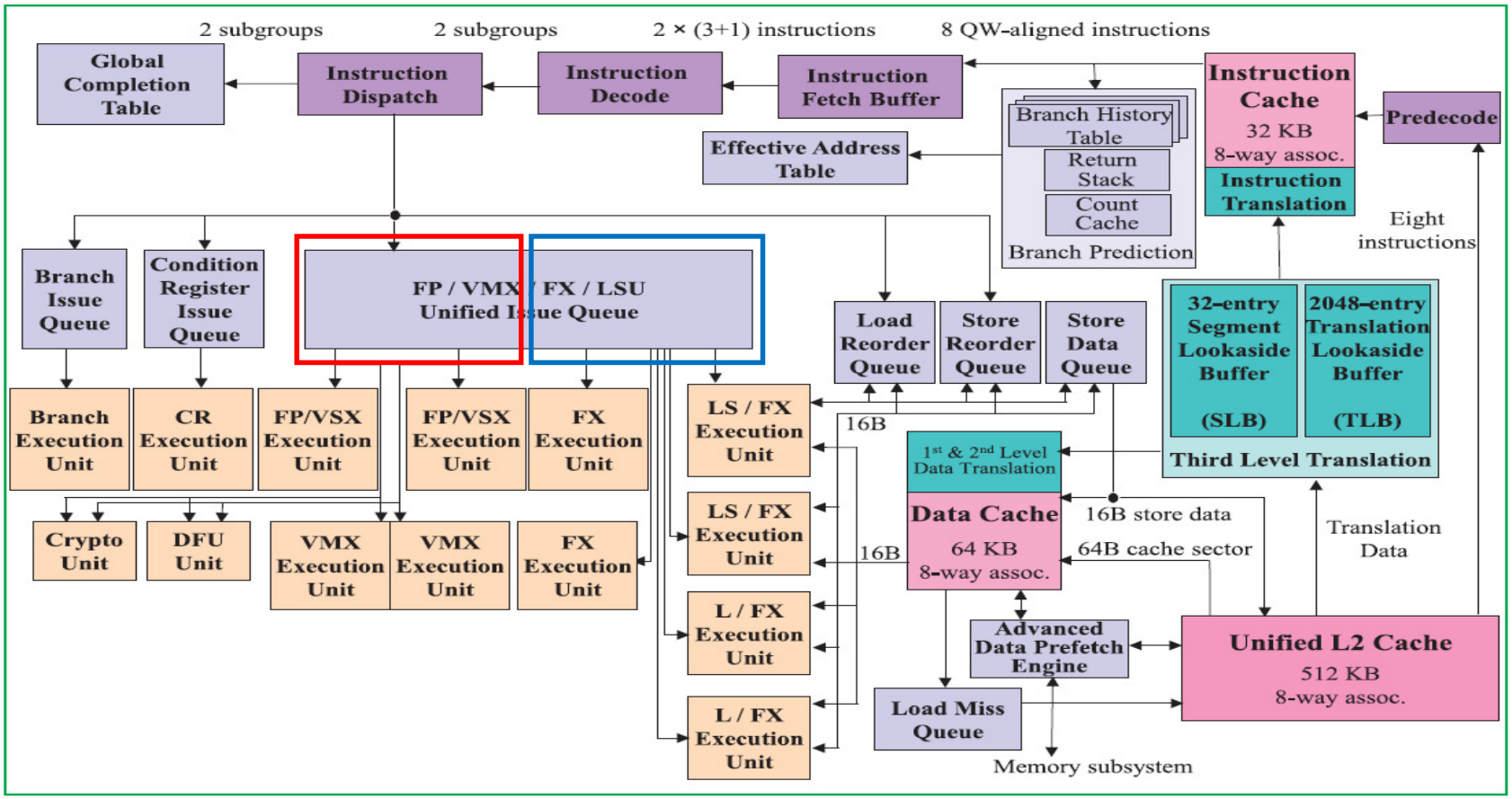

Figure 2

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

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**.

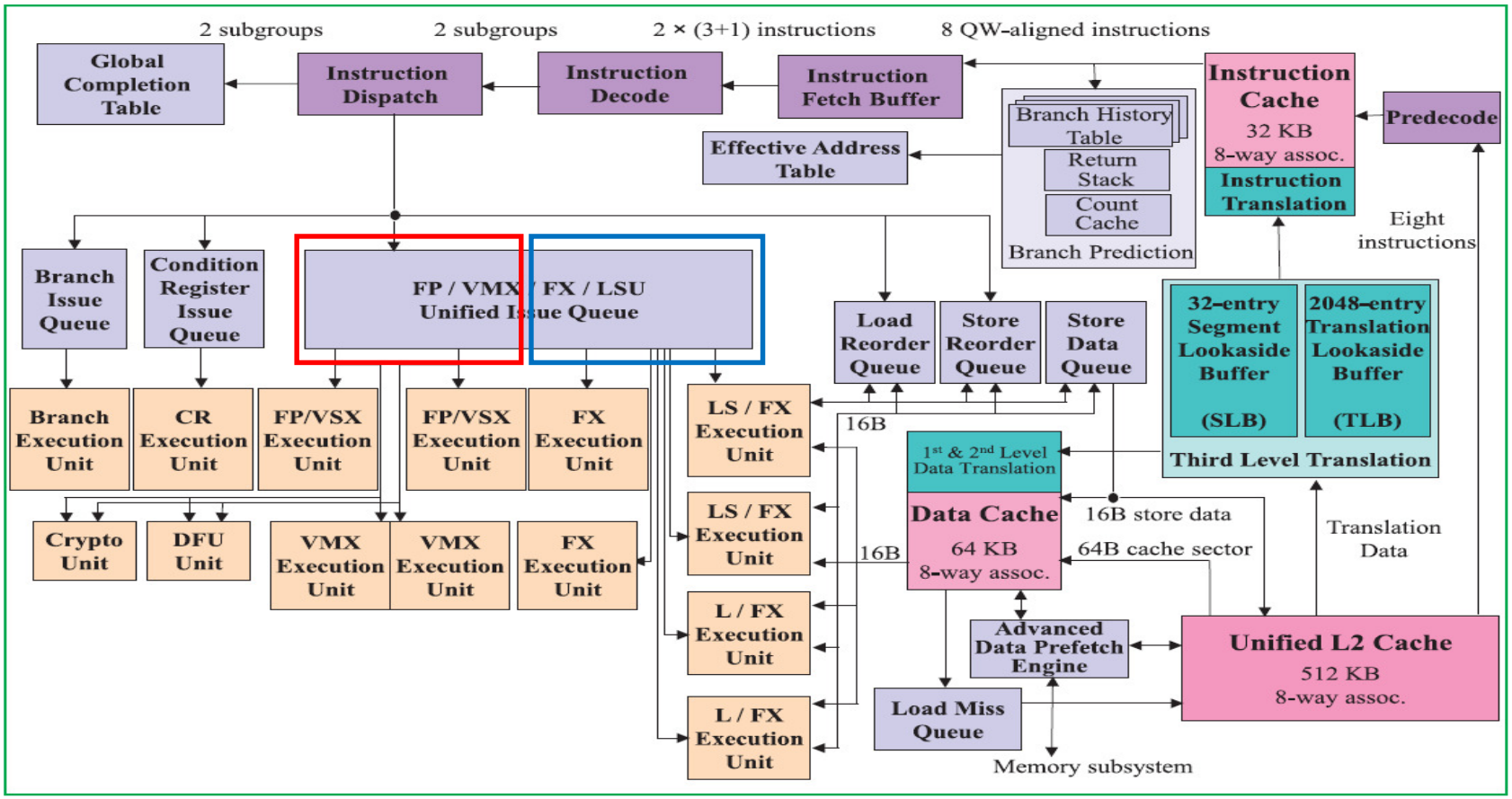

Figure 2

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

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.**

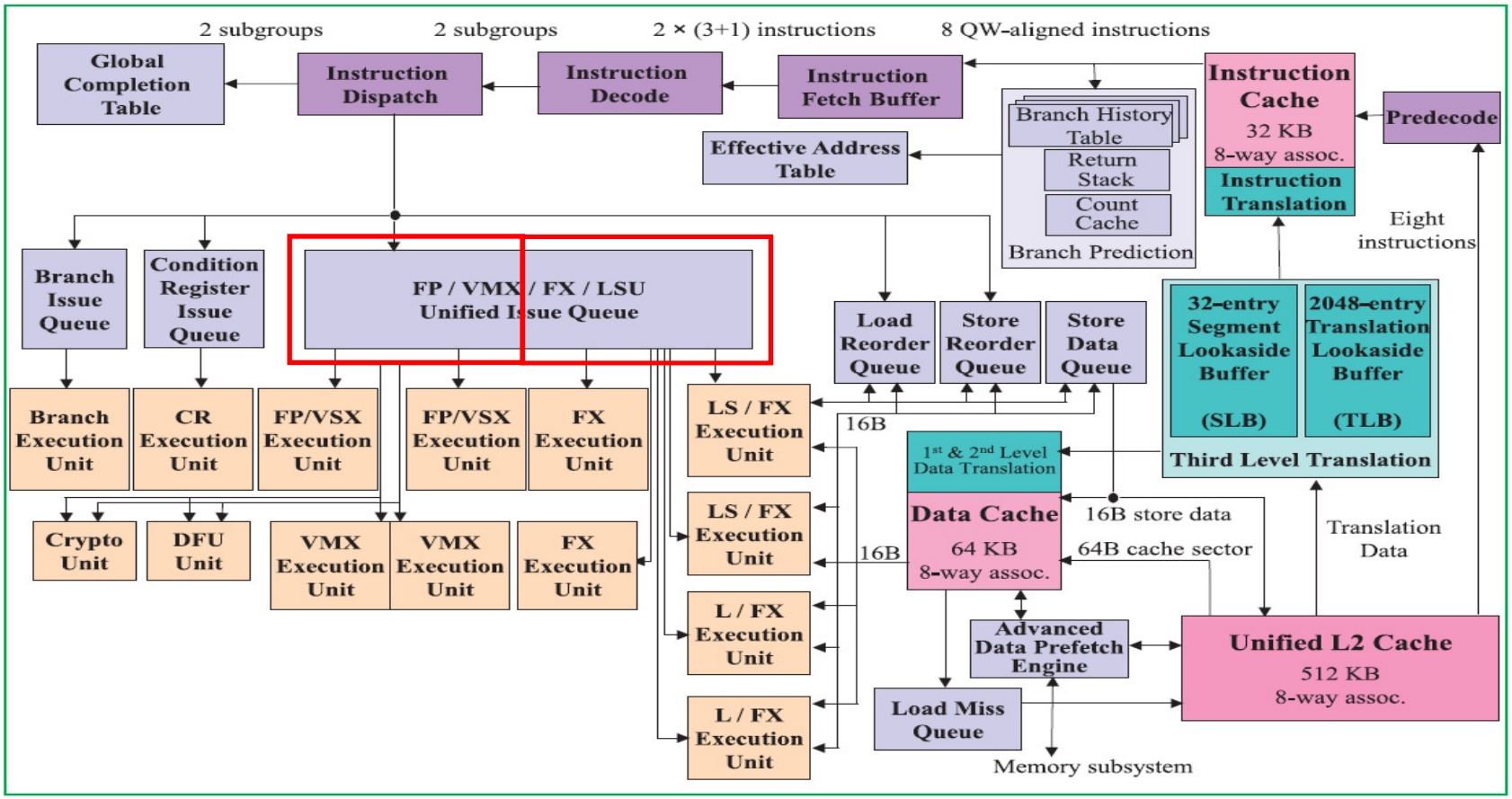

Figure 2

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

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.**

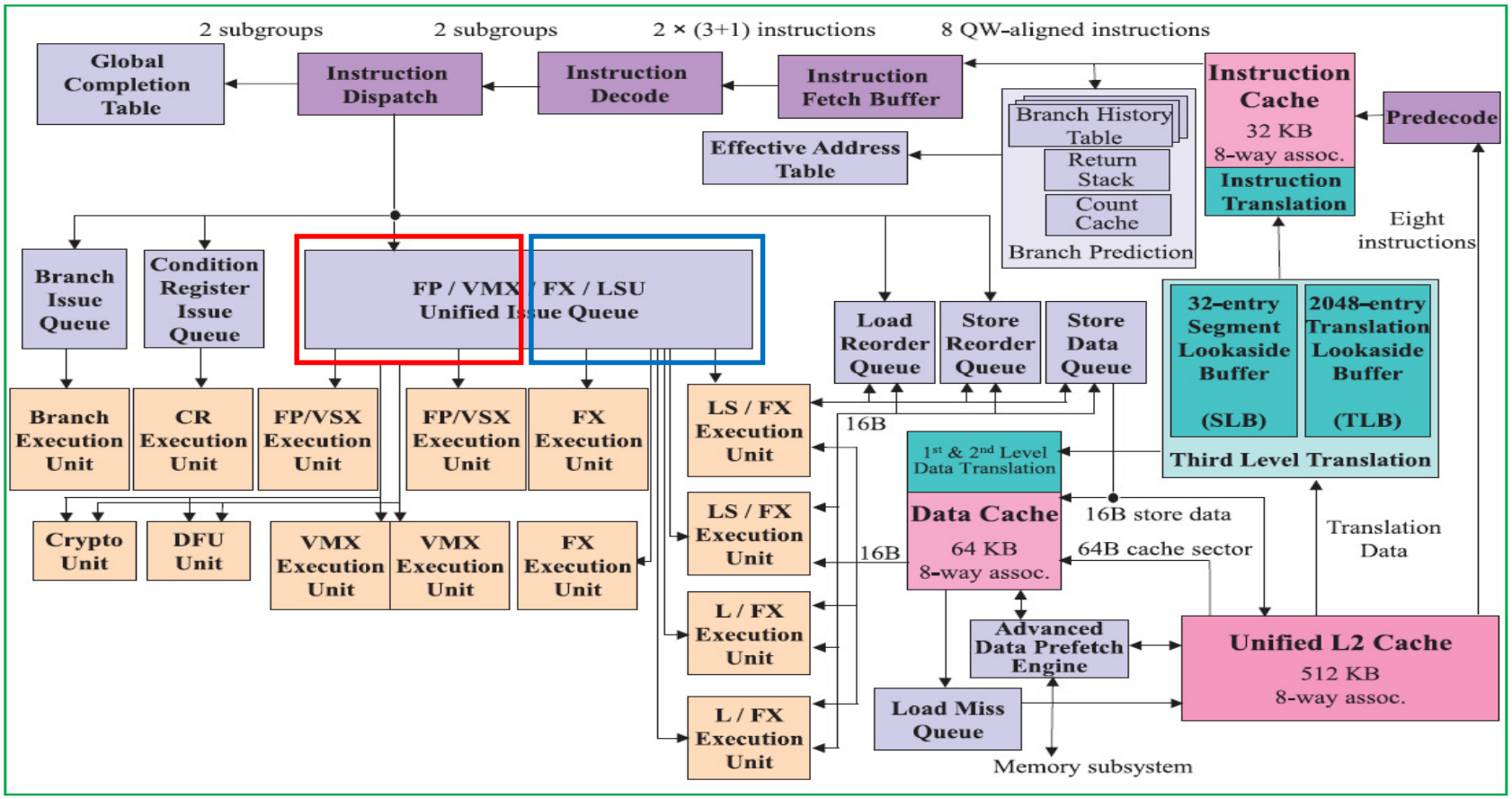

Figure 2

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

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.

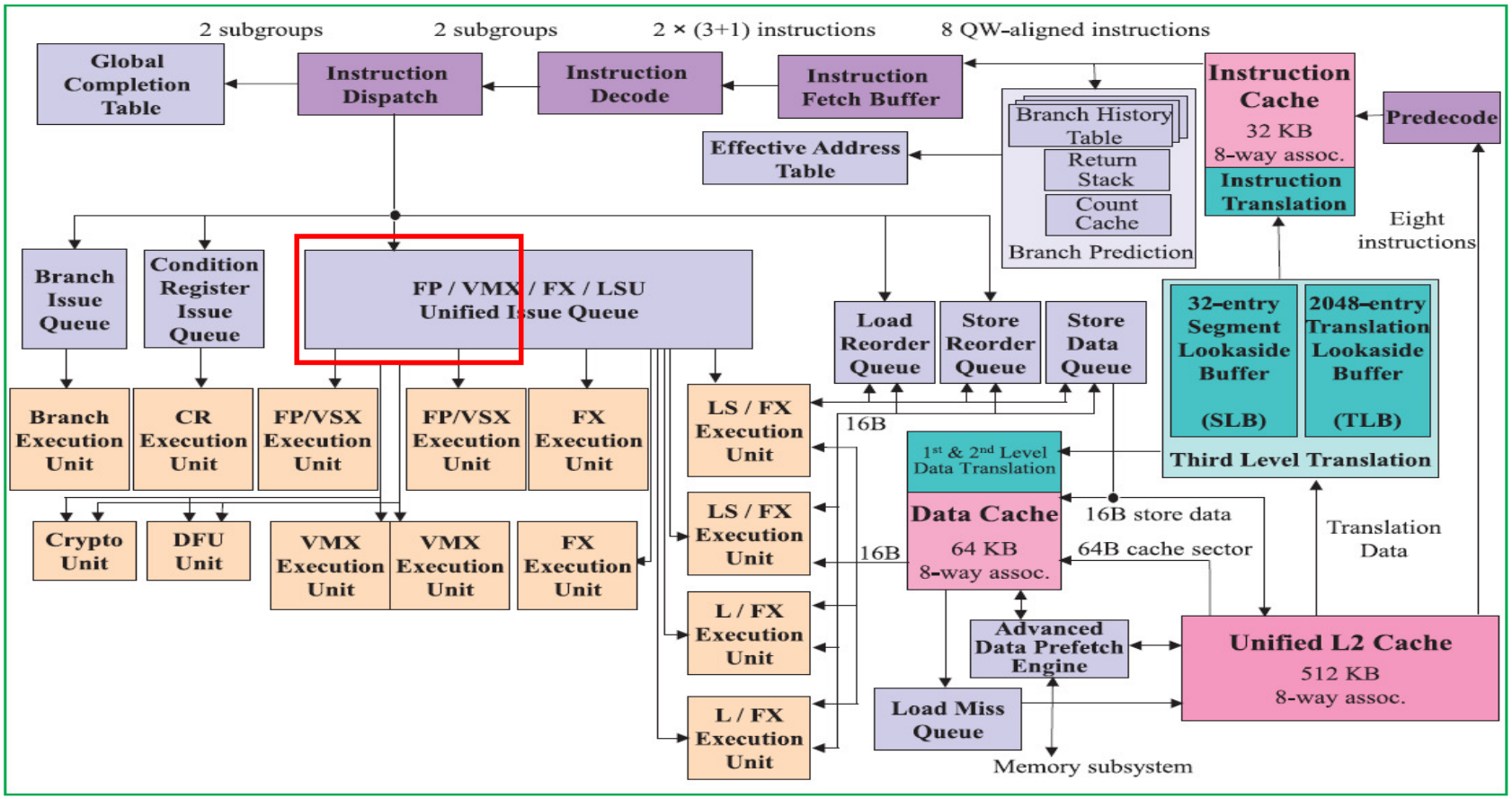

Figure 2

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

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.

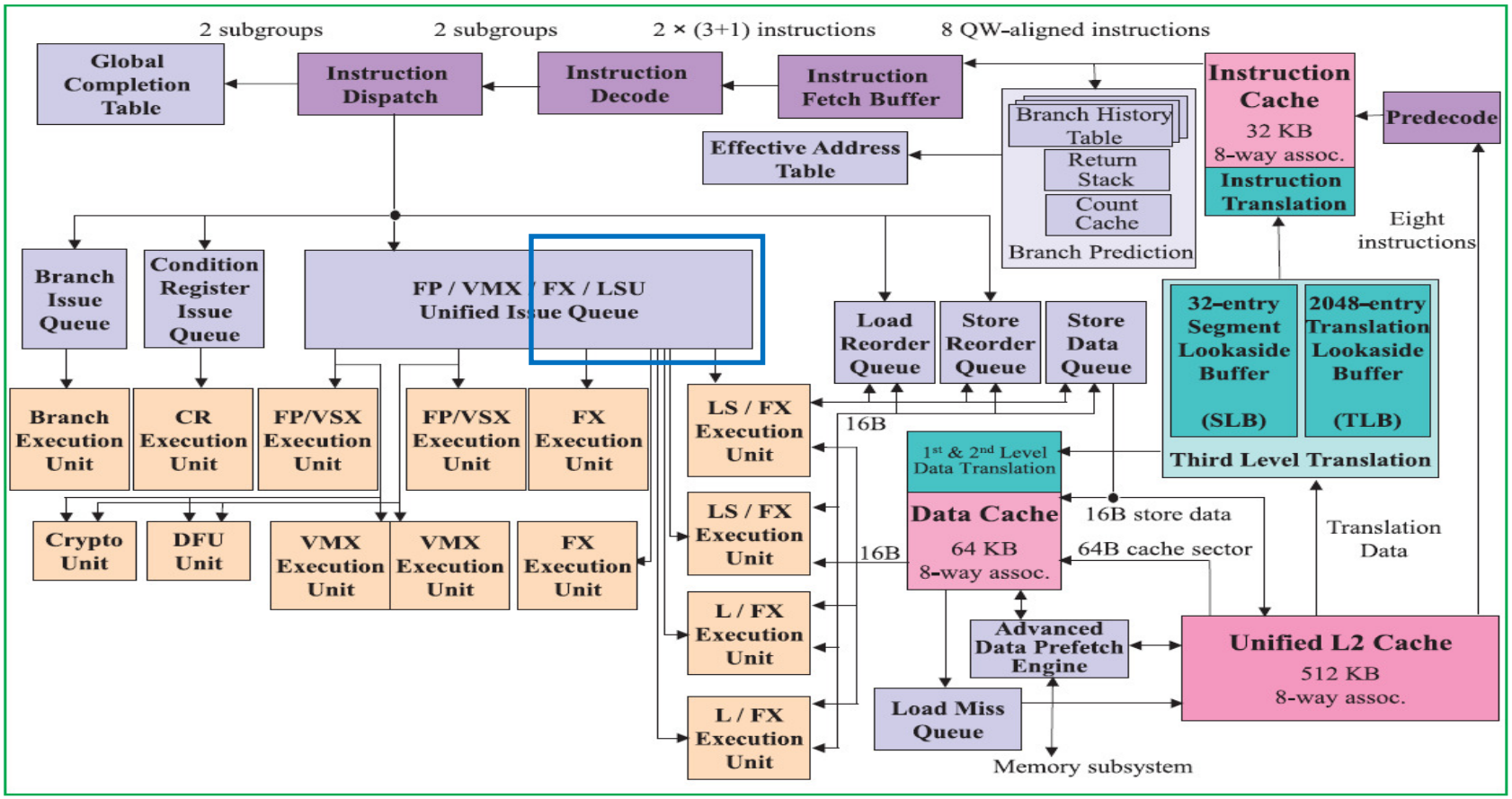

Figure 2

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

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.**

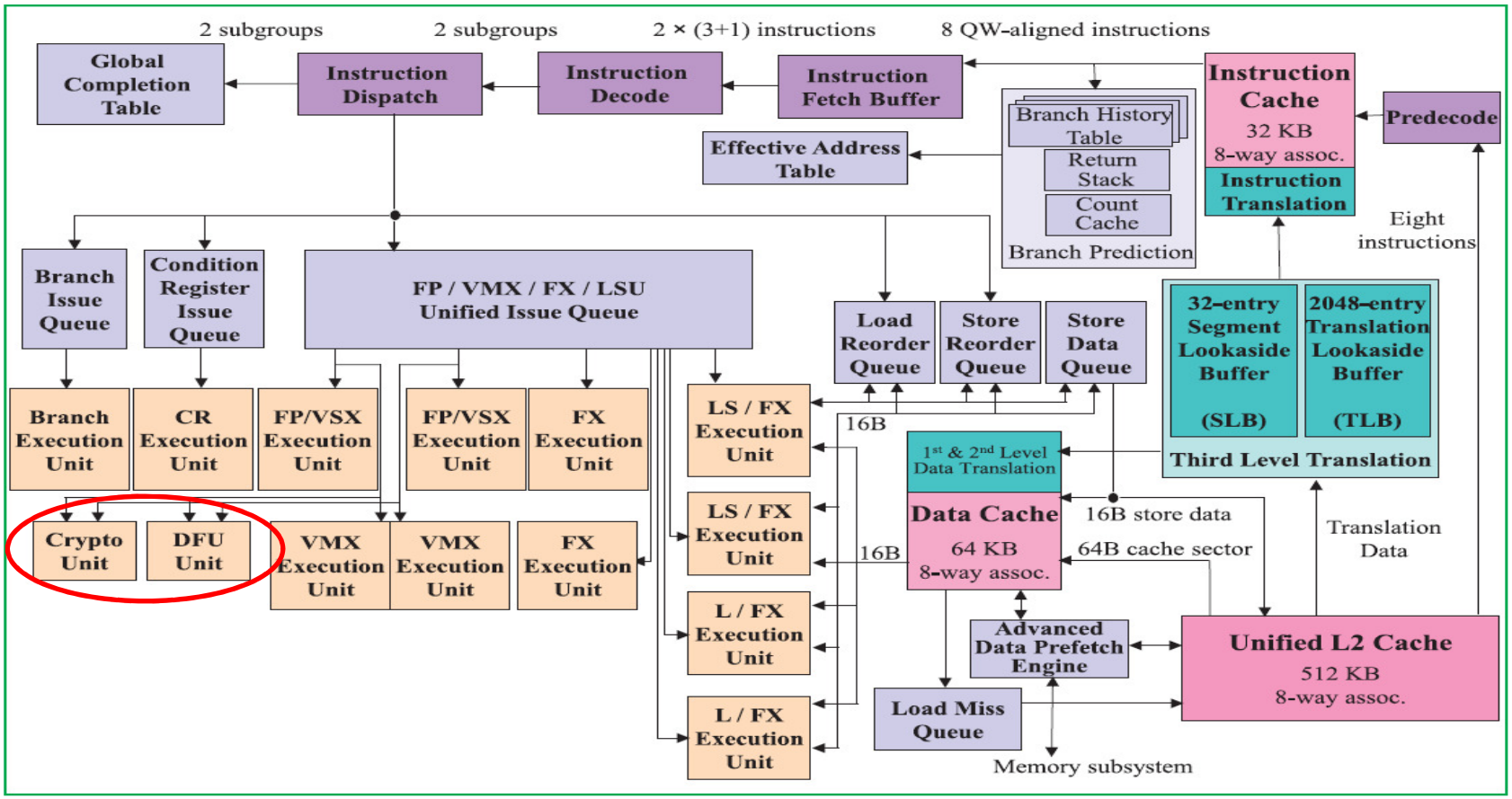

Figure 2

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

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

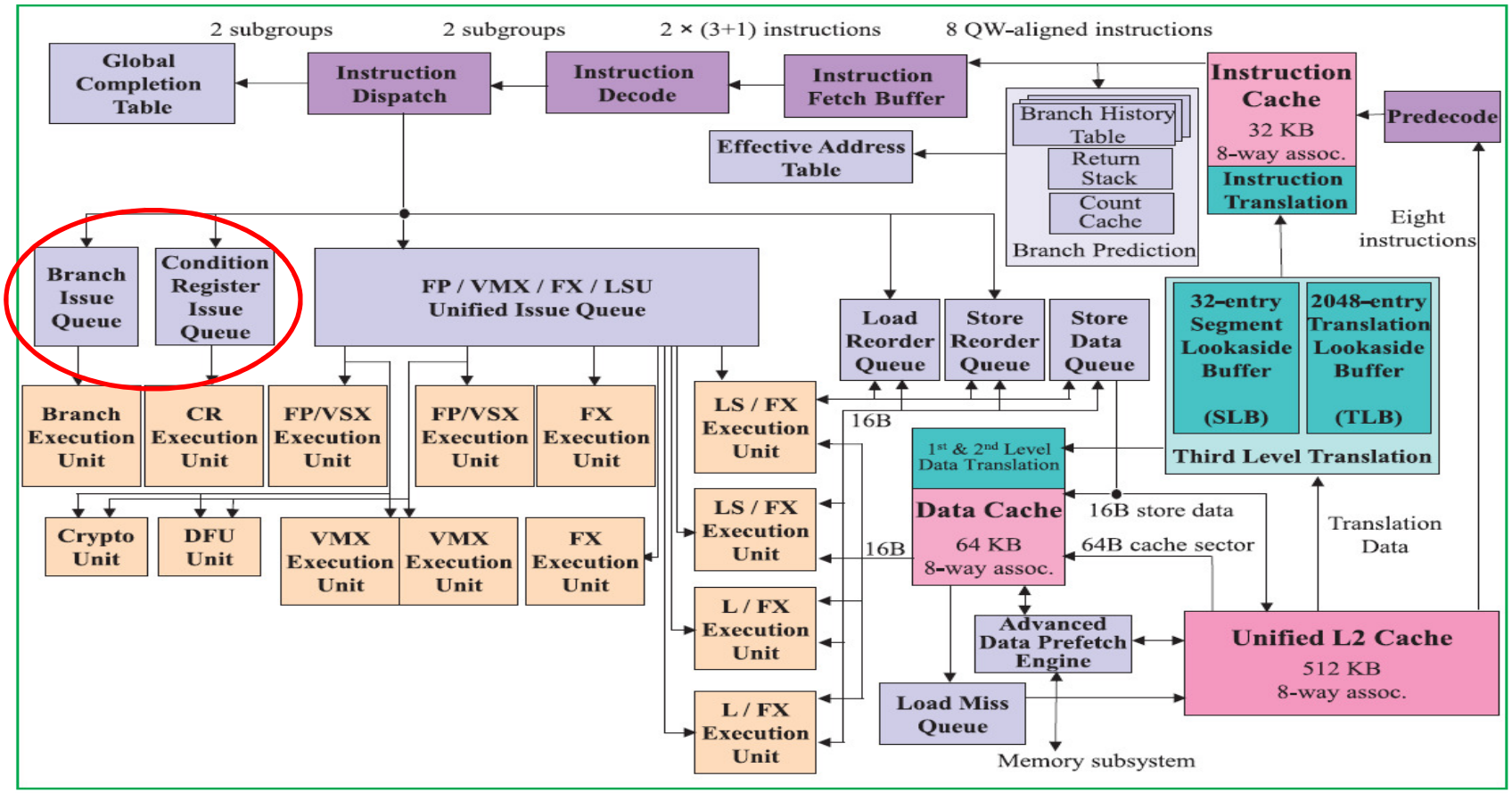

Figure 2

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

Instruction Fetch Unit (IFU)

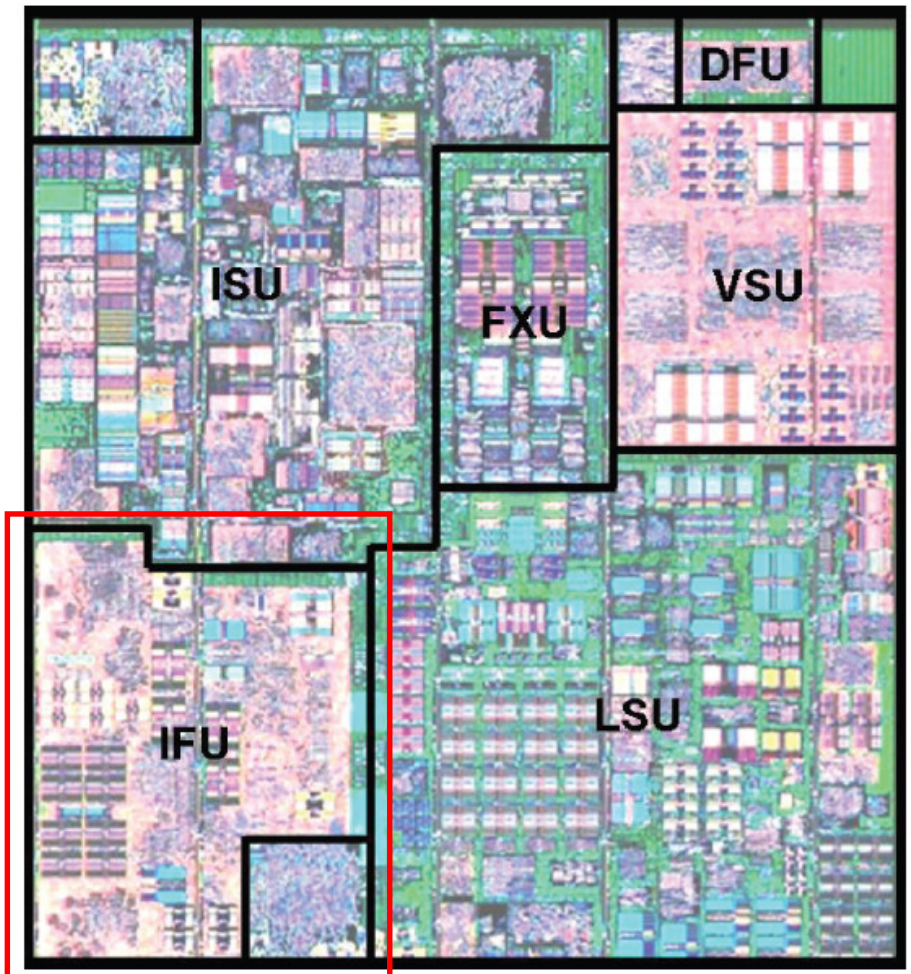

Figure 1

POWER8 processor core floorplan.

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.

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.

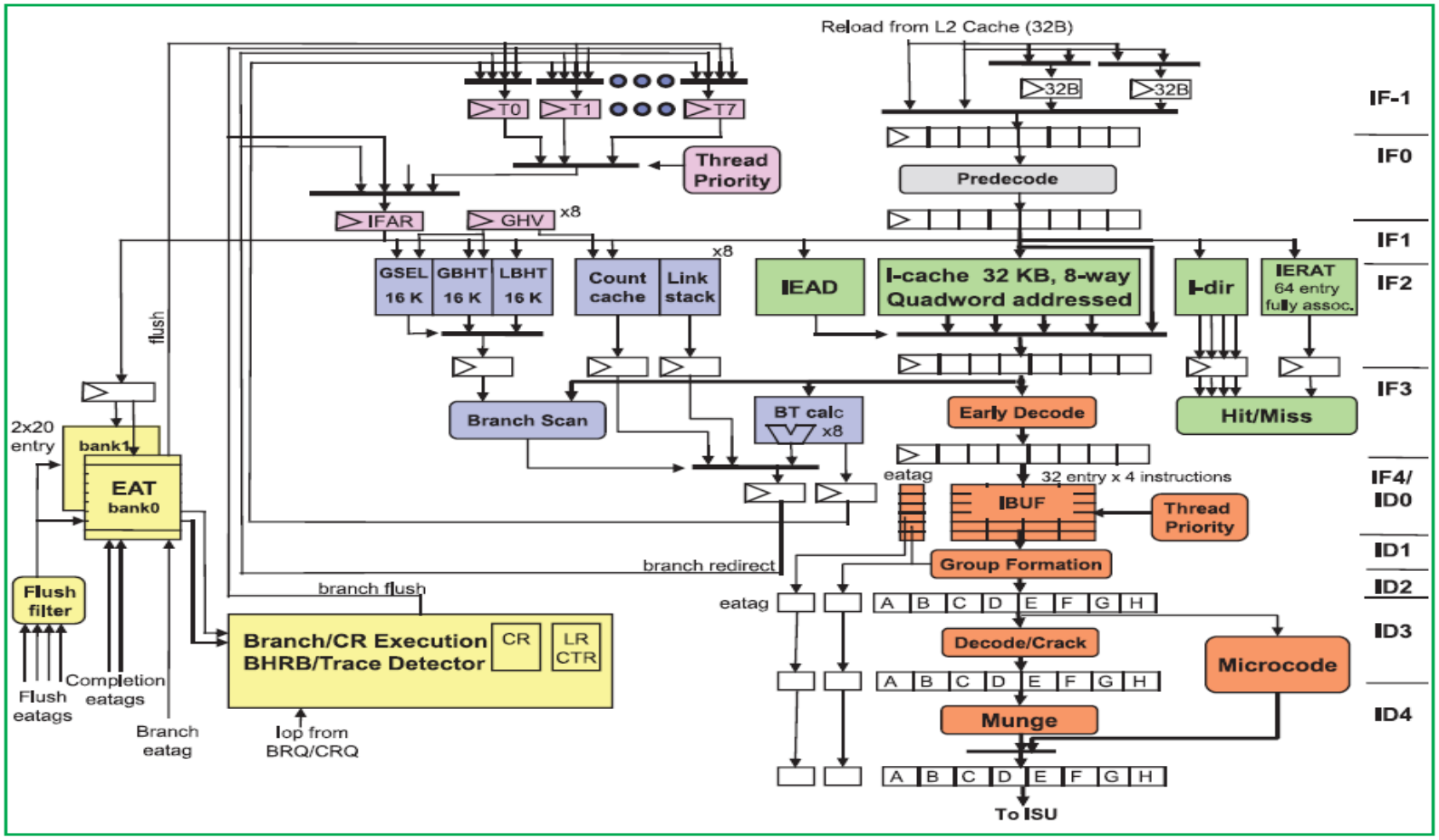

Figure 3

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.)

It uses branch prediction mechanisms to produce this stream well ahead of the point of execution of the latest committed instruction.

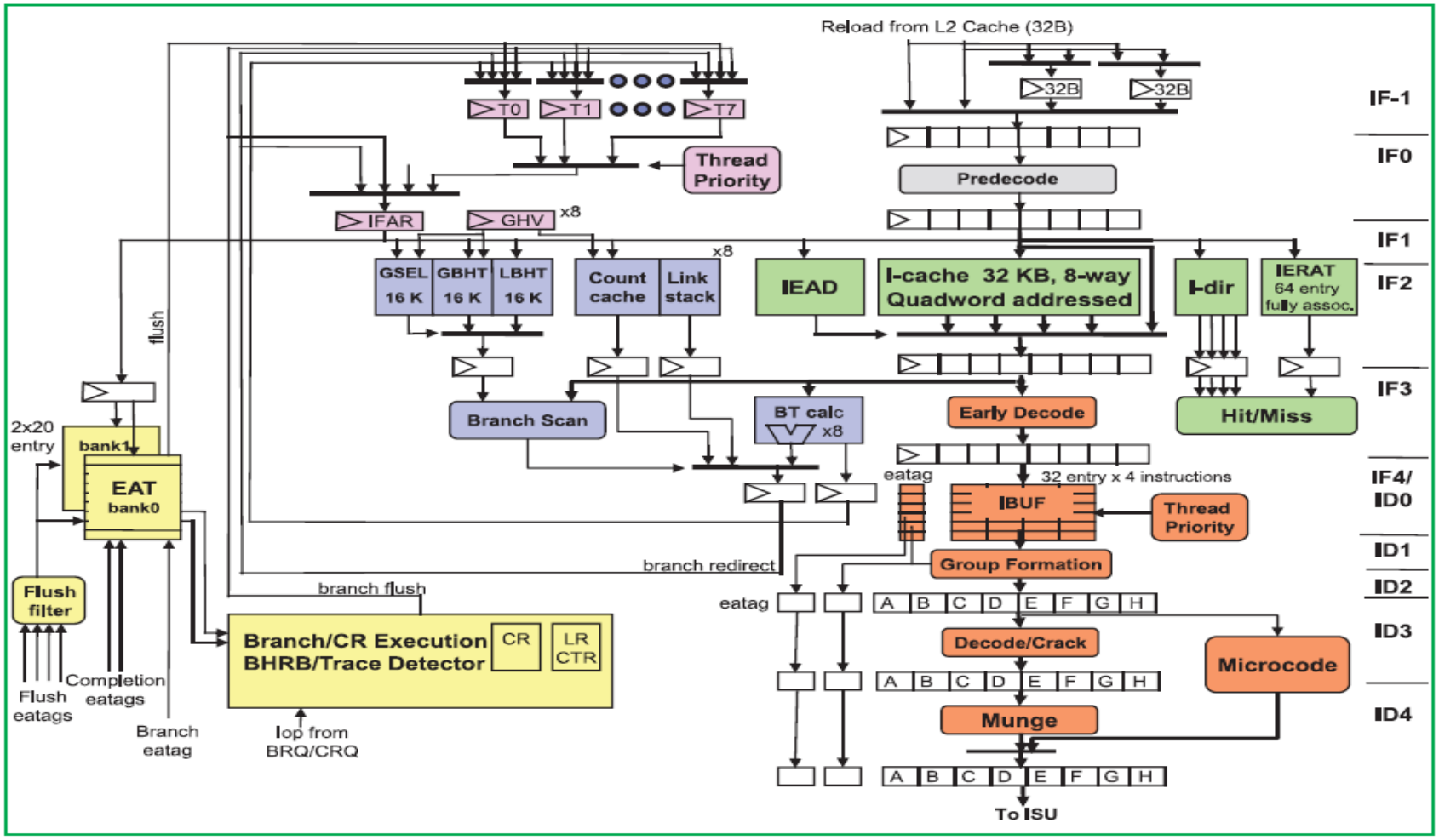

Figure 3

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.)

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**.

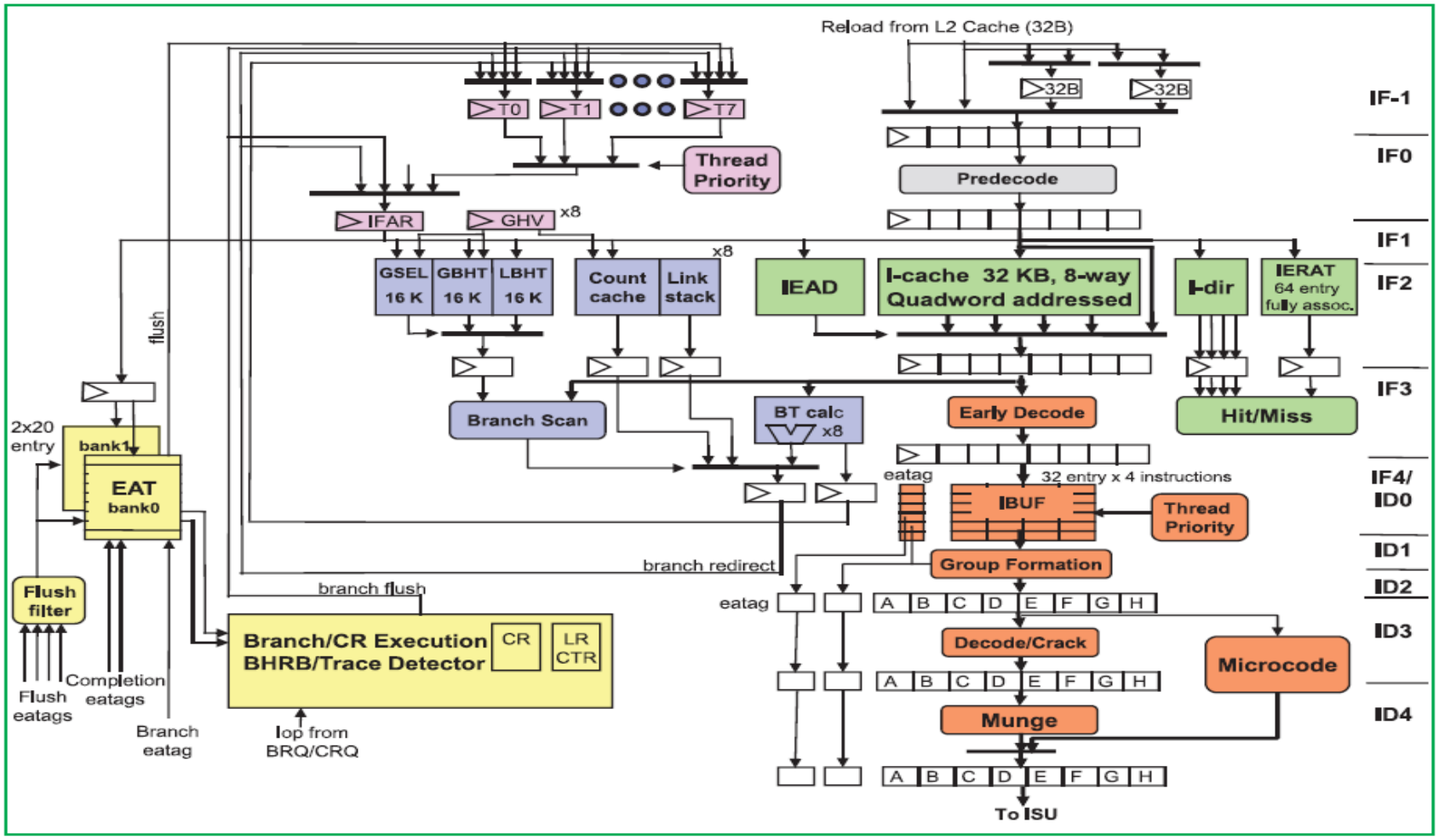

Figure 3

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.)

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.)

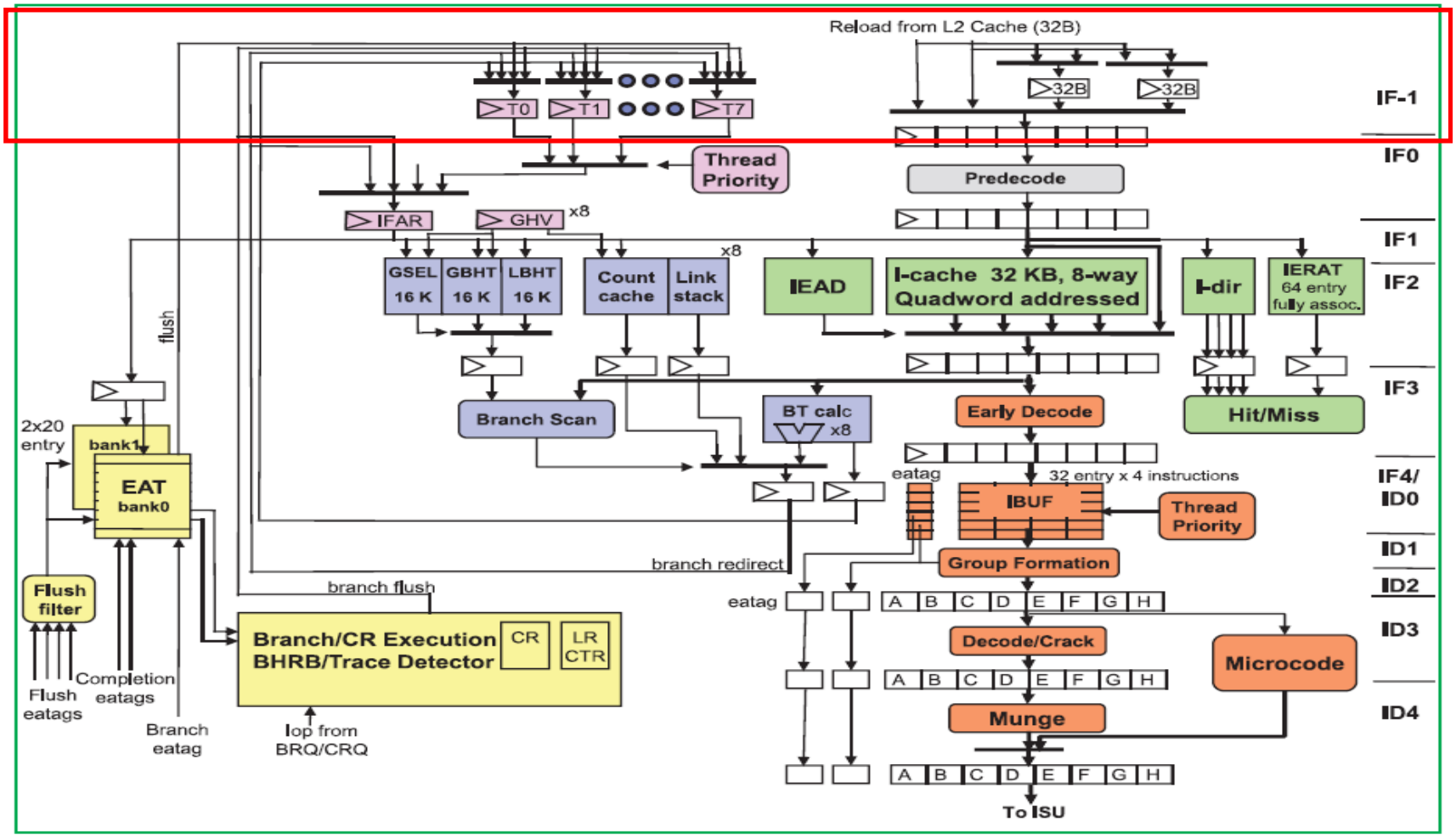

Figure 3

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.)

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.)

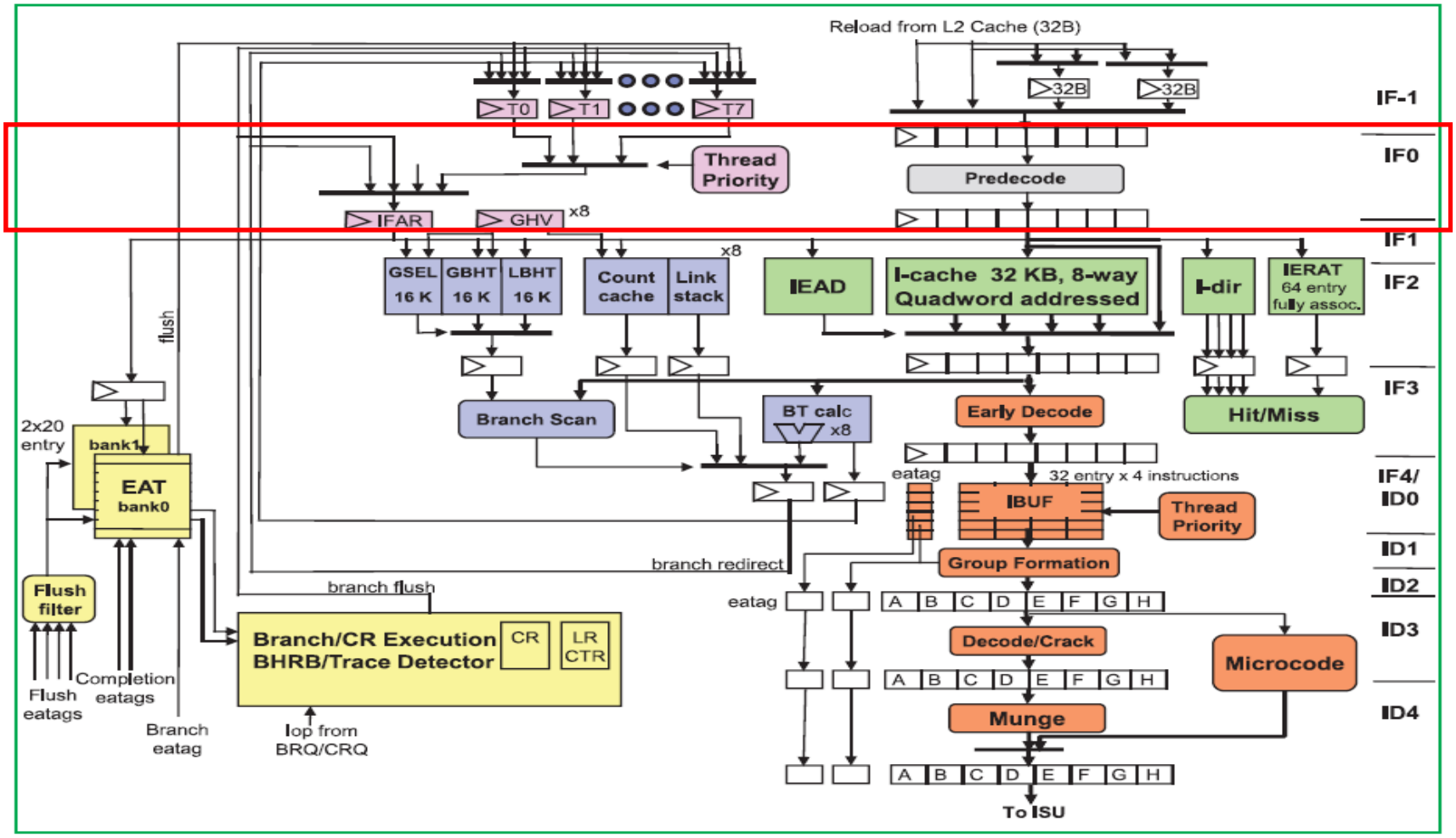

Figure 3

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.)

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.)

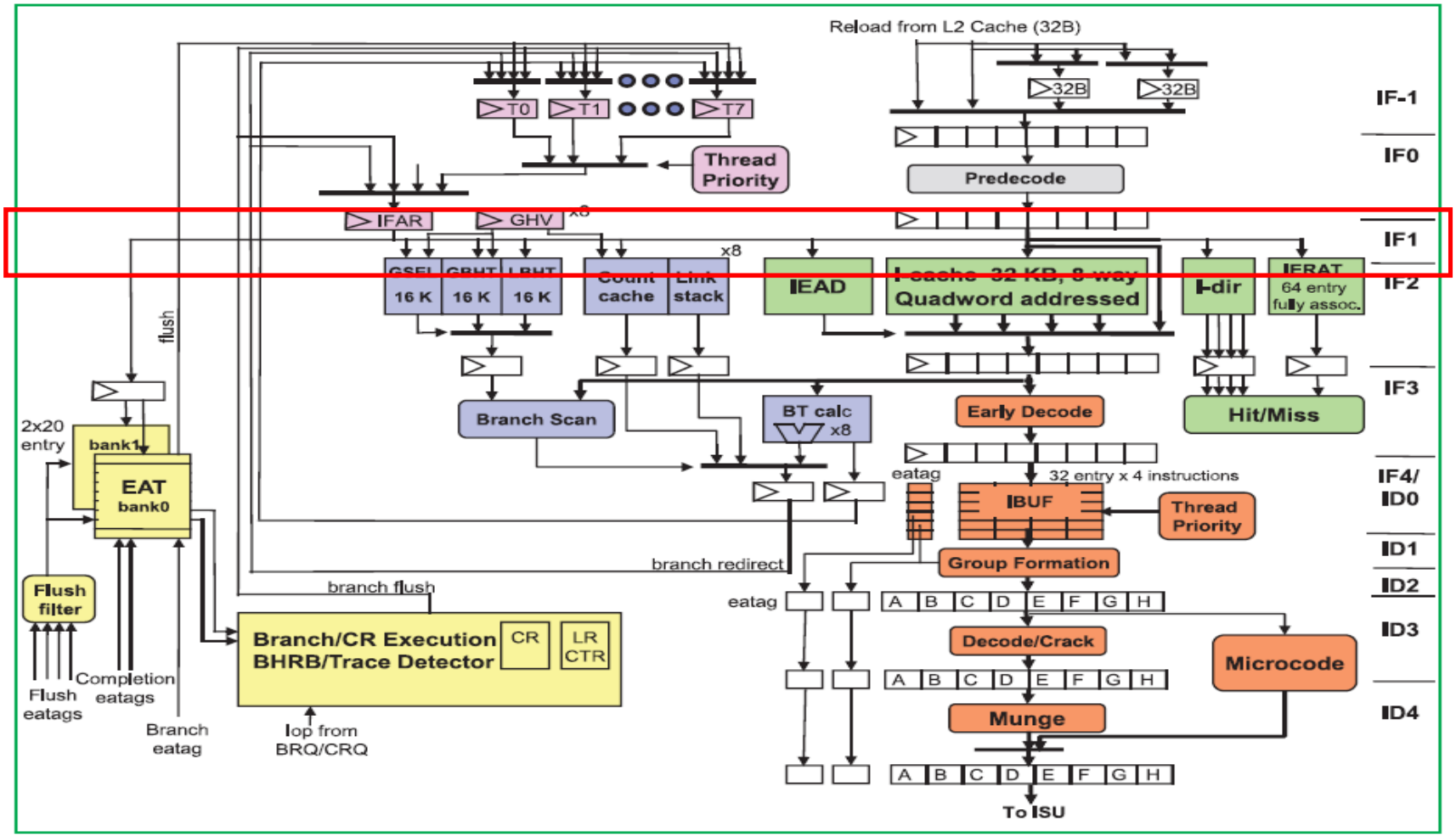

Figure 3

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.)

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.)

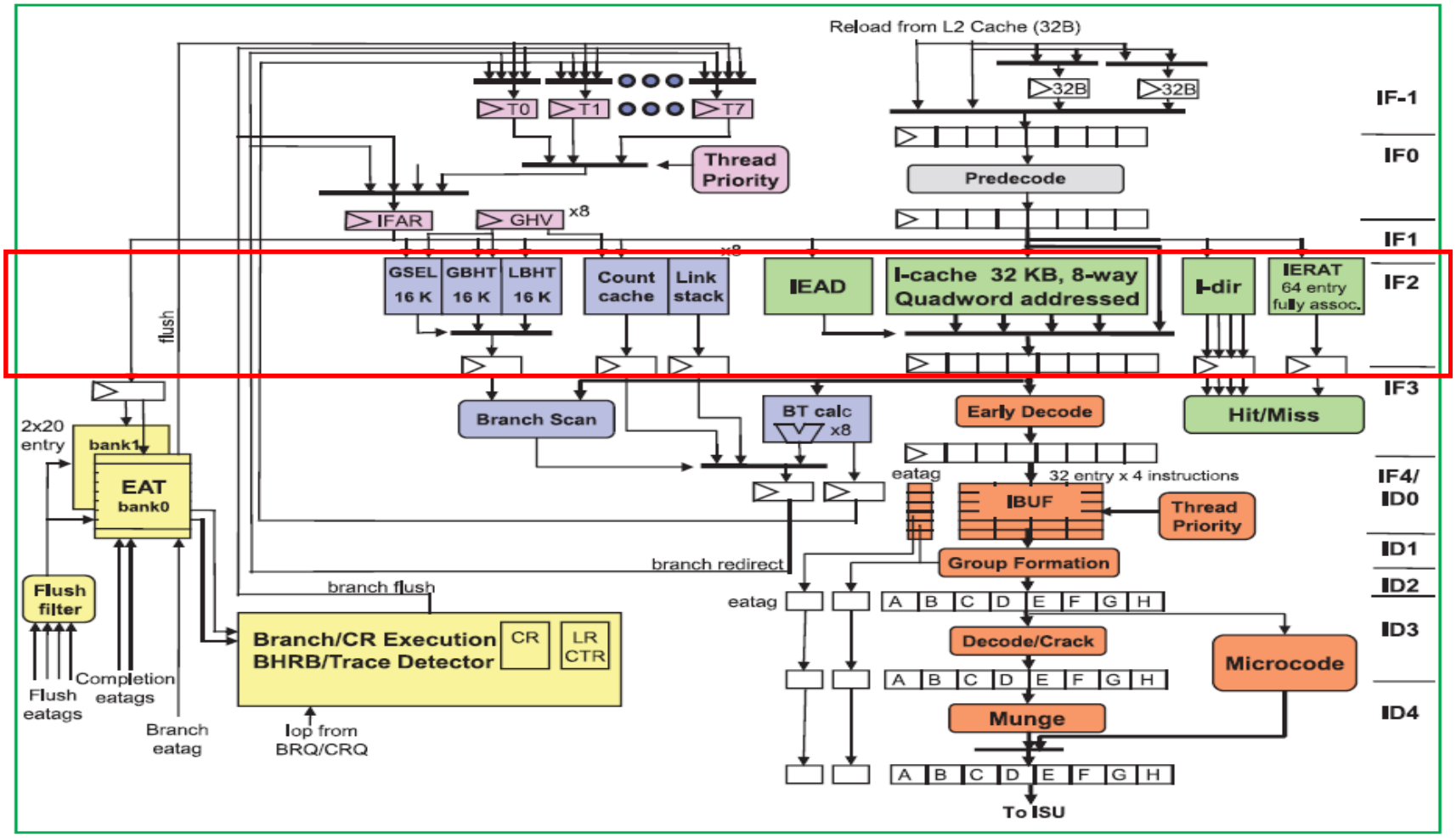

Figure 3

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.)

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.)

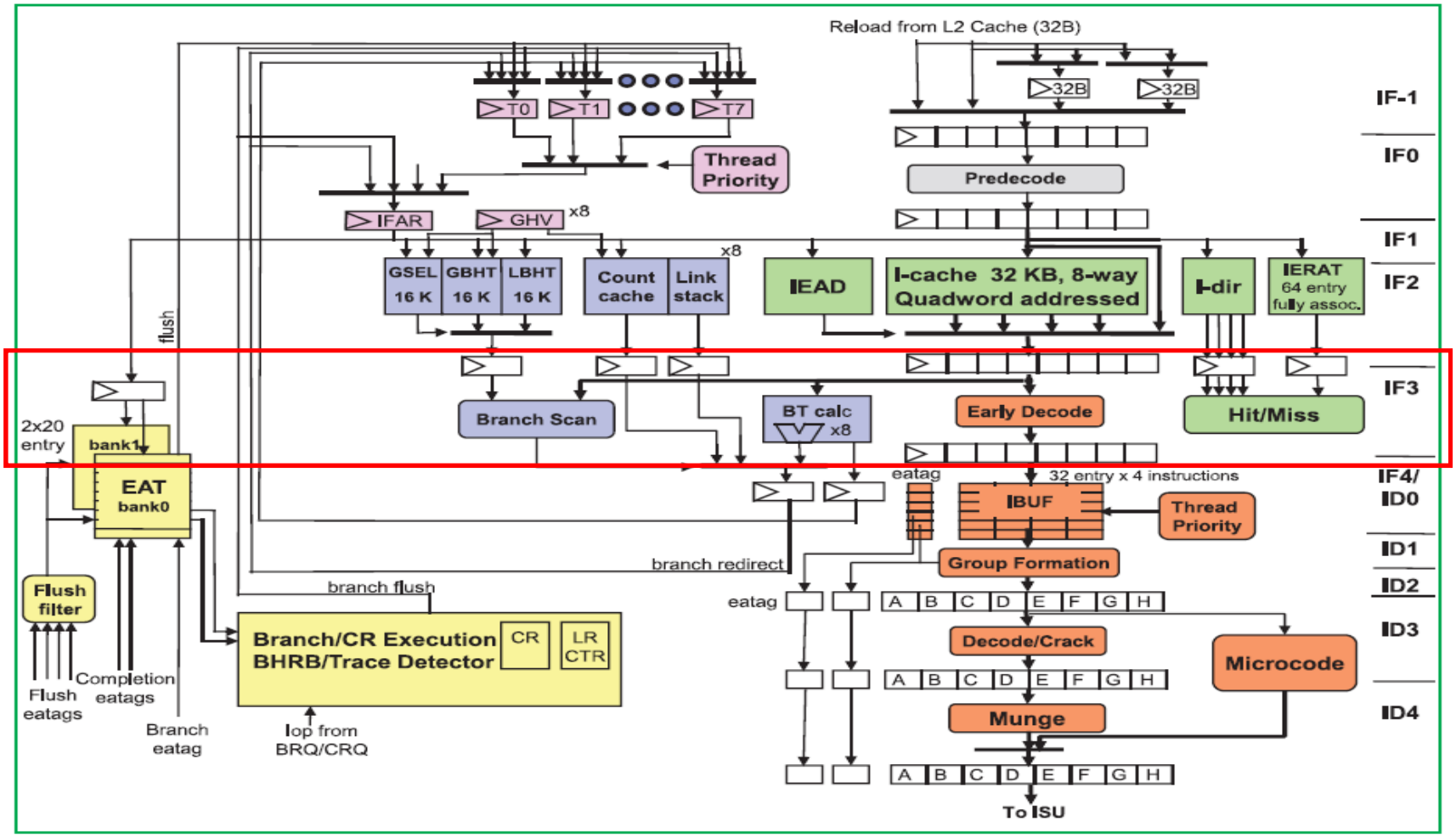

Figure 3

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.)

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.)

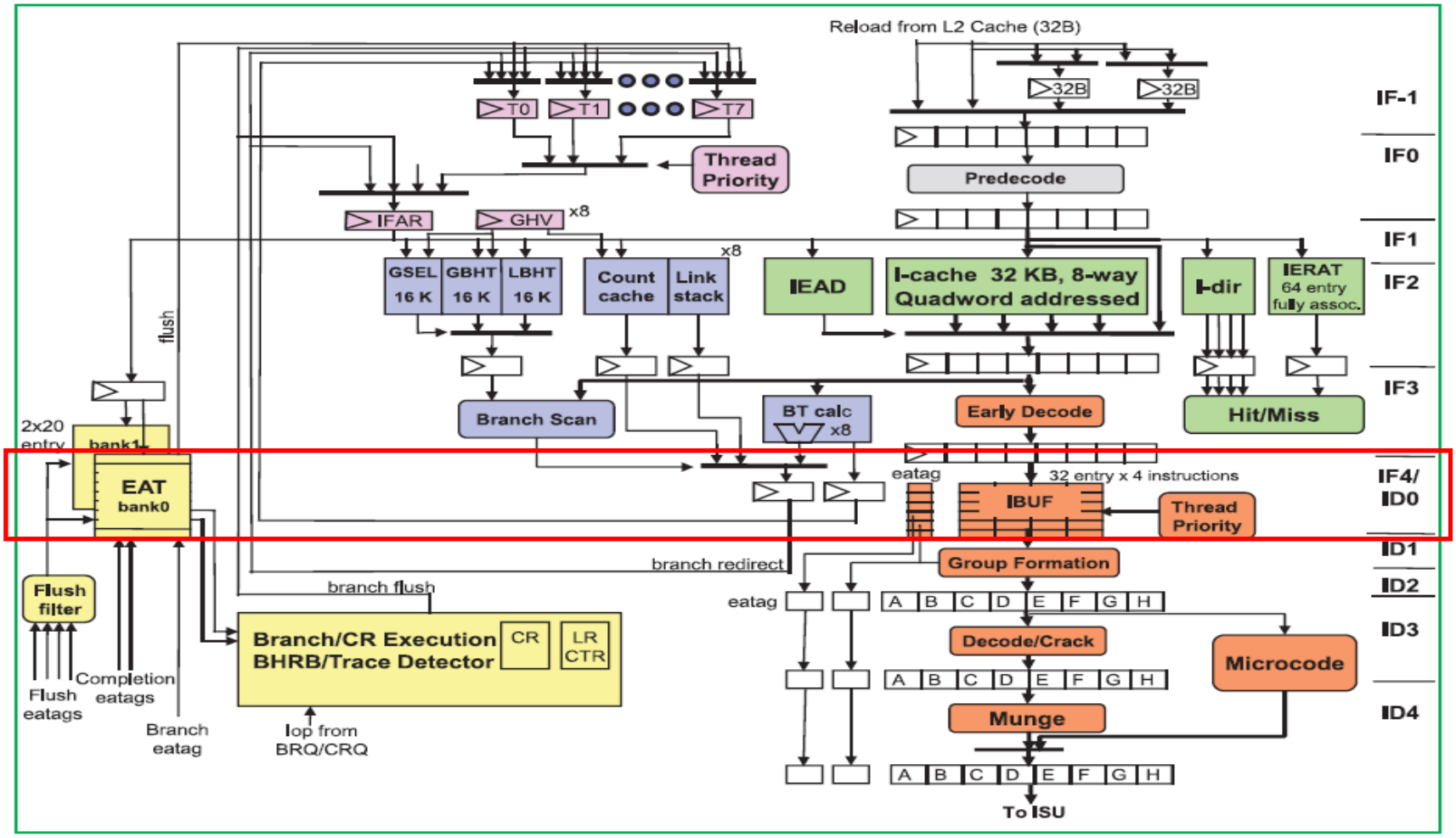

Figure 3

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.)

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.)

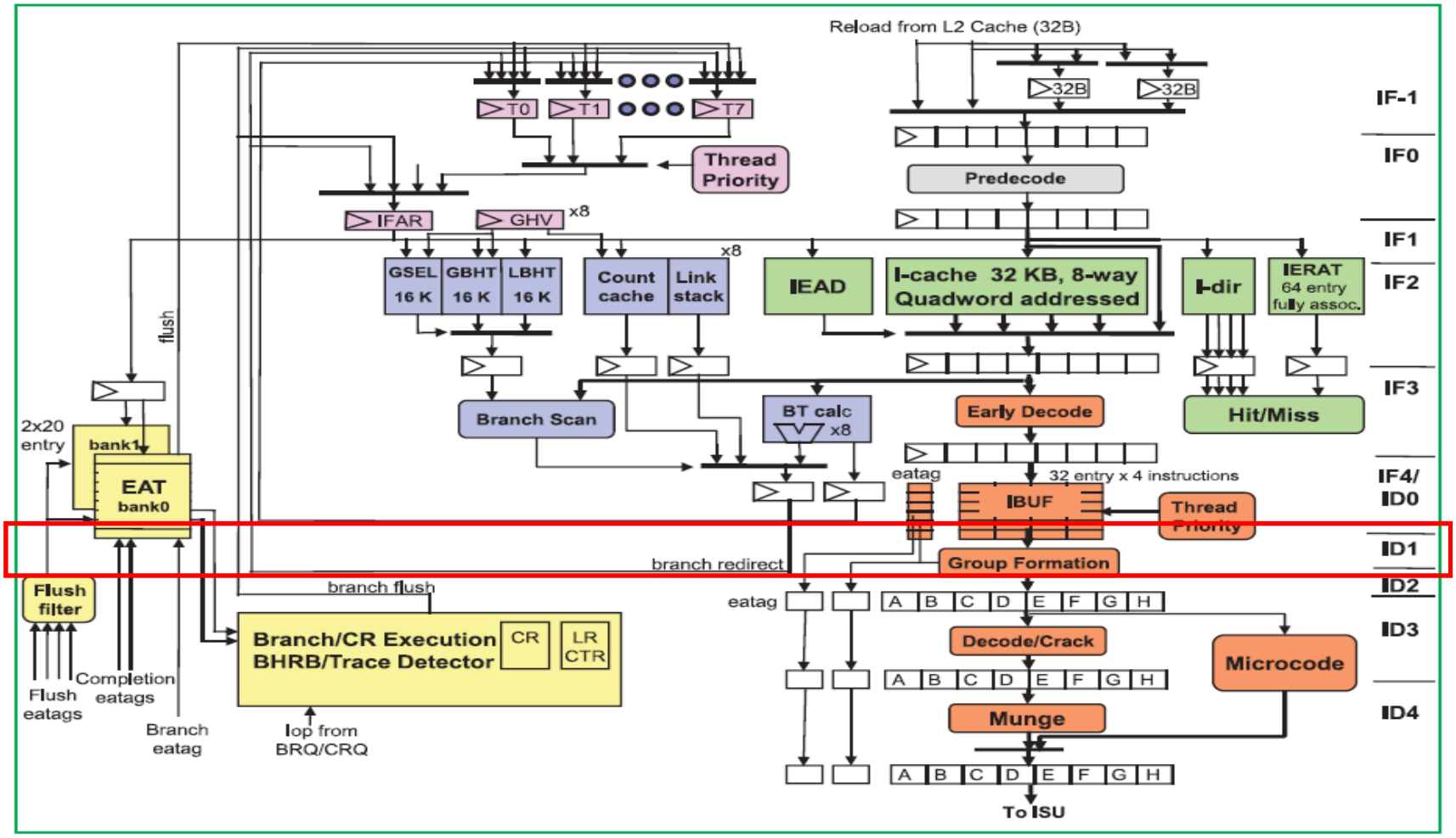

Figure 3

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.)

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.)

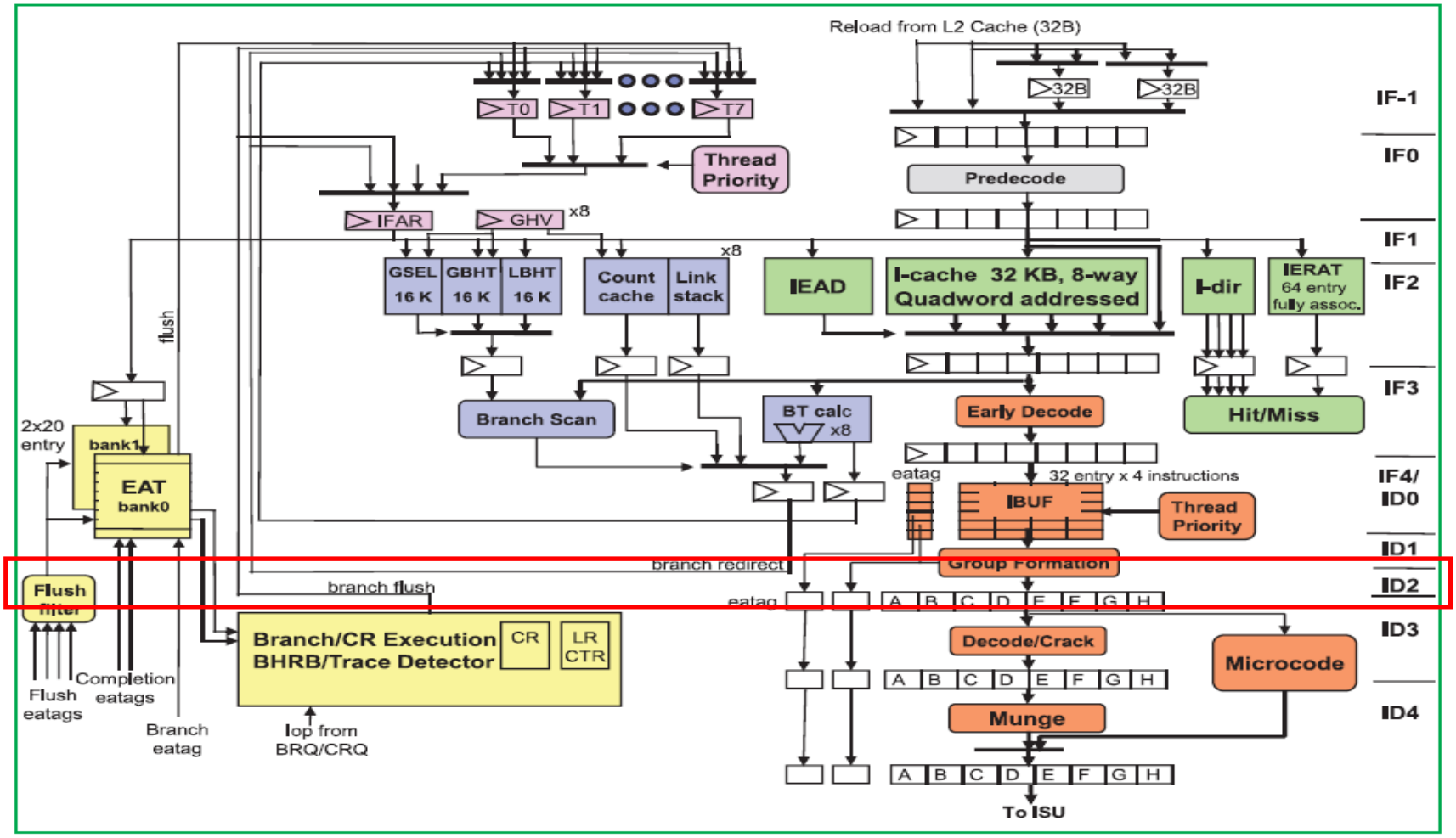

Figure 3

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.)

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.)

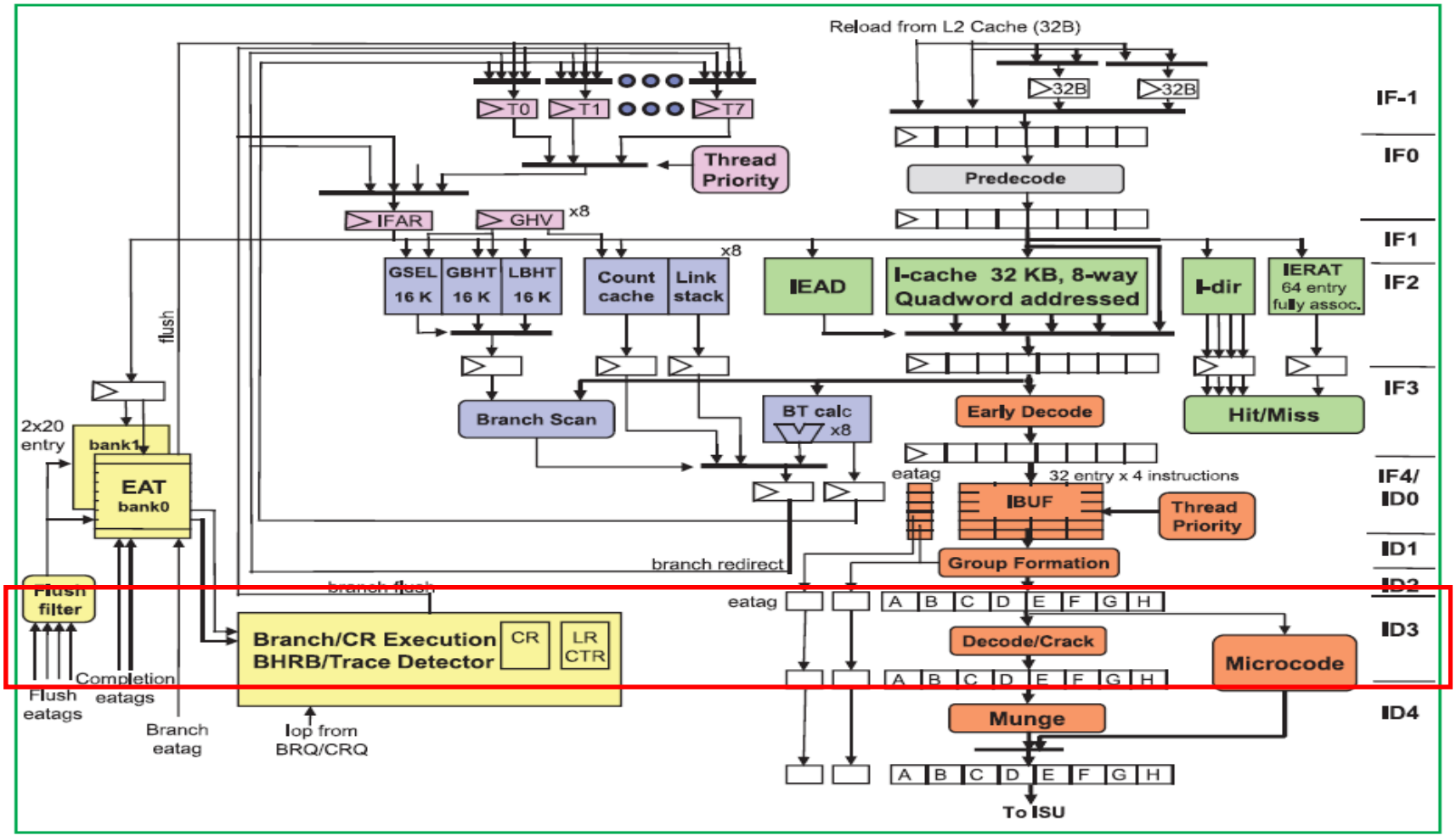

Figure 3

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.)

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.)

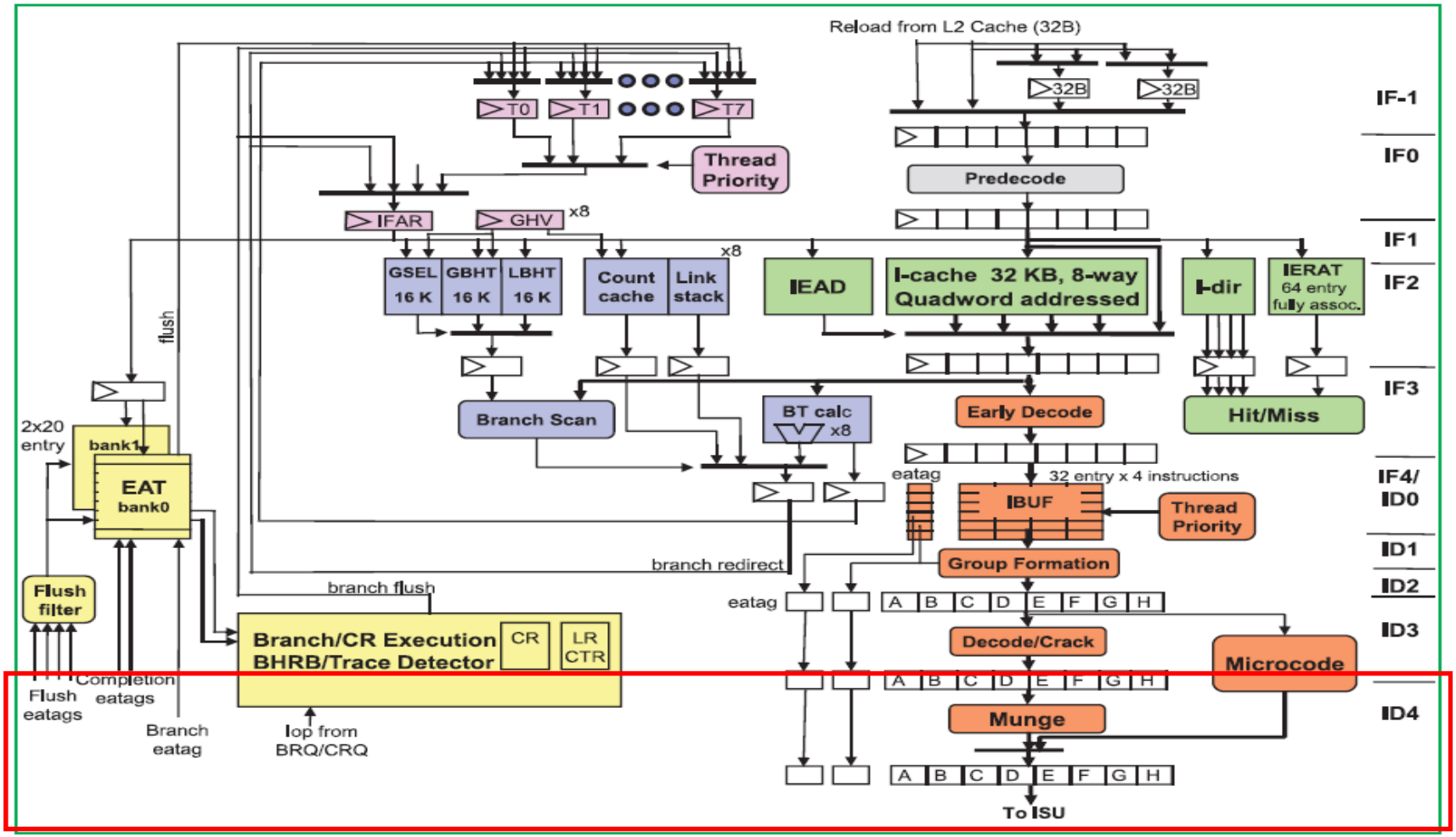

Figure 3

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 Unit *(continued)*

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.

Instruction fetching and pre-decoding

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 pre-fetches. For each demand load request, the pre-fetch engine initiates additional pre-fetches for sequential cache lines following the demand load.

Demand and pre-fetch 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.

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.)

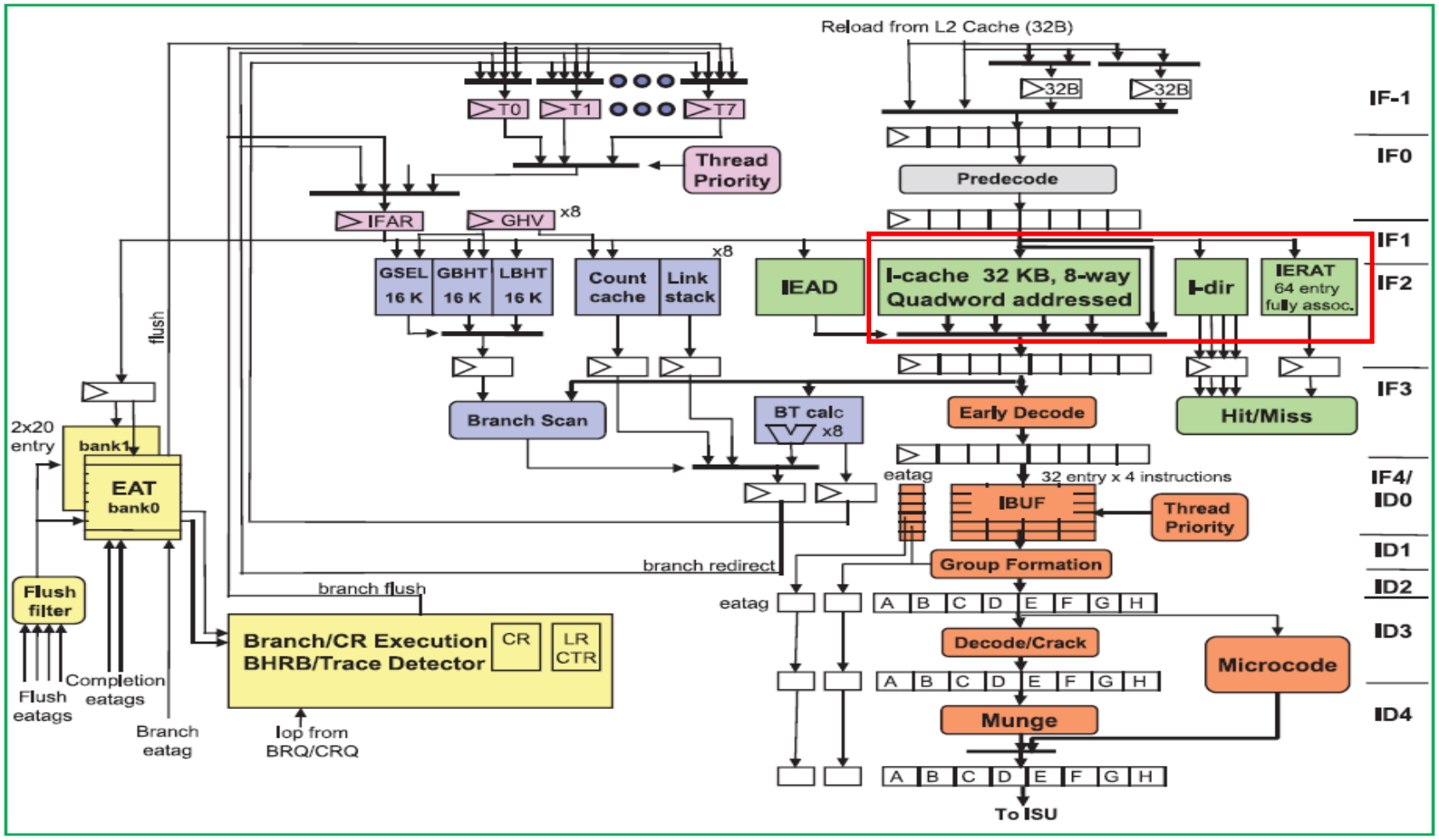

Figure 3

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 fetching and pre-decoding

Instruction prefetching is supported in ST, SMT2, and SMT4 modes only.

Up to three sequential lines are pre-fetched 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.

Pre-fetches are not guaranteed to be fetched and depending on the congestion in the POWER8 processor nest, some pre-fetches may be dropped.

…

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 *(of instructions)*

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

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.)

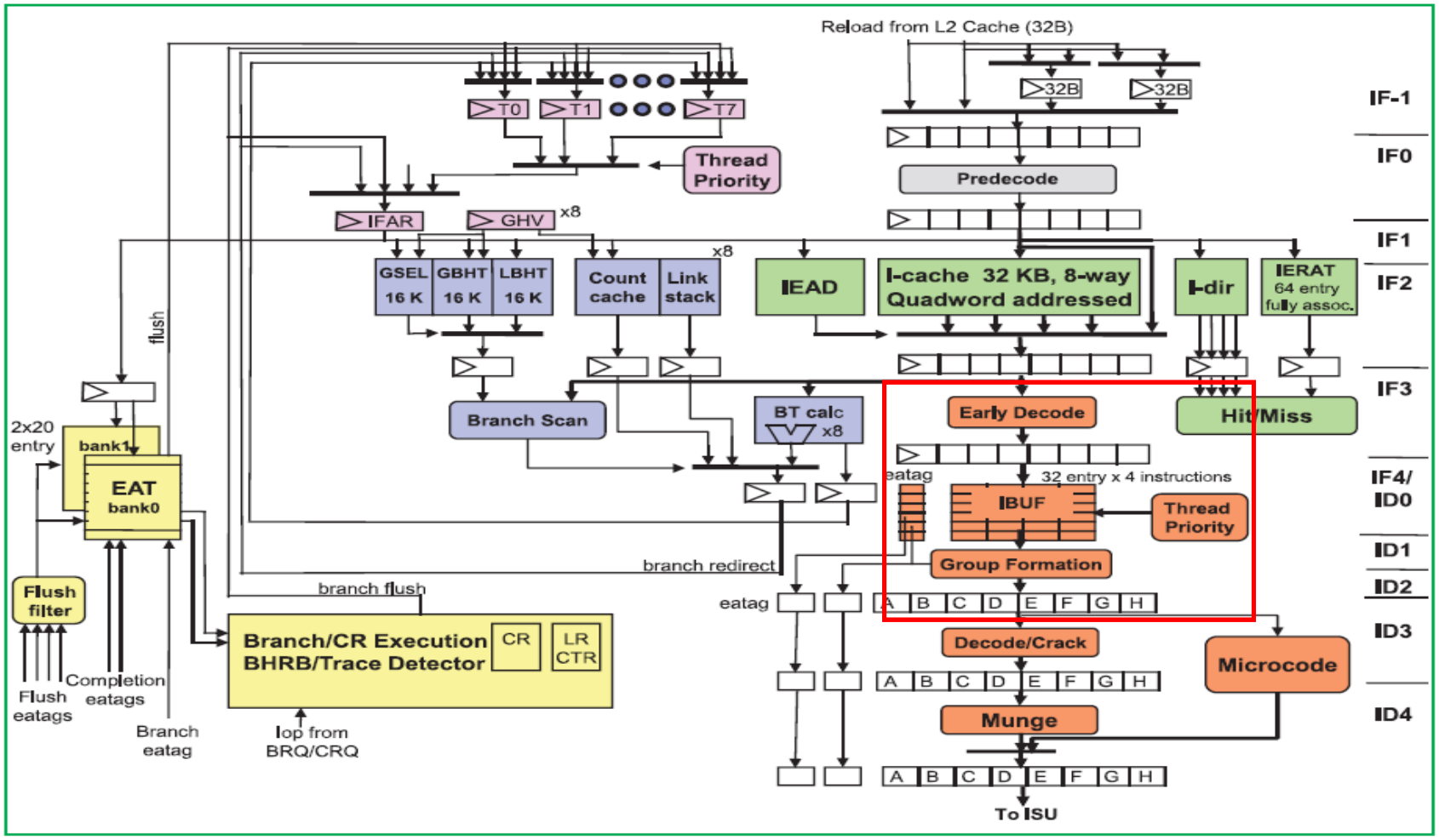

Figure 3

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 decode -- after group formation *(of instructions)*

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.

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.)

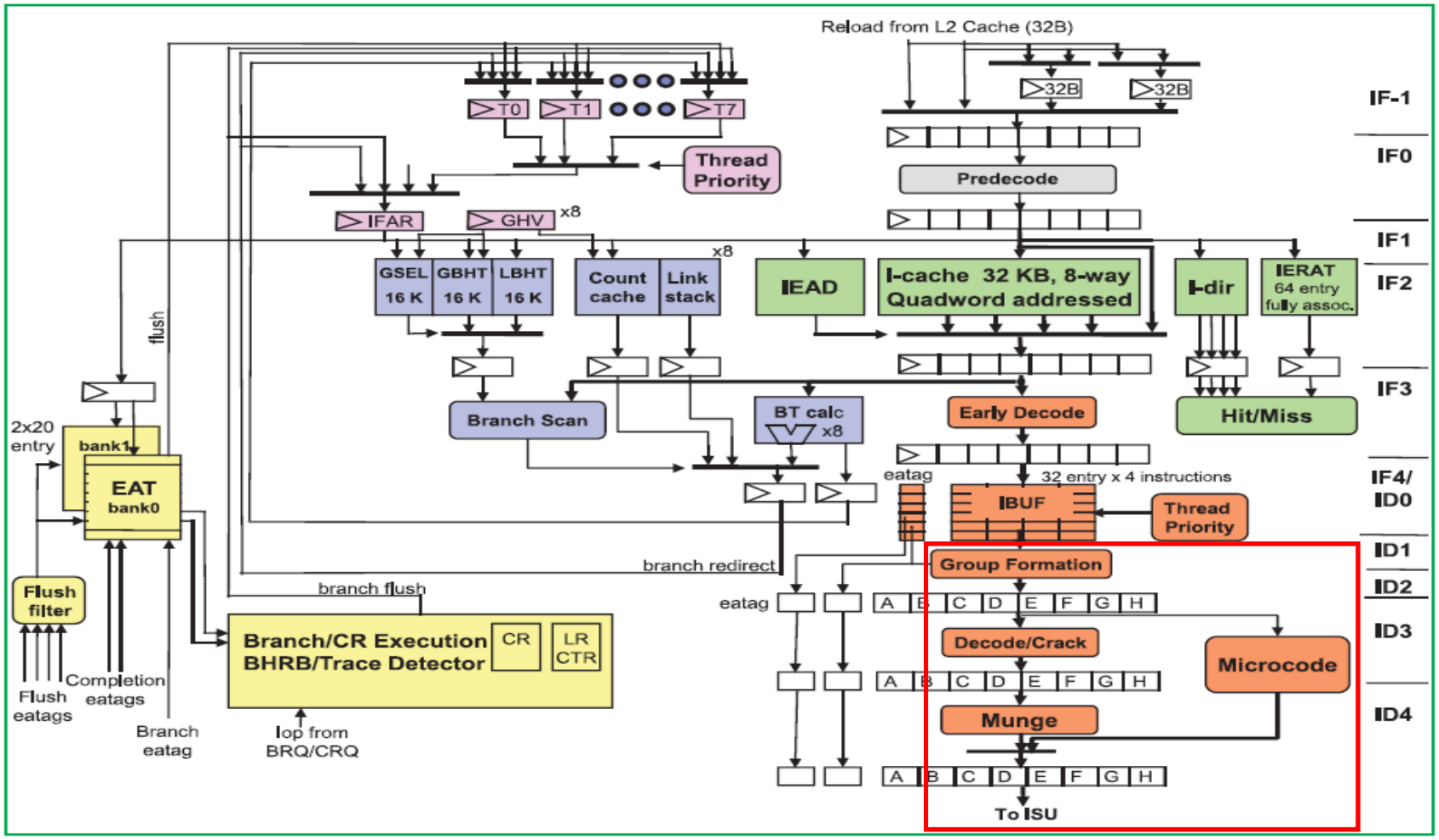

Figure 3

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 Sequencing Unit (ISU)

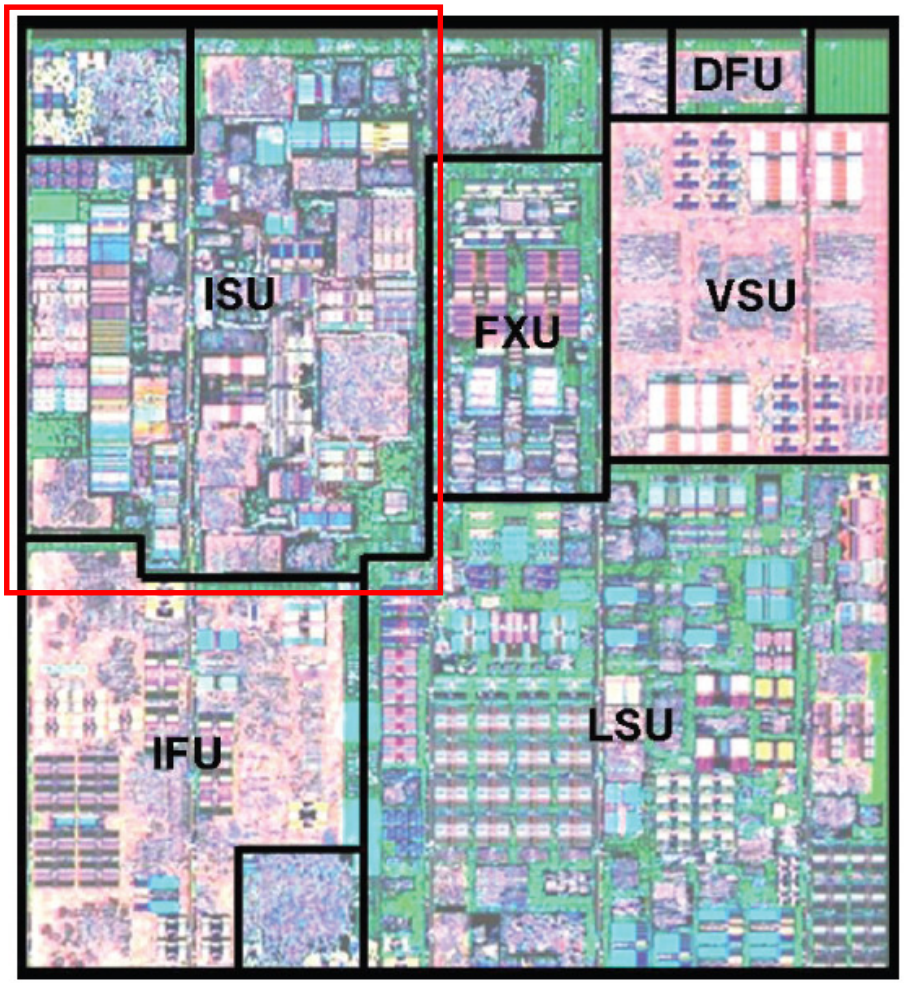

Figure 1

POWER8 processor core floorplan.

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.

Figure 5 illustrates the logical flow of instructions in the ISU.

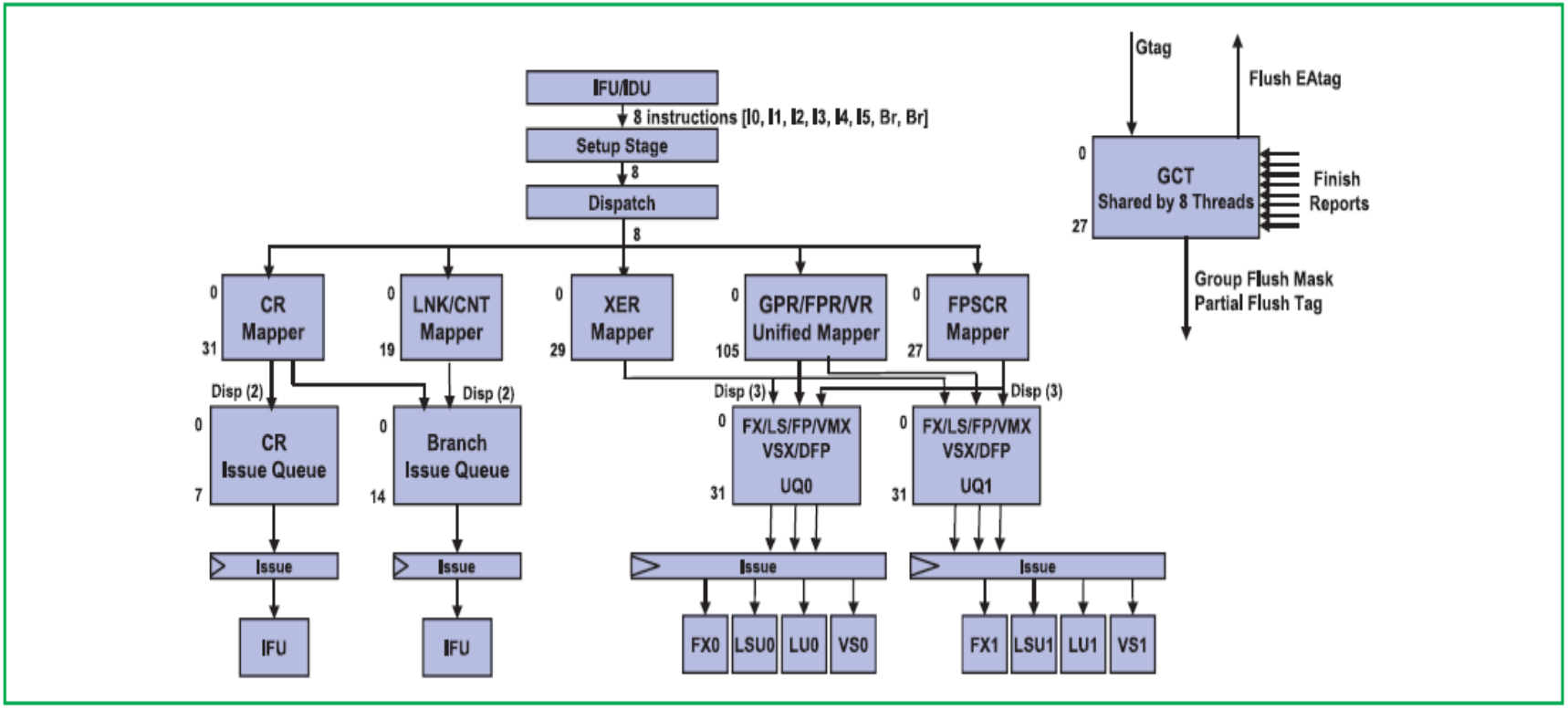

Figure 5

POWER8 instruction sequencing unit (logical flow).

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.

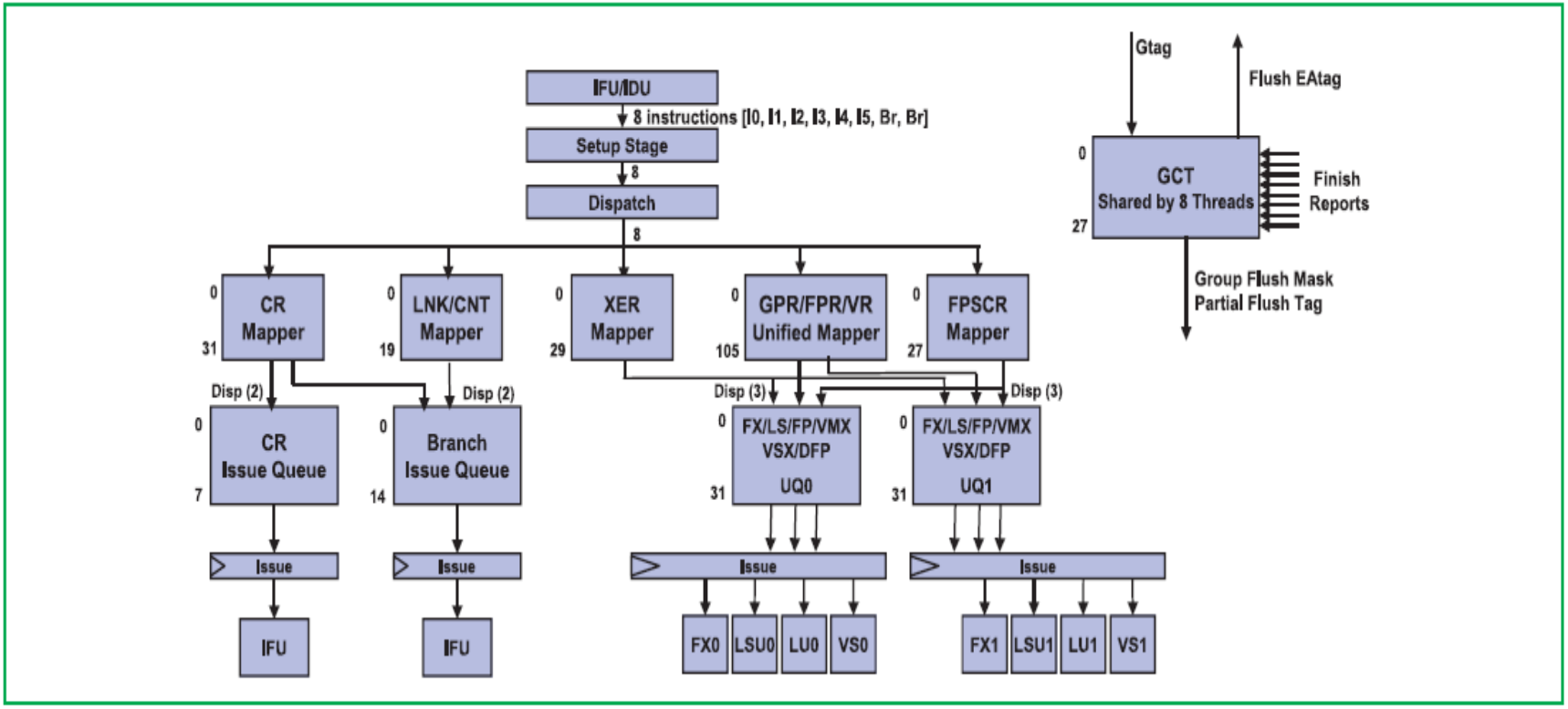

Figure 5

POWER8 instruction sequencing unit (logical flow).

Instruction Sequencing Unit (ISU) *(continued)*

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 resources 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.

Figure 5 illustrates the logical flow of instructions in the ISU.

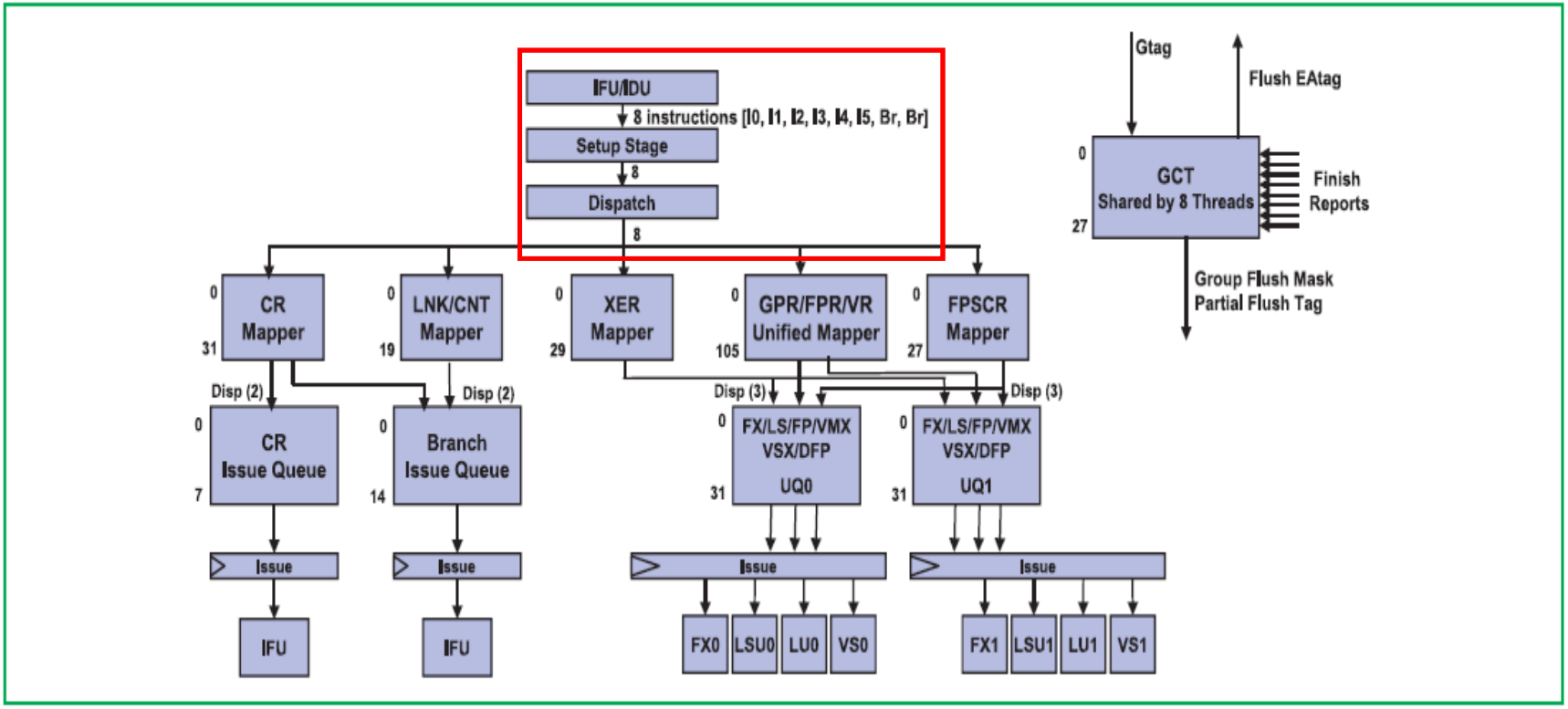

Figure 5

POWER8 instruction sequencing unit (logical flow).

Figure 5 illustrates the logical flow of instructions in the ISU.

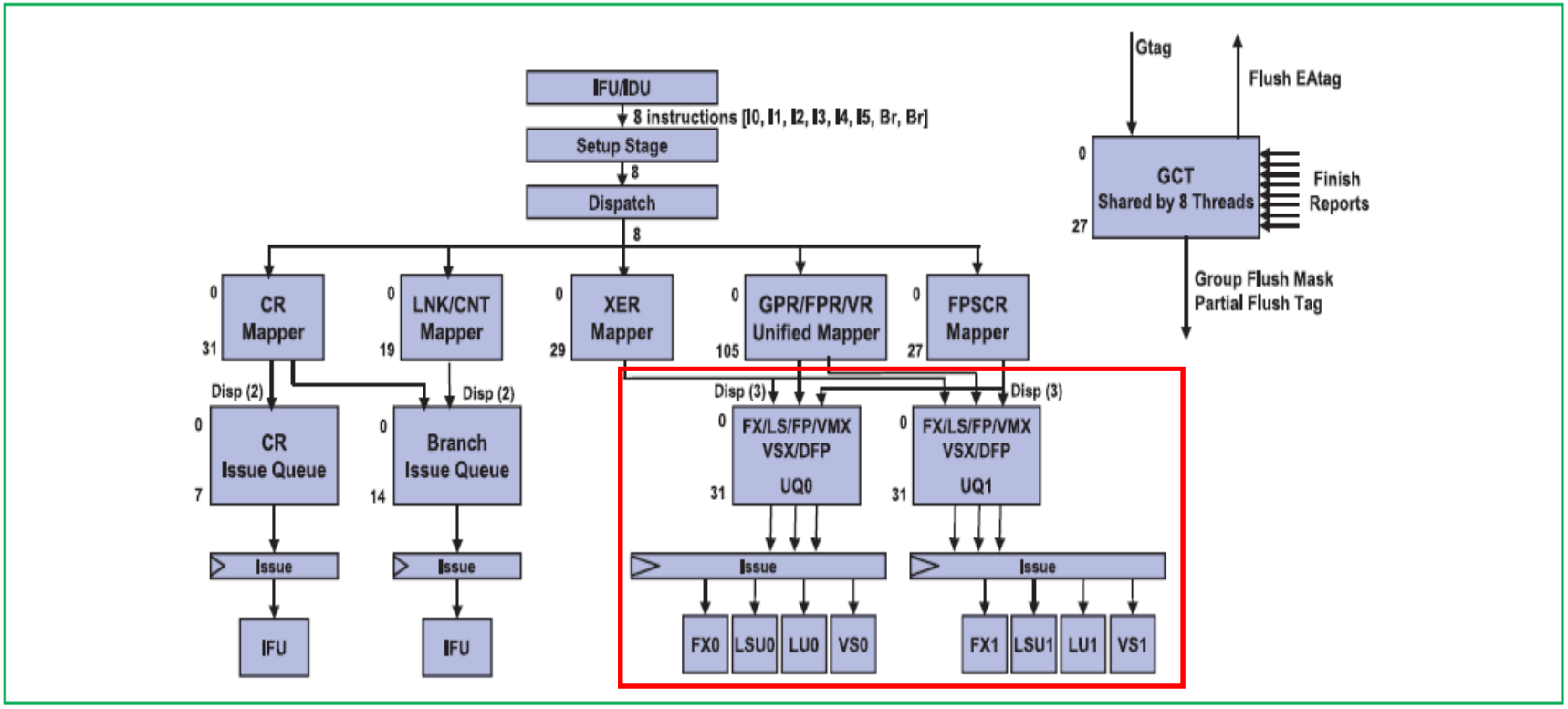

Figure 5

POWER8 instruction sequencing unit (logical flow).

ISU *(and the)* Global Completion Table (GCT)

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.

Figure 5 illustrates the logical flow of instructions in the ISU.

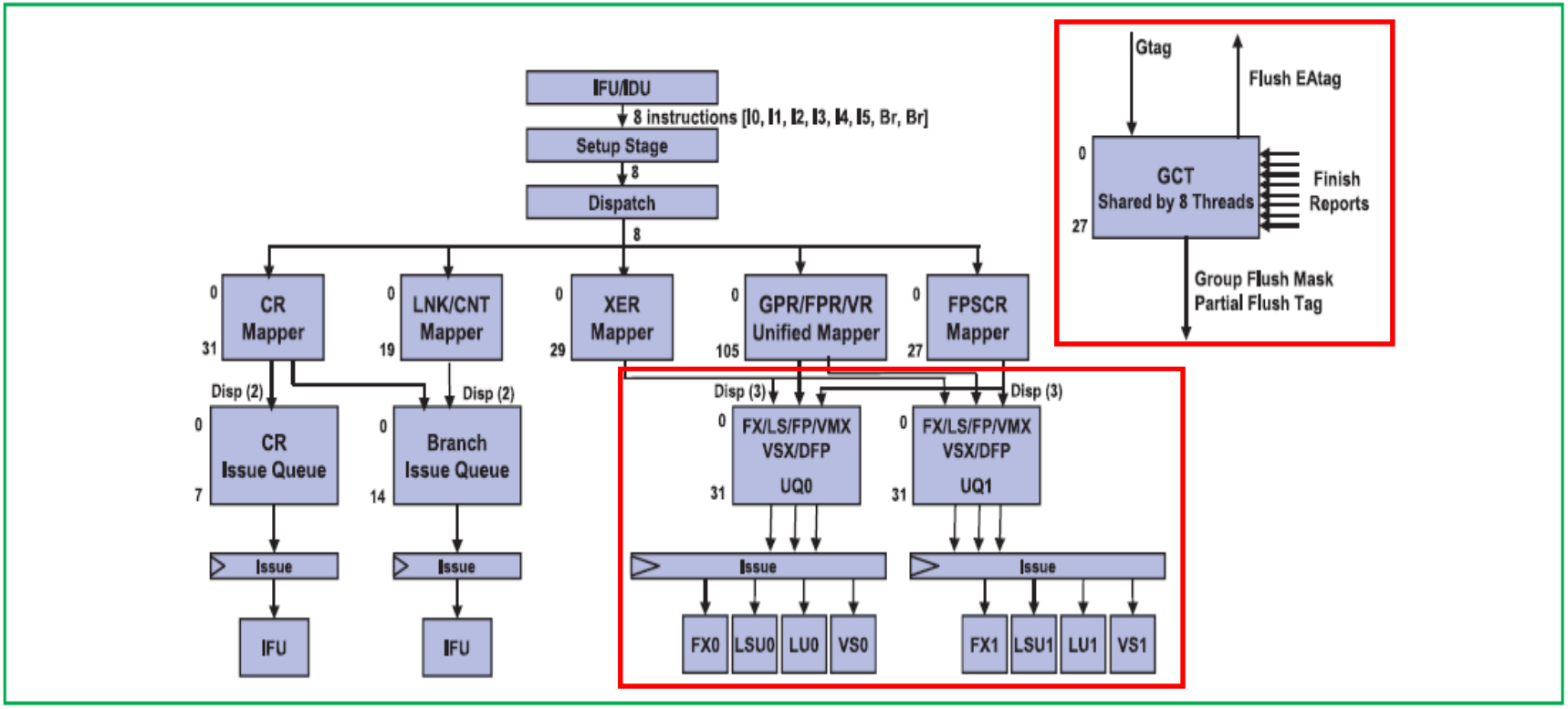

Figure 5

POWER8 instruction sequencing unit (logical flow).

Instructions flow from the memory hierarchy through various issue queues and then are sent to the functional units for execution.

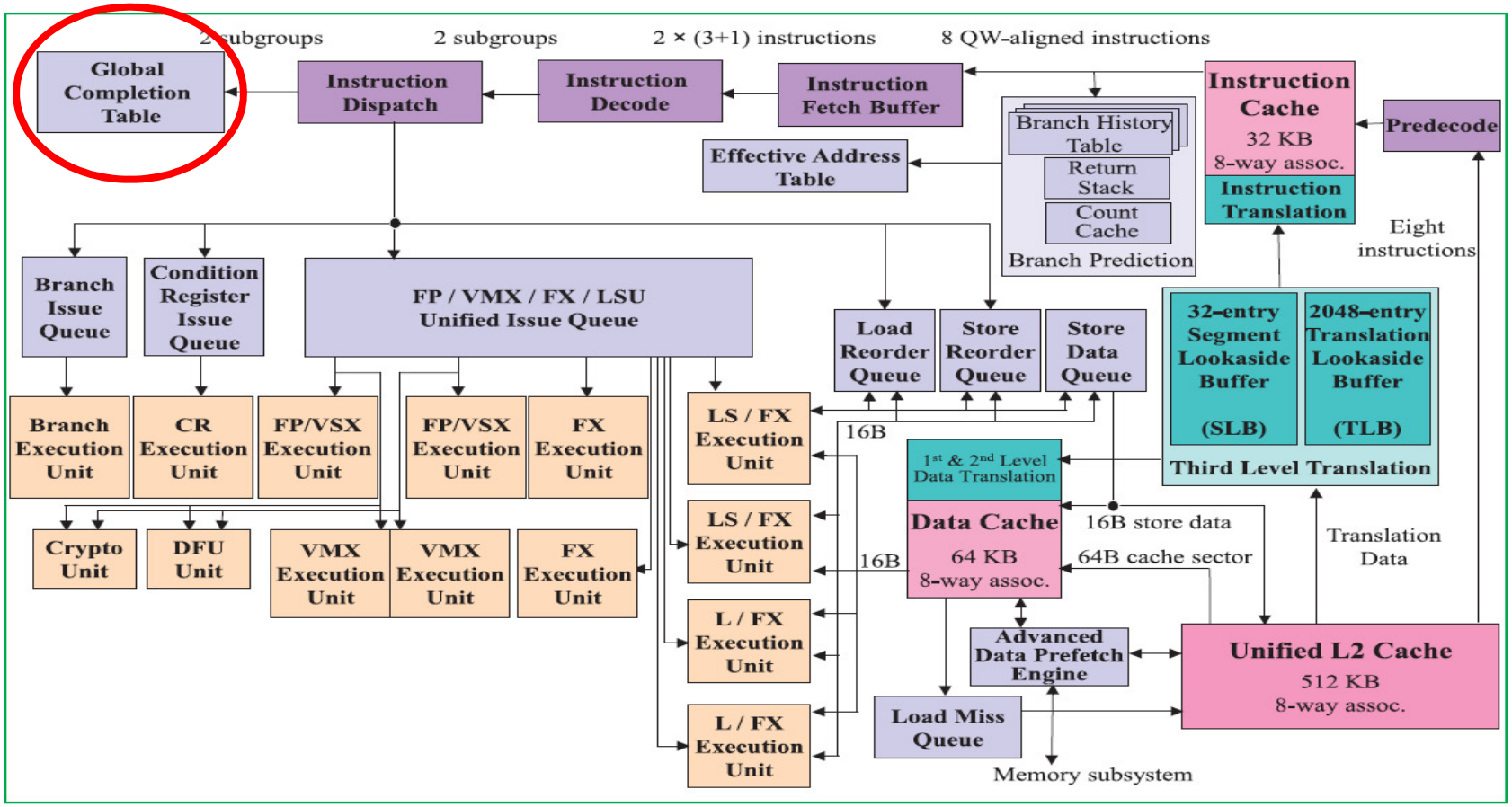

Figure 2

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

ISU *(and the)* Global Completion Table (GCT) *(continued)*

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.

ISU *(and the)* Global Completion Table (GCT) *(continued)*

In ST mode, only one group, consisting of up to eight instructions, can complete per cycle.

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.

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.

The missing tuning factor: ST/SMT-2/-4/-8 threading mode

- We've been watching the **AIX:vmstat –IWw 1:cpu:pc|:ec** values

```
System configuration: lcpu=24 mem=98304MB ent=3.50

     kthr                memory                        page                             faults                   cpu                     time
---------------- ------------------------ ---------------------------------- -------------------- ------------------------- ---------
  r   b    p   w         avm       fre     fi    fo    pi    po     fr     sr    in     sy    cs us sy id wa    pc     ec hr mi se
 66 212    0 376    10599516      8908 208898   190     0     0 209880 474371 23305  32689 49347 40 60  0  0  5.97 170.6 07:45:49
135 179    0 408    10603753     13740 229049    66     0     0 235597 324458 21610  34255 42169 39 61  0  0  5.96 170.3 07:45:50
 88 195    0 476    10620193      7681 208767   200     0     0 217973 424420 23268  35204 47365 41 59  0  0  5.96 170.3 07:45:51
 74 308    0 508    10615392      7983 170112    79     0     0 165232 577709 27767  40485 68236 36 62  0  2  5.96 170.4 07:45:52
111 171    0 623    10614238      7600 283700    39     0     0 280305 381740 16645  23758 27180 30 70  0  0  5.94 169.6 07:45:53
 75 139    0 616    10615800     11800 176554   211     0     0 202322 508624 18288  56699 50940 50 49  0  2  5.95 170.0 07:45:54
 80 140    0 416    10619720     10024 260671   175     0     0 247113 416103 18099  27871 32296 34 66  0  1  5.95 170.0 07:45:55
 79 123    0 494    10619541     11802 218591    73     0     0 220304 436819 21385  35251 41714 36 64  0  0  5.97 170.6 07:45:56
107 134    0 661    10622384      7771 204261   203     0     0 200149 438075 24327  34351 52368 37 63  0  0  5.98 170.7 07:45:57
 70 222    0 496    10625852      8871 196040   129     0     0 200242 439267 24022  41875 51451 38 62  0  0  5.96 170.4 07:45:58
 18 237    0 492    10625100     11543 170450   269     0     0 183842 352926 14443  65697 37923 51 45  0  4  5.93 169.3 07:45:59
138 251    0 652    10629865      8225 280327    65     0     0 271484 511750 16995  26588 27617 27 73  0  0  5.94 169.7 07:46:00
110  68    0 472    10626466      7520 278207    17     0     0 276192 516765 16093  35379 26578 29 71  0  0  5.92 169.2 07:46:02
113 218    0 350    10627603      7530 235577    67     0     0 235086 522790 20146  45574 42703 32 68  0  0  5.99 171.1 07:46:03
 99 187    0 334    10645179      7996 240955   157     0     0 256486 545004 20127  39842 37642 30 70  0  0  5.94 169.6 07:46:04
109  82    0 494    10650842      8024 269364   146     0     0 275012 525465 17350  59493 27922 30 70  0  0  5.94 169.8 07:46:05
103 110    0 248    10655552      9025 274516   100     0     0 281271 526103 16200  74331 26622 28 72  0  0  5.94 169.8 07:46:07
 84 114    0 360    10651811     10365 251368   169     0     0 245856 582747 20175  67976 40111 32 68  0  0  5.95 169.9 07:46:08
102 148    0 279    10658461      7441 285750   164     0     0 291276 462540 16155  84832 25935 29 71  0  0  5.91 168.9 07:46:09
 96 148    0 338    10663679     11152 220862   322     0     0 228257 625080 21903  86171 45734 35 65  0  0  5.98 170.9 07:46:10
     kthr                memory                        page                             faults                   cpu                     time
---------------- ------------------------ ---------------------------------- -------------------- ------------------------- ---------
  r   b    p   w         avm       fre     fi    fo    pi    po     fr     sr    in     sy    cs us sy id wa    pc     ec hr mi se
 83 192    0 474    10662515     10066 283292   179     0     0 292776 398988 10242  47919 19938 47 53  0  1  5.95 170.0 07:46:11
 61 176    0 306    10660046     10750 220960   208     0     0 204797 559097 17661  54067 36157 46 53  0  0  5.97 170.5 07:46:12
 87 130    0 484    10662862      7645 243972   167     0     0 241235 442606 19228  29701 36625 34 66  0  0  5.96 170.2 07:46:13
 73 214    0 270    10669973      7617 208389   134     0     0 218368 451905 20355  37358 43749 43 57  0  0  5.96 170.2 07:46:14
 64 169    0 350    10669921      7555 225201   117     0     0 226223 354926 21032  41807 41315 37 63  0  0  5.97 170.6 07:46:15
 24 218    0 391    10676954      9836 209241    51     0     0 217487 475267 21227  51247 44534 33 64  0  3  5.95 170.0 07:46:16
 78 170    0 276    10689099      7661 226187   145     0     0 237468 415431 19934  32417 42240 41 59  0  0  5.89 168.1 07:46:17
139 144    0 536    10690574      8024 185162   226     0     0 183546 607865 25510  35254 68076 35 65  0  0  6.04 172.5 07:46:18
101 161    0 403    10696958      7616 264120    22     0     0 274347 314681 18946  28118 34260 30 70  0  0  5.93 169.5 07:46:19
 95 238    0 402    10700104     10392 198493   214     0     0 203411 559791 23737  31972 71900 35 64  0  1  5.95 169.9 07:46:20
 86 179    0 370    10698534      9452 251838    47     0     0 247821 248156 19971  28538 38416 31 69  0  0  6.00 171.6 07:46:21
107 173    0 324    10693430      7610 199484   225     0     0 194030 437450 24024  36601 61687 36 64  0  0  5.95 170.0 07:46:22
 50 216    0 714    10691688      7525 189307   255     0     0 183217 526700 23571  38127 55190 40 59  0  1  5.98 170.8 07:46:23
 93 220    0 418    10698380      7740 213037   101     0     0 221767 379842 23297  32474 50401 36 64  0  0  5.97 170.5 07:46:24
 37 217    0 598    10703911      8824 198431    44     0     0 203602 422083 23904  41602 55736 33 67  0  0  5.97 170.5 07:46:25
 70 205    0 546    10705976      7520 215674    99     0     0 214059 337136 24801  36854 55341 29 71  0  1  5.97 170.5 07:46:26
 60 217    0 428    10706406      8045 182738   203     0     0 187695 473692 25275  34238 63691 38 61  0  0  5.97 170.5 07:46:27
 27 242    0 390    10710258      7521 208430   283     0     0 207623 420118 22340  45530 43630 42 58  0  0  5.97 170.5 07:46:28
  0   0    0 432    10707620      7558 207454   345     0     0 207502 387871 23063  38031 49638 37 63  0  0  5.97 170.5 07:46:29
 69 212    0 527    10709781      7992 187406   235     0     0 193002 445210 23583  44806 52071 40 60  0  0  5.96 170.4 07:46:30
     kthr                memory                        page                             faults                   cpu                     time
---------------- ------------------------ ---------------------------------- -------------------- ------------------------- ---------
  r   b    p   w         avm       fre     fi    fo    pi    po     fr     sr    in     sy    cs us sy id wa    pc     ec hr mi se
```

The missing tuning factor: ST/SMT-2/-4/-8 threading mode

- The `:pc` | `:ec` values tell us how much CPU is used and the ec%, or CPUbusy%
- Next, it is clear the POWER8 core has markedly improved threading capability
- Perhaps now we should begin working with the missing tuning factor too, and not just use the `:pc` | `:ec` values to monitor CPU utilization

page				faults			cpu				time				
pi	po	fr	sr	in	sy	cs	us	sy	id	wa	pc	ec	hr	mi	se
0	0	209880	474371	23305	32689	49347	40	60	0	0	5.97	170.6		07:45:49	
0	0	235597	324458	21610	34255	42169	39	61	0	0	5.96	170.3		07:45:50	
0	0	217973	424420	23268	35204	47365	41	59	0	0	5.96	170.3		07:45:51	
0	0	165232	577709	27767	40485	68236	36	62	0	2	5.96	170.4		07:45:52	
0	0	280305	381740	16645	23758	27180	30	70	0	0	5.94	169.6		07:45:53	
0	0	202322	508624	18288	56699	50940	50	49	0	2	5.95	170.0		07:45:54	
0	0	247113	416103	18099	27871	32296	34	66	0	1	5.95	170.0		07:45:55	
0	0	220304	436819	21385	35251	41714	36	64	0	0	5.97	170.6		07:45:56	
0	0	200149	438075	24327	34351	52368	37	63	0	0	5.98	170.7		07:45:57	
0	0	200242	439267	24022	41875	51451	38	62	0	0	5.96	170.4		07:45:58	
0	0	183842	352926	14443	65697	37923	51	45	0	4	5.93	169.3		07:45:59	
0	0	271484	511750	16995	26588	27617	27	73	0	0	5.94	169.7		07:46:00	
0	0	276192	516765	16093	35379	26578	29	71	0	0	5.92	169.2		07:46:02	
0	0	235086	522790	20146	45574	42703	32	68	0	0	5.99	171.1		07:46:03	
0	0	256486	545004	20127	39842	37642	30	70	0	0	5.94	169.6		07:46:04	
0	0	275012	525465	17350	59493	27922	30	70	0	0	5.94	169.8		07:46:05	
0	0	281271	526103	16200	74331	26622	28	72	0	0	5.94	169.8		07:46:07	
0	0	245856	582747	20175	67976	40111	32	68	0	0	5.95	169.9		07:46:08	
0	0	291276	462540	16155	84832	25935	29	71	0	0	5.91	168.9		07:46:09	
0	0	228257	625080	21903	86171	45734	35	65	0	0	5.98	170.9		07:46:10	
page				faults			cpu				time				
pi	po	fr	sr	in	sy	cs	us	sy	id	wa	pc	ec	hr	mi	se

The missing tuning factor: ST/SMT-2/-4/-8 threading mode

- We should begin tuning POWER8 with more attention to its innate capability
- Tuning POWER8 by :pc and :ec% values alone is missing a deeper dimension
- We should begin controlling an ignored factor; I will call it “SMT threadedness”

- Too often I find workloads barely able to a keep a thread active on a core
- A POWER core can show great productivity -- but only if we push it harder
- Do we agree that compelling more work from our investment is a good thing?

- Do you want to see what I mean? Sure, no problem.
- What can you distinguish between the top and bottom on the next slide?

AIX:vmstat -IWw 1: Which is more CPUcore efficient?

```
System configuration: lcpu=24 mem=98304MB ent=3.50

   kthr                 memory                              page                          faults                cpu              time
--------------- ------------------------ ------------------------------------- ------------------ --------------------------- --------
  r   b   p   w        avm        fre     fi    fo    pi    po     fr     sr    in     sy    cs us sy id wa   pc    ec hr mi se
 66 212   0 376   10599516       8908 208898   190     0     0 209880 474371 23305  32689 49347 40 60  0  0 5.97 170.6 07:45:49
135 179   0 408   10603753      13740 229049    66     0     0 235597 324458 21610  34255 42169 39 61  0  0 5.96 170.3 07:45:50
 88 195   0 476   10620193       7681 208767   200     0     0 217973 424420 23268  35204 47365 41 59  0  0 5.96 170.3 07:45:51
 74 308   0 508   10615392       7983 170112    79     0     0 165232 577709 27767  40485 68236 36 62  0  2 5.96 170.4 07:45:52
111 171   0 623   10614238       7600 283700    39     0     0 280305 381740 16645  23758 27180 30 70  0  0 5.94 169.6 07:45:53
 75 139   0 616   10615800      11800 176554   211     0     0 202322 508624 18288  56699 50940 50 49  0  2 5.95 170.0 07:45:54
 80 140   0 416   10619720      10024 260671   175     0     0 247113 416103 18099  27871 32296 34 66  0  1 5.95 170.0 07:45:55
 79 123   0 494   10619541      11802 218591    73     0     0 220304 436819 21385  35251 41714 36 64  0  0 5.97 170.6 07:45:56
107 134   0 661   10622384       7771 204261   203     0     0 200149 438075 24327  34351 52368 37 63  0  0 5.98 170.7 07:45:57
 70 222   0 496   10625852       8871 196040   129     0     0 200242 439267 24022  41875 51451 38 62  0  0 5.96 170.4 07:45:58
 18 237   0 492   10625100      11543 170450   269     0     0 183842 352926 14443  65697 37923 51 45  0  4 5.93 169.3 07:45:59
138 251   0 652   10629865       8225 280327    65     0     0 271484 511750 16995  26588 27617 27 73  0  0 5.94 169.7 07:46:00
110  68   0 472   10626466       7520 278207    17     0     0 276192 516765 16093  35379 26578 29 71  0  0 5.92 169.2 07:46:02
113 218   0 350   10627603       7530 235577    67     0     0 235086 522790 20146  45574 42703 32 68  0  0 5.99 171.1 07:46:03
 99 187   0 334   10645179       7996 240955   157     0     0 256486 545004 20127  39842 37642 30 70  0  0 5.94 169.6 07:46:04
109  82   0 494   10650842       8024 269364   146     0     0 275012 525465 17350  59493 27922 30 70  0  0 5.94 169.8 07:46:05
103 110   0 248   10655552       9025 274516   100     0     0 281271 526103 16200  74331 26622 28 72  0  0 5.94 169.8 07:46:07
 84 114   0 360   10651811      10365 251368   169     0     0 245856 582747 20175  67976 40111 32 68  0  0 5.95 169.9 07:46:08
102 148   0 279   10658461       7441 285750   164     0     0 291276 462540 16155  84832 25935 29 71  0  0 5.91 168.9 07:46:09
 96 148   0 338   10663679      11152 220862   322     0     0 228257 625080 21903  86171 45734 35 65  0  0 5.98 170.9 07:46:10
   kthr                 memory                              page                          faults                cpu              time
--------------- ------------------------ ------------------------------------- ------------------ --------------------------- --------
  r   b   p   w        avm        fre     fi    fo    pi    po     fr     sr    in     sy    cs us sy id wa   pc    ec hr mi se
```

```
System configuration: lcpu=64 mem=98304MB ent=2.00

   kthr                 memory                              page                          faults                cpu              time
--------------- ------------------------ ------------------------------------- ------------------ --------------------------- --------
  r   b   p   w        avm        fre     fi    fo    pi    po     fr     sr    in     sy    cs us sy id wa    pc    ec hr mi se
  0   0   0   0   13157874     104956    938    51     0     0      0      0  1151  31013  6879 71  5 24  0 10.89 544.6 22:30:04
 16   1   0   4   13159509     101976   1308    36     0     0      0      0  1099  19090  5028 64  3 32  0 10.96 548.0 22:30:05
 12   2   0   4   13157043     102588   1857    60     0     0      0      0  1041  13968  4398 61  3 36  0 11.62 581.2 22:30:06
 12   2   0   4   13160790      97452   1384     8     0     0      0      0   450  79248  3397 61  2 37  0 11.65 582.7 22:30:07
 12   1   0   4   13156803      99825   1613    14     0     0      0      0   486  13369  3638 61  2 37  0 10.84 541.9 22:30:08
 10   2   0   2   13156883      97990   1756     7     0     0      0      0   658  80019  4007 61  2 37  0 10.27 513.3 22:30:09
 11   2   0   4   13156834      95549   2501    46     0     0      0      0   825   6014  4327 62  2 36  0 10.50 525.0 22:30:10
  8   2   0   2   13156834      93249   2287    50     0     0      0      0   794  14215  4372 61  2 37  0 10.07 503.4 22:30:11
 14   2   0   4   13156834      89791   3447    65     0     0      0      0  1414   7407  5589 61  2 36  0 10.91 545.7 22:30:12
 12   2   0   4   13156834      88134   1656     8     0     0      0      0   568   5845  3900 62  2 37  0 10.10 505.2 22:30:13
  9   2   0   2   13156834      86150   1986    21     0     0      0      0   560  13775  3804 61  2 37  0 10.08 503.9 22:30:14
 10   2   0   2   13156834      84387   1763    17     0     0      0      0   637   6477  3855 62  2 36  0 10.07 503.6 22:30:15
 10   2   0   4   13156834      82772   1600    19     0     0      0      0   535   6517  3757 62  1 37  0  9.97 498.7 22:30:16
  8   2   0   4   13156834      80782   1985    20     0     0      0      0   596  13078  6712 61  2 37  0 10.26 513.1 22:30:17
  9   2   0   4   13156834      78924   1848    28     0     0      0      0   680  12331  4065 61  2 37  0  9.83 491.5 22:30:18
  9   2   0   4   13156834      76985   1939     7     0     0      0      0   593   6042  3812 61  2 37  0  9.45 472.5 22:30:19
 12   2   0   2   13156834      74687   2327    40     0     0      0      0   690   6528  4030 61  2 37  0  9.73 486.7 22:30:20
  0   0   0   4   13156834      72848   1805    13     0     0      0      0   501  13816  3718 62  2 37  0  9.85 492.6 22:30:21
  8   2   0   4   13156834      71102   1744     8     0     0      0      0   477   5866  3461 62  1 37  0  9.38 469.0 22:30:22
 10   2   0   4   13156834      68841   2261    12     0     0      0      0   672   5842  4107 62  2 37  0  9.77 488.7 22:30:23
   kthr                 memory                              page                          faults                cpu              time
--------------- ------------------------ ------------------------------------- ------------------ --------------------------- --------
  r   b   p   w        avm        fre     fi    fo    pi    po     fr     sr    in     sy    cs us sy id wa    pc    ec hr mi se
```

Single Threading mode (ST)

- A hard ST mode (AIX:smtctl –t 1) is not the same as a soft ST mode
- A hard ST mode (AIX:smtctl –t 1) cannot progress to SMT-2/-4/-8 on its own

- A soft ST mode may only be achieved with a hard SMT-2/-4/-8 mode setting (whether by default AIX:smtctl –t 4 or a hard AIX:smtctl –t 2|8)
- A soft ST mode may progress to SMT-2/-4/-8 when needed

- A soft ST mode is unfortunately universal and virtually the default standard
- A soft ST mode is rooted in configuring too many virtual CPUs for SPLPARs
- A soft ST mode is also rooted in configuring too many dedicated CPU cores

- But, when a soft ST mode is needed, it is the fastest AND the most wasteful

Single Threading mode (ST)

- ST mode offers the most responsiveness/attention/dedication when ec<=100
- This paper shows that optimally "feeding" the CPUcore is the main goal
- Devoting a CPUcore to a single thread means "feeding the CPUcore with all possible fury"
- ST mode ensures the most instructions and data possible are prefetched, fetched, loaded/stored, decoded, grouped, dispatched, executed, completed per cycle – but only for one thread
- In ST mode, the dispatched instructions are executed/balanced between both sets of 8 + 8 execution pipelines of the core
- **ST mode is most appropriate for workloads with fewer threads that are compute-intensive, not IO dependent, and have sustained activity durations**
- **ST mode is also most appropriate for workloads with immediate response-time demands at the expense of wasted/idle CPUcycles**
- Most enterprise workloads do not need the dedication of ST mode on POWER8
- Optional for study: Set AIX:schedo:vpm_throughput_mode=1 (default=0)

Simultaneous Multi Threading mode (SMT-2)

- A hard SMT-2 mode (AIX:smtctl –t 2) is not the same as a soft SMT-2 mode
- A hard SMT-2 mode (AIX:smtctl –t 2) cannot progress to SMT-4/-8 on its own

- A soft SMT-2 mode may only be achieved with a hard SMT-4/SMT-8 mode setting (whether by default AIX:smtctl –t 4 or a hard AIX:smtctl –t 8)
- A soft SMT-2 mode may progress to SMT-4/-8 when needed

- A soft SMT-2 mode should be the standard threading model for POWER8/AIX workloads **not needing the dedicated attention of ST mode**
- A soft SMT-2 mode should be the standard threading model for POWER8/AIX workloads **not needing a dedicated CPU LPAR implementation**

- A soft SMT-2 mode has a better balance of CPUcore utilization & performance
- A soft SMT-2 mode workload is easily monitored, i.e. `AIX:mpstat -w 2`

Simultaneous Multi Threading mode (SMT-2)

- But how do we ensure/implement an optimal soft SMT-2 mode?

- First, establish a higher SMT-4/SMT-8 mode “thread count” overflow capability
- Accept the default hard SMT-4 mode, or set a hard SMT-8 mode (smtctl –t 8)
- Next monitor AIX:mpstat –w 2 and learn to identify the real-time threadedness
- If in soft ST mode, remove a virtual CPU and monitor; repeat as needed
- If in any SMT-4 mode, add a virtual CPU and monitor; repeat as needed
- Alternatively, study and implement the more sophisticated tactic, i.e. schedo
- Dynamically set AIX:schedo:vpm_throughput_mode=2 (default=0)

- **For workloads not needing a soft ST mode for unfettered performance, a soft SMT-2 mode is confidently acceptable for POWER8/AIX production service**

Simultaneous Multi Threading mode (SMT-4)

- What about purposely tuning to use a soft SMT-4 mode? Is it ever useful?
- Yes, and more so for LPARs configured with two or more virtual CPUs

- A soft SMT-4 mode is subjectively applicable for any nonproduction workload
- Next, some (if not most) batch workloads are more throughput-focused overall, and do not require the per-thread responsiveness of soft ST/SMT-2 mode
- Also, some workloads have a high concurrent count of very short duration threads that rapidly-repeatedly do virtually nothing as they quickly jump on&off CPUcores; confirm w/sustained 20:1 ratio of **AIX:mpstat –w 2:cs to :ics**
- Finally, to exploit full utilization of limited software licenses, a soft SMT-4 mode will ensure every available atom of productivity is extracted per licensed core

- For any of the use-cases above, execute AIX:smtctl –t 8, then study and dynamically set AIX:schedo:vpm_throughput_mode=4 (default=0)

Simultaneous Multi Threading mode (SMT-8)

- What about tuning to use a hard SMT-8 mode? Is it ever useful?
- Yes, it is specifically useful for setting a soft SMT-4 mode in the slide above
- There is no hard SMT-16 mode, so a soft SMT-8 mode cannot be set

- There are likely amazing applications perfect for hard SMT-8 mode – but given my POWER8/AIX enterprise focus, I haven't run across them yet
- Most enterprise workloads do not have enough concurrently running threads to achieve a natural SMT-8 thread density; when attempted, they are typically holding at a steady SMT-4 thread density

- Of course, a hard SMT-8 mode can be forced by explicit directive
- This directive is setting AIX:schedo:vpm_throughput_mode=8 (default=0)

Load/Store Unit (LSU)

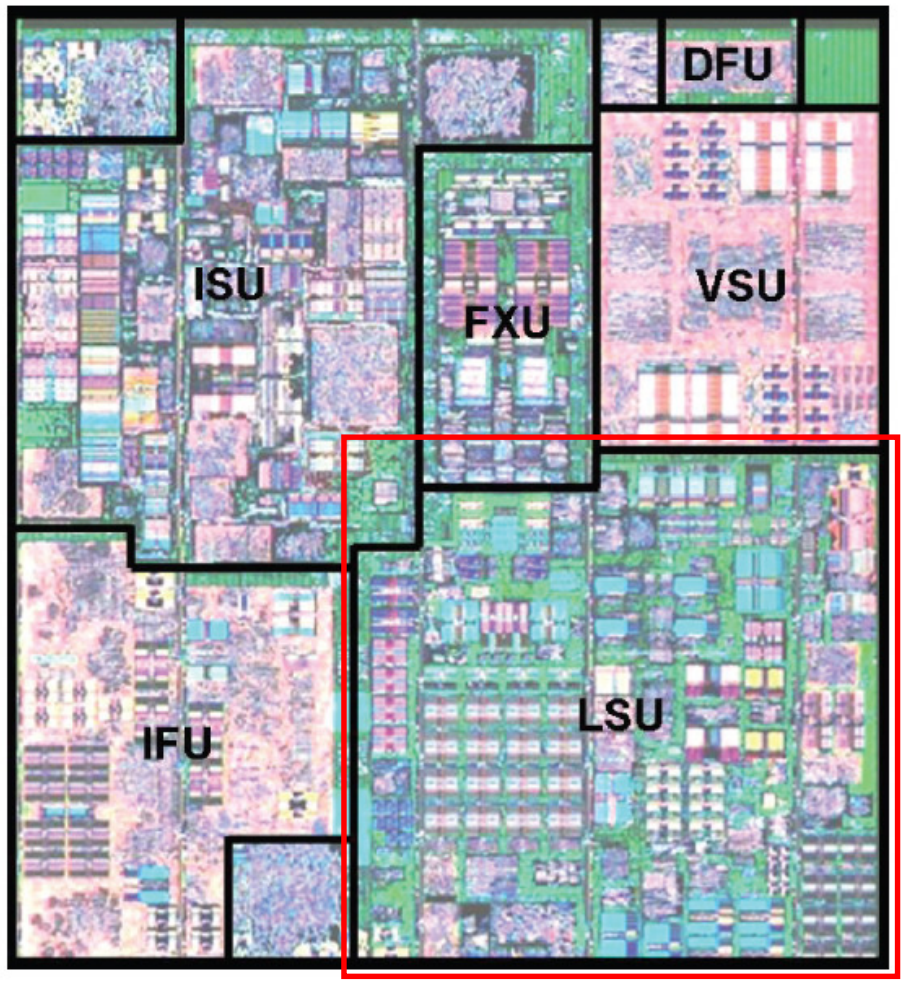

Figure 1

POWER8 processor core floorplan.

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.

Figure 6 illustrates the microarchitecture of the POWER8 LS0 pipeline.

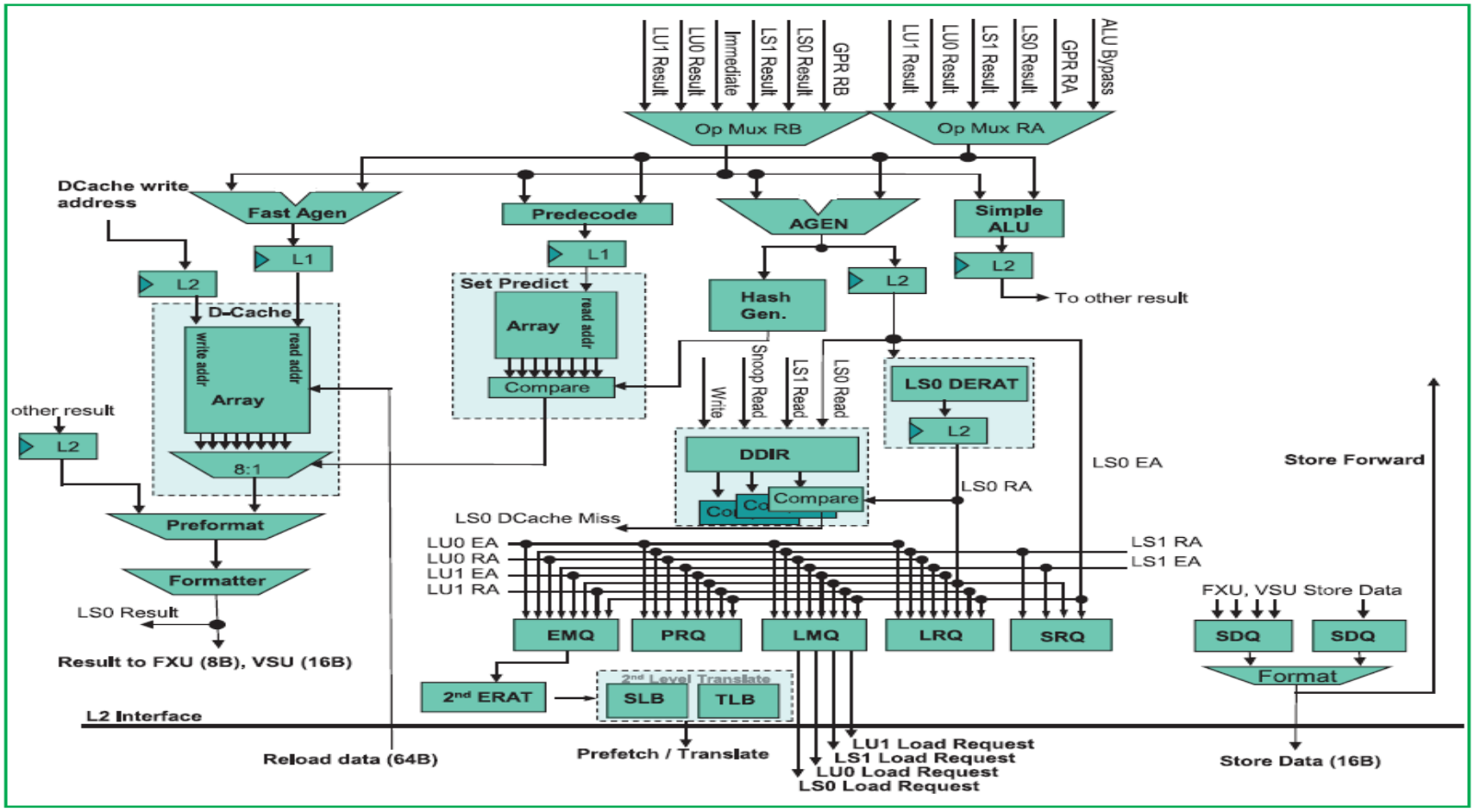

Figure 6

POWER8 LSU micro architecture (LS0 pipe shown).

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

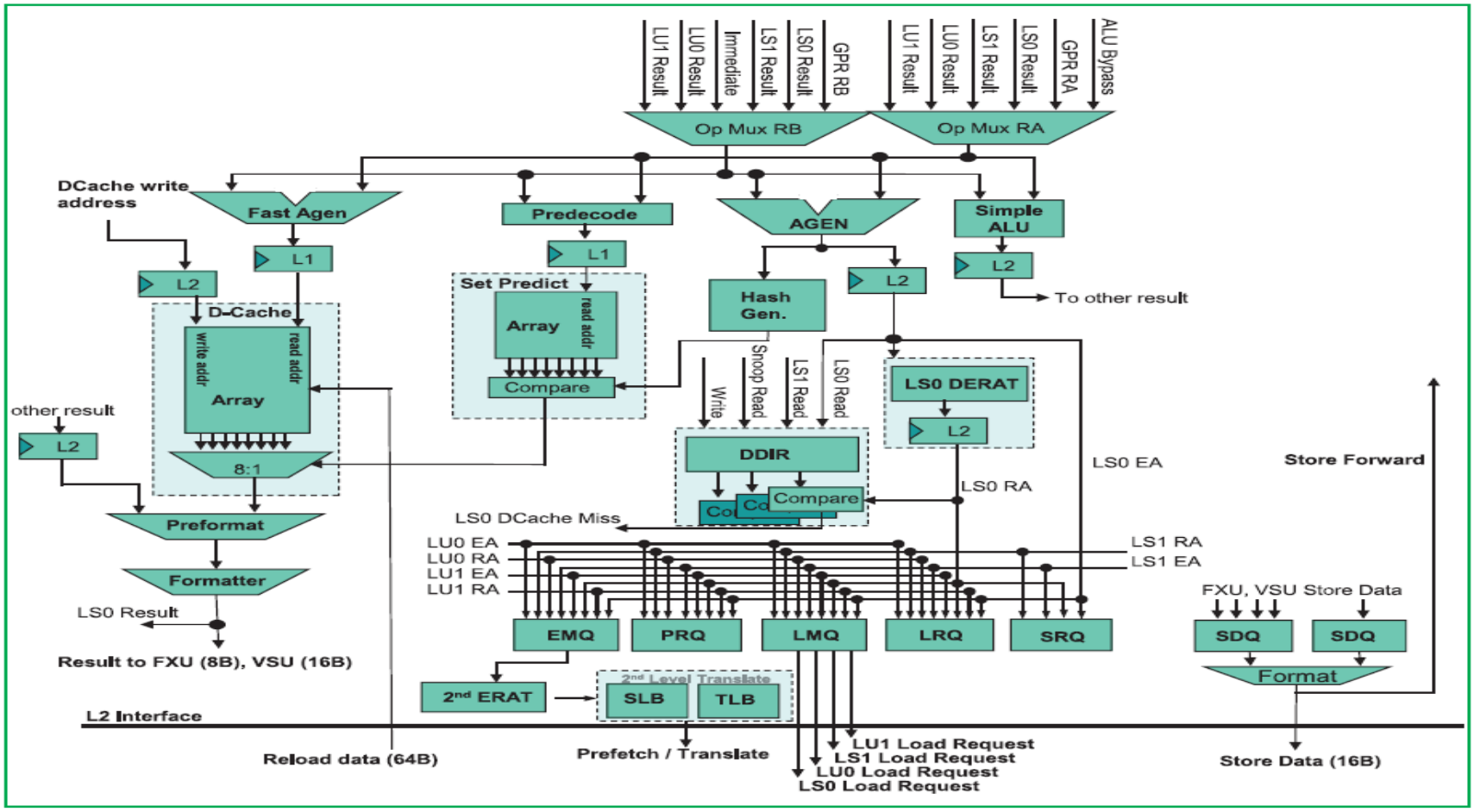

Figure 6

POWER8 LSU micro architecture (LS0 pipe shown).

Load/Store Unit (LSU) *(continued)*

The POWER8 LSU contains two symmetric load pipelines (L0 and L1) and two symmetric load/store pipelines (LS0 and LS1).

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.

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.

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).

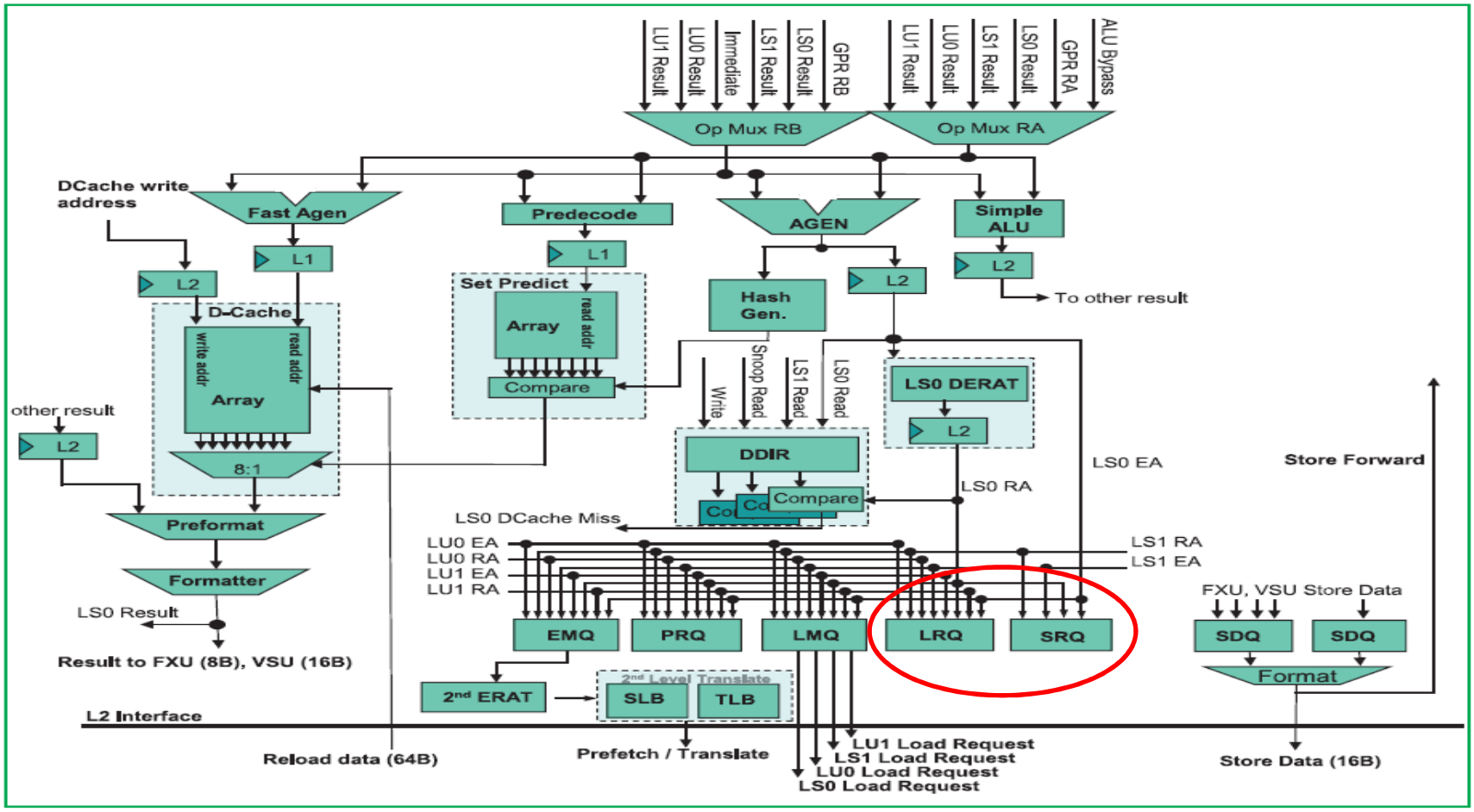

Figure 6

POWER8 LSU micro architecture (LS0 pipe shown).

ISU 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)**.

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.

ISU Address Translation *(continued)*

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 Translation Lookaside Buffer (TLB) that is shared by all active threads.

ISU Address Translation *(continued)*

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.

ISU Address Translation *(continued)*

The primary Data Effective-to-Real Address Translation (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.

ISU Address Translation *(continued)*

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.

ISU Address Translation *(continued)*

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.

ISU Address Translation *(continued)*

The Translation Lookaside Buffer (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.

- **Tight&Fat:** Configure fewer vCPUs, grant 0.7-0.9 eCPU per vCPU, and drive the core-level harder with SMT-2/4/8 thread-level workloads on POWER8.
- Tight&Fat aims to preclude use of “UnCapped” shared-CPU capacity
- Tight&Fat aims to avoid running beyond CPU Entitlement, i.e. ec>100
- Tight&Fat aims to keep vCPUs on their Home cores for the hottest TLB hits
- Tight&Fat means vCPUs do not visit strange CPUcores with no TLB content

ISU Address Translation *(continued)*

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.

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.

Data prefetch *(continued)*

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.

Fixed-Point Unit (FXU)

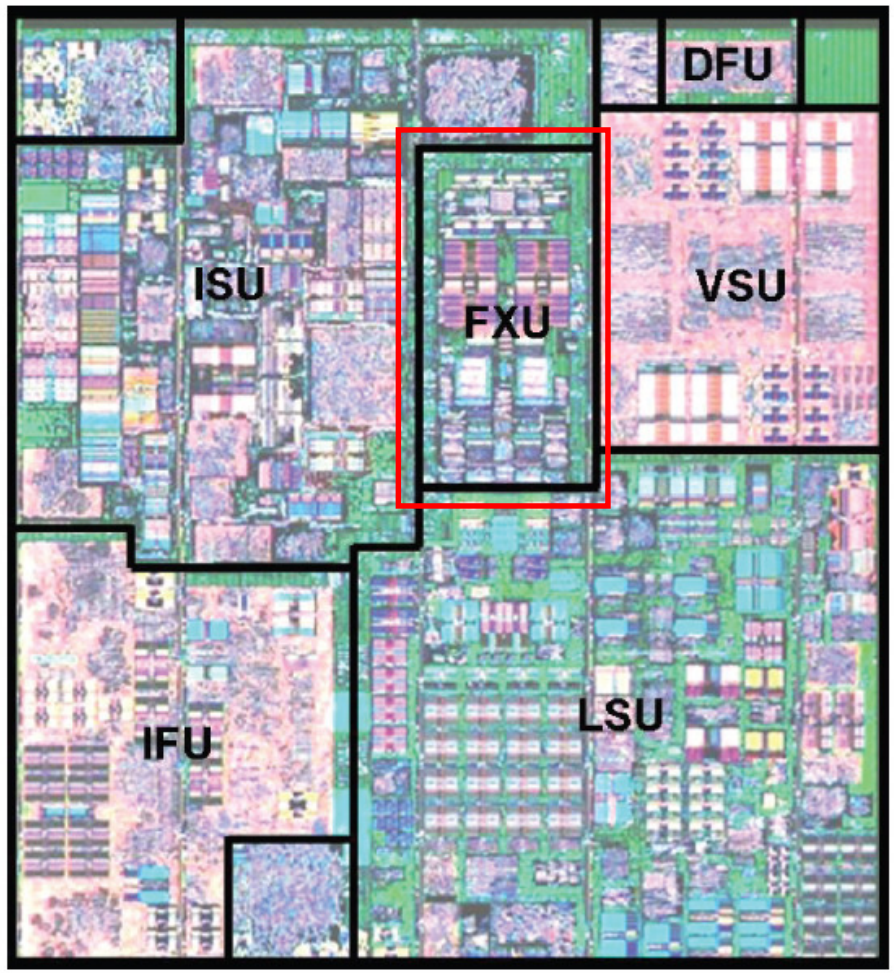

Figure 1

POWER8 processor core floorplan.

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.

Fixed-Point Unit (FXU)

The Fixed-Point Unit (FXU) is composed of two identical pipelines (FX0 and FX1).

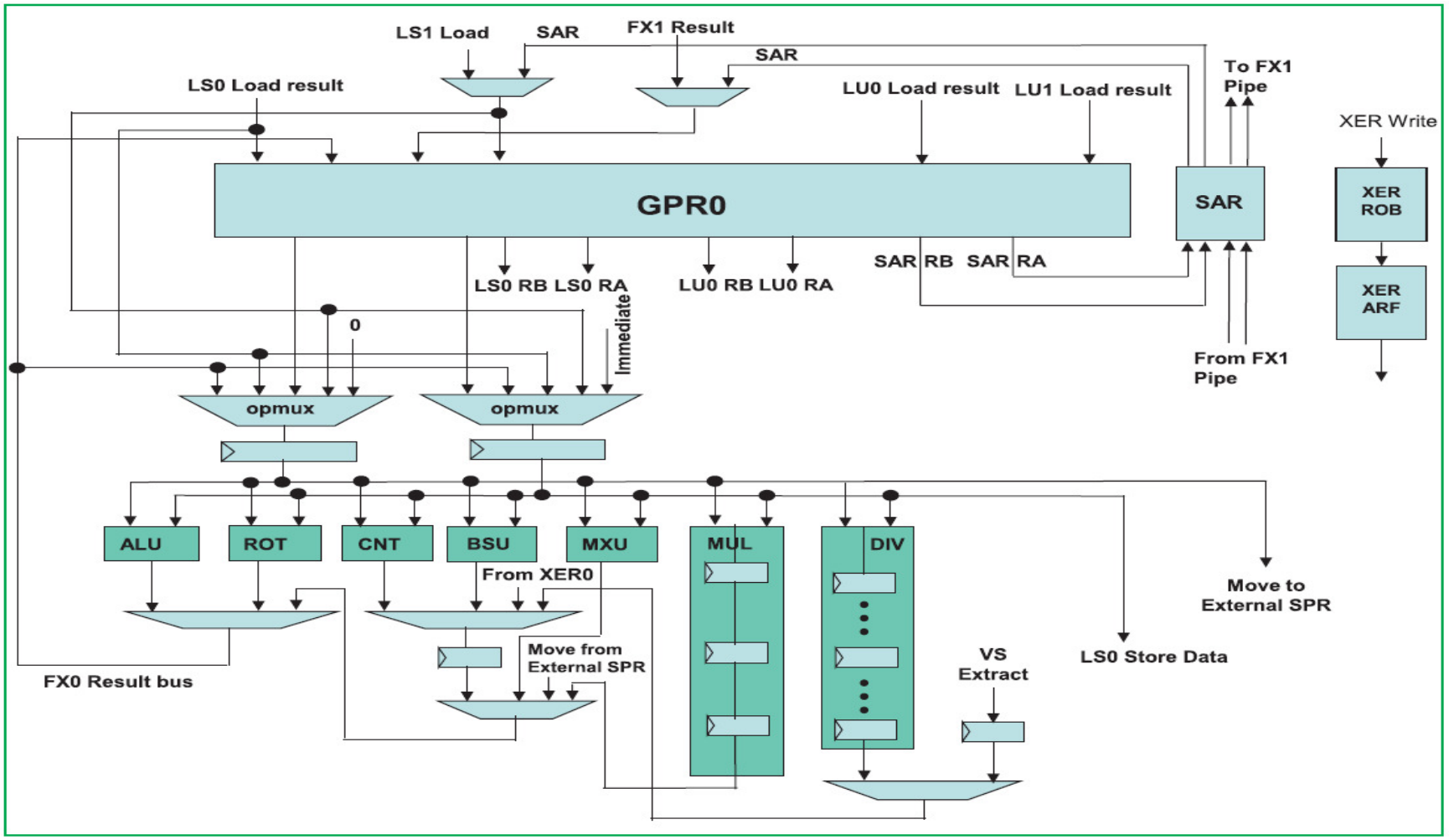

Figure 7

POWER8 FXU overview (FX0 pipe shown).

Fixed-Point Unit (FXU) *(continued)*

As shown in Figure 7, each FXU 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)

Fixed-Point Unit (FXU) *(continued)*

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.

Vector-and-Scalar Unit (VSU)/Decimal Floating Point Unit (DFU)

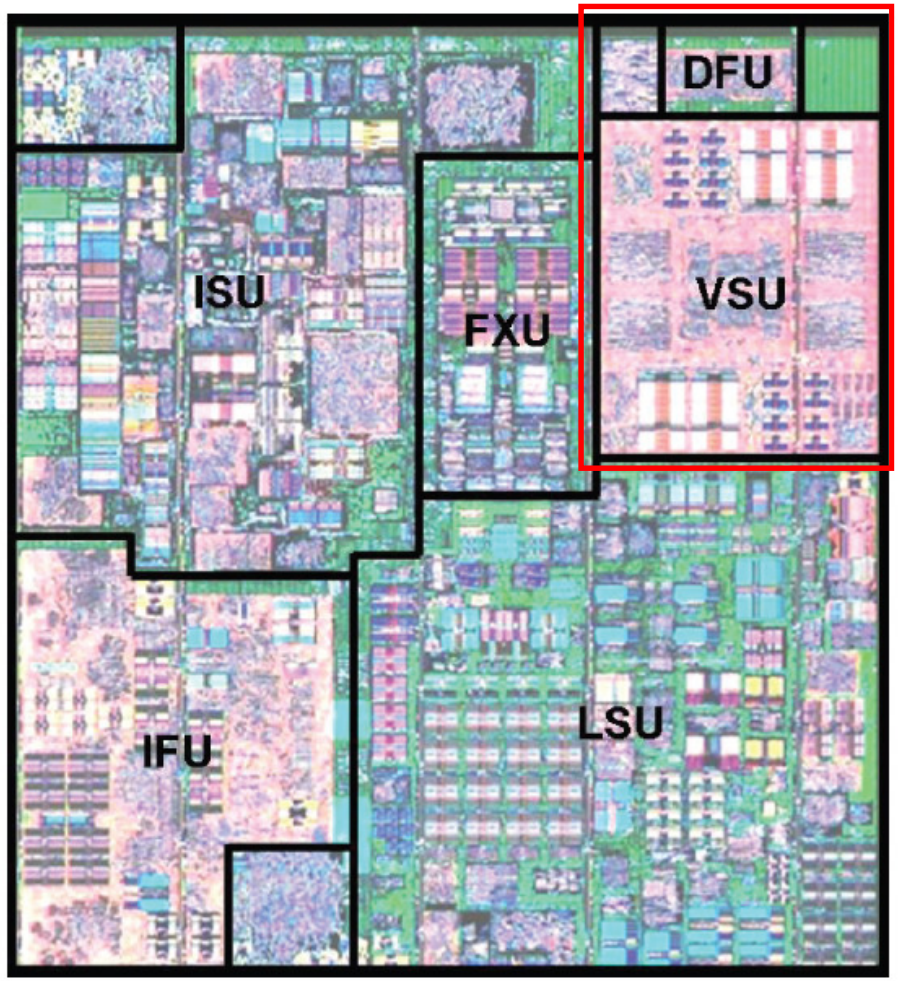

Figure 1

POWER8 processor core floorplan.

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.

The POWER8 processor **Vector-and-Scalar Unit (VSU)**, shown in Figure 8, **has been completely redesigned** from its initial implementation in the POWER7 processor **to support the growing computation and memory bandwidth requirements of business analytics and big data applications**.

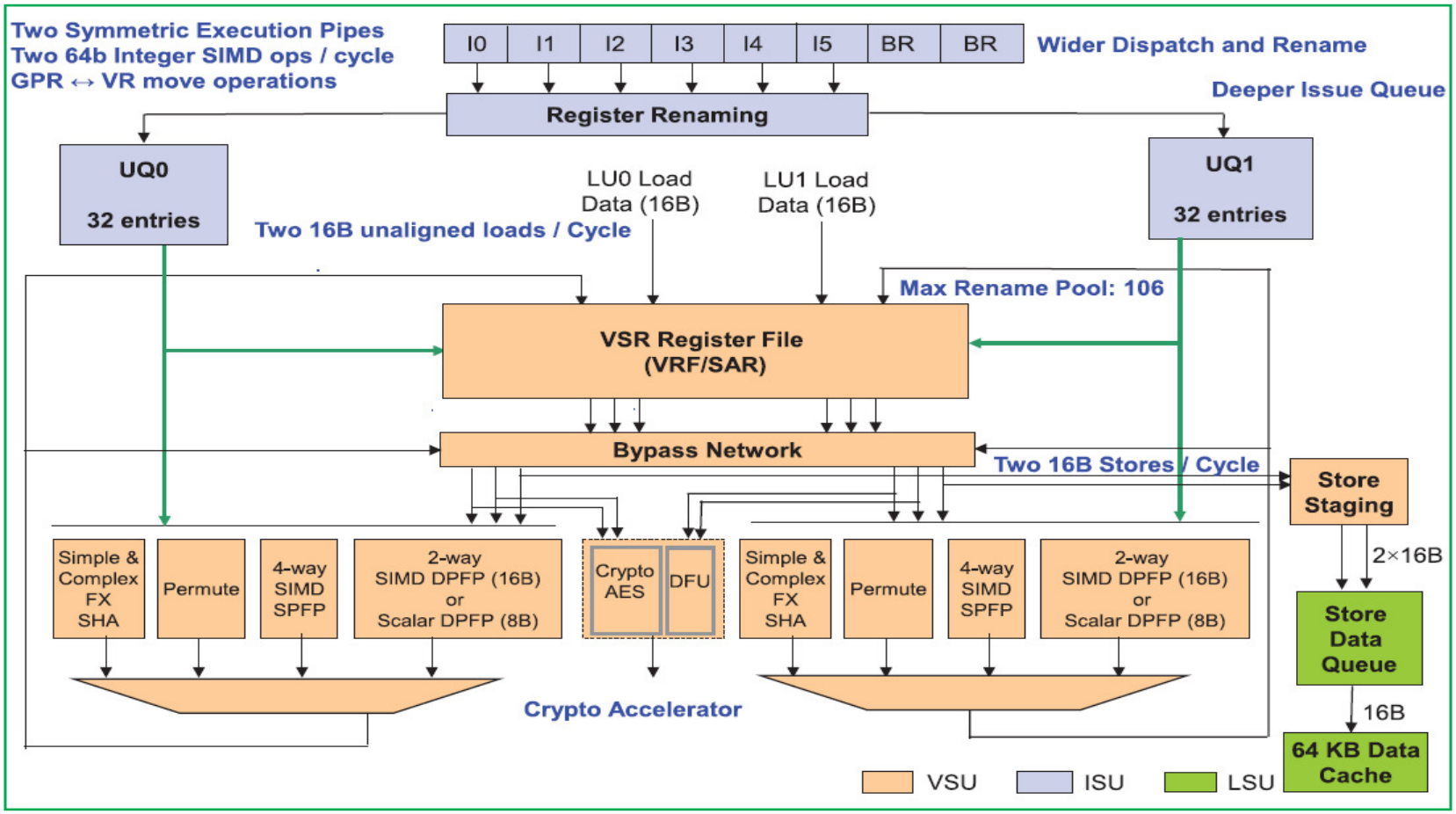

Figure 8

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

The POWER8 VSU now supports **dual issue of all scalar and vector instructions of the Power ISA.**

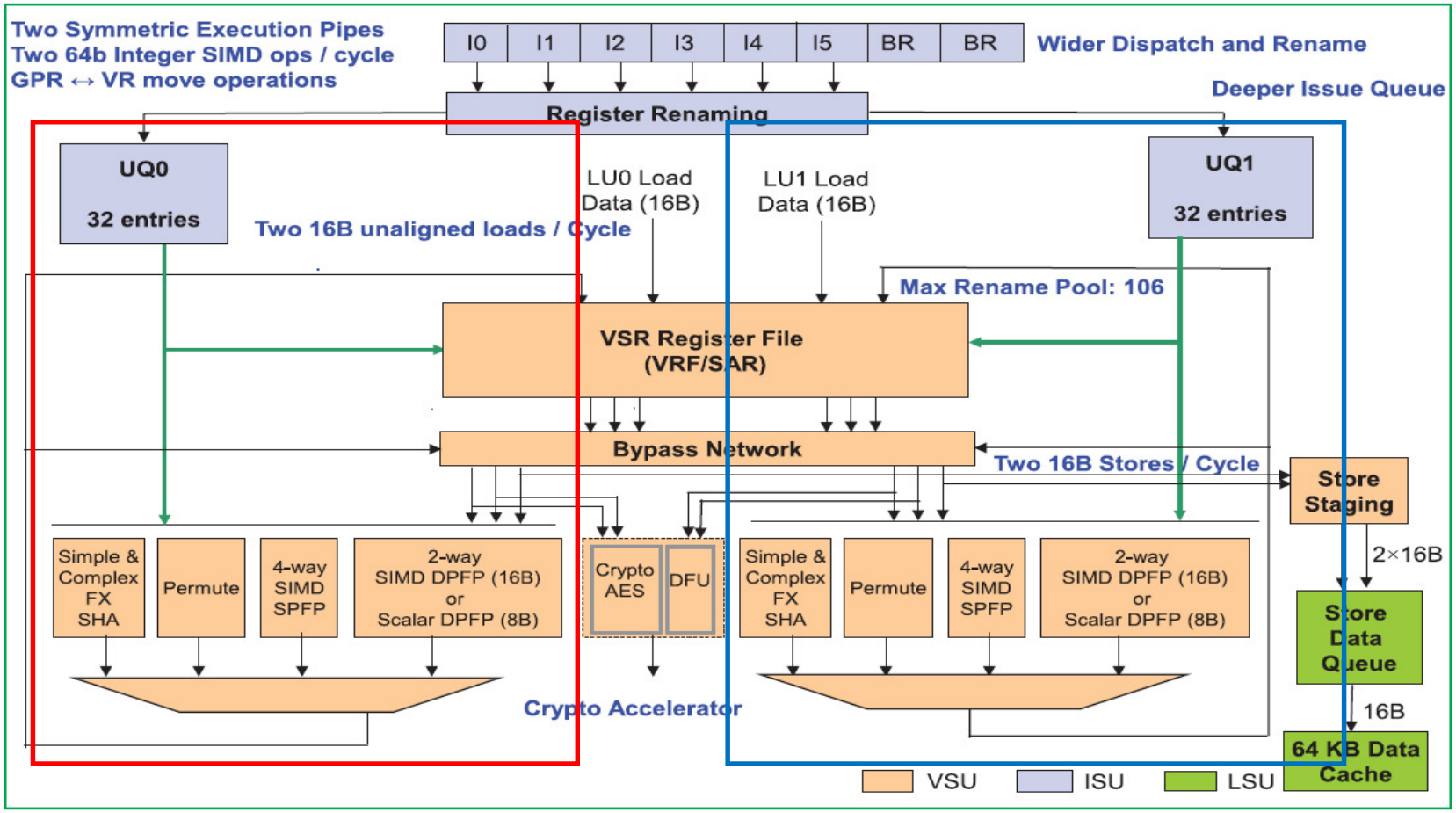

Figure 8

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

Vector-and-Scalar Unit (VSU) *(continued)*

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
- execution of all floating-point compare instructions using the two-cycle Simple Unit (XS) pipeline to speedup branch execution

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.

Vector-and-Scalar Unit (VSU) *(continued)*

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.

Vector-and-Scalar Unit (VSU) *(continued)*

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.

Decimal Floating Point Unit (DFU)

The Decimal Floating Point Unit (DFU) in the POWER8 core allows fully pipelined execution of the Power ISA “Decimal Floating Point” instructions.

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.

VSU and DFU

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.

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.

Summary and Conclusion

…

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.

Today, with an established history of high-performance success, POWER8 has proved it **"can provide about 1.5 times more single thread performance and twice the throughput performance over a POWER7 core."**

So much so, the nature of my work with performance-tuning POWER8/AIX workloads is substantially different from POWER5/6/7. Bluntly, it runs so fast, it covers for a host of past indiscretions.

No matter, there will always be new indiscretions. The enterprise will soon evolve and grow workloads to tax even POWER8's amazing capabilities.

Thank You

Prepared and Edited by Earl Jew (not an author of the whitepaper)
earlj@us.ibm.com; (310) 251-2907; Los Angeles, California, USA
Senior IT Consultant for IBM Power Systems and System Storage
IBM STG Lab Services Power Systems Delivery Practice

Balaram Sinharoy (an author of the whitepaper)

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 (an author of the whitepaper)

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 (an author of the whitepaper)

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 (an author of the whitepaper)

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 (an author of the whitepaper)

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 (an author of the whitepaper)

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 (an author of the whitepaper)

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 (an author of the whitepaper)

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 (an author of the whitepaper)

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.

Jose´ E. Moreira (an author of the whitepaper)

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 (an author of the whitepaper)

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 (an author of the whitepaper)

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 (an author of the whitepaper)

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 (an author of the whitepaper)

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 (an author of the whitepaper)

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 (an author of the whitepaper)

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 (an author of the whitepaper)

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 (an author of the whitepaper)

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 (an author of the whitepaper)

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 (an author of the whitepaper)

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.