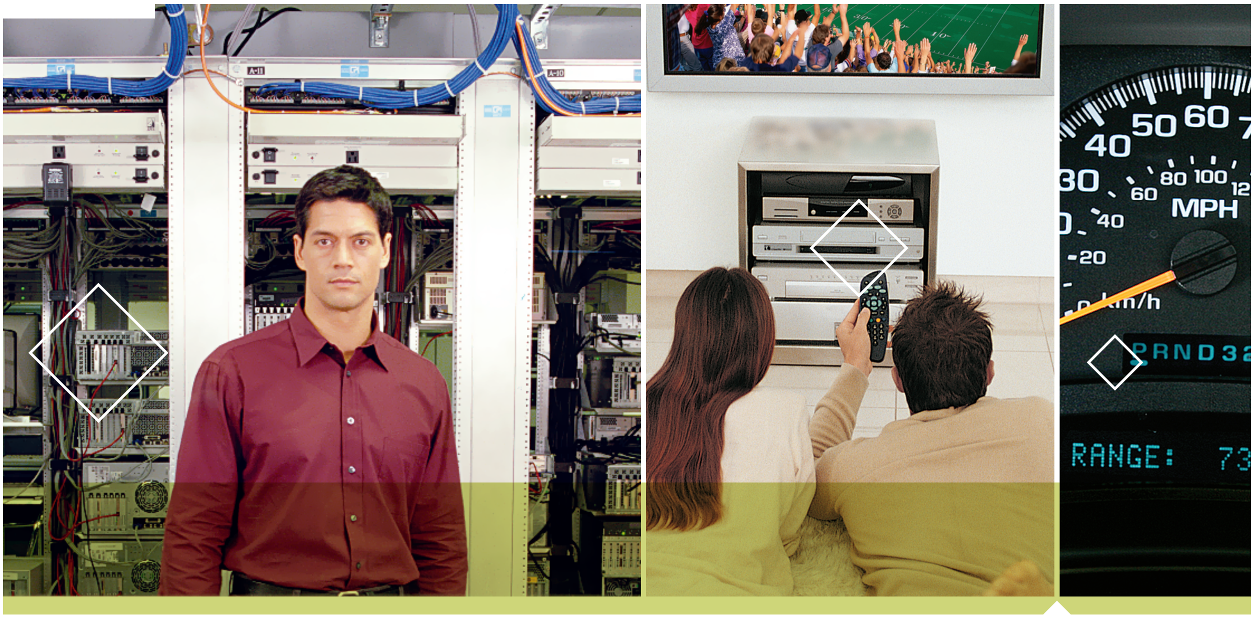
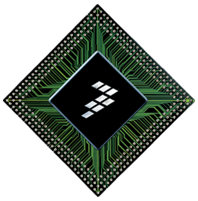


Power Architecture™ Technology Primer



Power Architecture™ technology addresses a wide range of implementations from high-performance general purpose processors to revolutionary communication processors and highly integrated embedded microcontrollers. This book offers an introduction to Power Architecture technology as it applies to the amazingly diverse world of Freescale microprocessors and microcontrollers.

freescale.com/powerarchitecture

Power Architecture™ Technology Primer

PWRARCPRMRM
Rev. 1
05/2007

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or
+1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064
Japan
0120 191014 or
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
 Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
+1-800 441-2447 or
+1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor
 @hibbertgroup.com

Document Number: PWRARCPRMRM
Rev. 1, 05/2007

Contents

Contents

Section	Title	Page

Coevolution—Power Architecture™ Technology and its Environments

When computing first slipped beyond the corporate mainframes and into small businesses and homes, the focus was mainly on making processors smaller and faster. In 1975, the first computers billed as ‘portable’ weighed in at a svelte 50 pounds, a twenty-fold reduction over the PC’s half-ton ancestor of the 1960s. Desktop computing begat personal computing, which ever since has grown persistently more and more personal.

What had been a cloistered domain of tapes and mainframes vaulted in an inner sanctum and tended by an inner circle of gnostics in white labcoats, evolved into style-challenged beige office appliances, and now entertaining and useful techno-bling that fits into our kids’ pockets and grandmothers’ purses. Typewriters, floppies, adding machines, and the plain old telephone have become extinct; they are replaced by whole systems on chips—digital minotaurs, sphinxes, jackalopes, and the other fantastic electronic chimeras.

When Alan Turing was trying to get an understanding of what was computable, he probably wasn’t thinking of such a proliferation of computing niches, microniches, and milliniches of the taxonomy—family: *computing*; genus: *network*, *personal*, *scientific*, *enterprise*, and *distributed*; species: *massively parallel*, *pervasive*, *palpable*, *ubiquitous*, *nomadic*, *wearable*, and so on.

Although miniaturization of processes and efficient designs made it possible to get more chips per die, it also made room first for on-chip caches, memory controllers, and additional coprocessors to move on-board. As the computing environment has capitalized on the new possibilities for further integration, processors have responded with a continual speciation and hybridization that has made the buzzword 'ecosystem' unavoidably apt for a market where the system is now on the chip.

Essential to such a dynamic ecosystem that offers so many possibilities is a comprehensive, adaptable, and coevolving computing architecture—Power Architecture™ technology. The Power Architecture technology, which has undergone thoughtful evolution over the past 15 years, is taking another significant step forward on the evolutionary path.

Through the work of the Power.org™ Power Architecture Advisory Council (PAAC), the Power Architecture specification, released in 2006, represents a merging of the previous PowerPC™ architecture specifications, structured in a way that preserves the consistency of the base user-level programming model across the environments—desktop computing, embedded, and server—yet provides the flexibility to adapt to changes in computing environments and in process technologies.

The name change reflects the formal broadening of the architecture’s scope. The Power Architecture technology maintains the original PowerPC architecture 1.10, and adds the Power ISA™ technology, which defines equivalent resources for both embedded and server devices. Details about the structure of the architecture are provided in “Stability, Flexibility, Familiarity” on page 7.

To more effectively address the persistent need for multiple, niche-specific architectural components, the Power Architecture technology extends the modularity built into the original layered architecture specification (Books I through III) by breaking the functionality of the architecture into components called “categories.”

The broadest categories define basic functionality common across computing environments, as follows:

- The Base category defines all of those elements common to all Power Architecture processors. Although it includes functionality defined in all three books, the Base category preserves almost all of the user application-level resources defined in the original PowerPC Book I, the user instruction set architecture (UISA). Other features from the original UISA, such as the floating-point and move assist instructions, are preserved as separate categories.
- The Embedded and Server categories define mutually exclusive resources appropriate for those environments. This document focuses on the functionality defined for Freescale's embedded processors.

Other categories address more specific features, such as the AltiVec technology (referred to as the Vector Category in the architecture) and the signal processing engine (SP category).

Some of these special features were optional in the PowerPC architecture. Others were previously defined as auxiliary processing units, or APUs, and were not part of the architecture. Many of those former APUs, began life as part of Freescale's embedded implementation standards (EIS), a layer of architecture for features common to Freescale processors, but outside of the formal architecture specification. The EIS continues to define such features, many of which are described in this document.

Allowing such special-purpose categories makes it possible to further extend the Power Architecture programming model to support new ecosystems for the new species of computing that may just now be bubbling around in the back of your mind.

The original PowerPC UISA remains at the center of the architecture. And it is the ability to extend its efficient, RISC-based application-level programming model into new computing spaces where much of the architecture's power resides.

The Power.org Community

The Power.org community, announced in 2004, is the open standards organization for developing, enabling, and promoting Power Architecture technology and specifications. It represents an international cross-section of semiconductor and electronics organizations including SoC firms, tool vendors, foundries, OS vendors, OEMs, independent hardware vendors, independent software vendors, and service providers; as well as individual developers, educational institutions, and government organizations.

The Power.org community's objectives are to develop standards and specifications, validate implementations, drive adoption of Power Architecture technology, and enable a complete design and manufacturing infrastructure that will resolve many of the technology and business issues hindering hardware development and innovation.

History: The PowerPC Architecture

The original PowerPC architecture developed by architects from Apple, IBM, and Motorola was rooted in IBM POWER™ architecture. It is notably RISC-based—a load-store, register-to-register architecture. Memory accesses are decoupled from computational instructions, which use on-chip registers to hold source and destination operands.

Although the first PowerPC architecture specification was crafted specifically for desktop systems, it was written as three books, to distinguish the application- and system-level programming models:

- Book I, the user instruction set architecture (UISA), defined the application-level programming model for all PowerPC devices. The PowerPC UISA represents the unchanging mitochondrial DNA, defining the application-level instruction set and programming model common across all architectural variants and preserved by the Power ISA definition.
- Book II, the virtual environment architecture (VEA), defined resources that support the time-base aspects of the memory model, and features for multiprocessor implementations; it is preserved by the Power Architecture definition.
- Book III, the operation environment architecture (OEA), defined operating system–level facilities such as memory translation and interrupts for desktop implementation.

The modular structure made it possible for other architecture-compliant devices, such as the first embedded PowerPC processors, to leverage the PowerPC UISA without being constrained to Book III's desktop-oriented operating system features.

The PowerPC Architecture Matures

The PowerPC architecture set aside ample register and opcode space both for implementation-specific resources and for formal extensions to the architecture. The very first processors, such as the MPC601 in 1993, implemented registers and register fields that were not defined in the architecture.

Every design since has had implementation-specific, special-purpose registers (SPRs) such as the hardware-implementation dependent (HID) registers used for configuration, or unique SPRs for diagnosing errors or optimizing software and hardware designs.

Some of these features have gradually become part of the architecture. For example, the MPC604 introduced a performance monitor facility that made it possible to characterize instruction and data traffic by counting cache misses, interrupts, page faults, and other events. Capturing such information made it possible to fine-tune software and hardware designs. The Power Architecture specification includes separate performance monitor categories for embedded and server devices.

The Book III interrupt model left some details up to the implementation as to whether hardware would automatically handle exception conditions, such as misalignment, or whether those conditions would take an interrupt. Additions to the programming model, such as the performance monitor and the AltiVec™ technology (Vector category), added interrupts.

Many implementation features were passed on to subsequent processors to become family traits, some of which are now part of the architecture, as is the case with many Freescale-defined APUs, such as the performance monitor and cache-locking categories.

All of this made it easy for the architecture to respond and improve, and for the Power.org community to put the UISA to use in different environments. The following sections trace how the original PowerPC architecture met the needs of computing trends and evolved into today's Power Architecture technology.

Architectural Extensibility—Alternatives to Book III's Hardware-Based MMU Model

As the MPC601 and MPC603 were drawing attention as the processors for Apple's Macintosh® computers based on PowerPC technology, Freescale designers were using the PowerPC UISA as the application-level programming model for its 5xx family of embedded cores, which were integrated into automotive processors and the first generation of PowerQUICC™ devices. Instead of the block-address translation (BAT) and the hardware-driven, fixed-page address translation prescribed by Book III, the 5xx cores provided a software-driven translation mechanism that supported variable page sizes.

The MPC603 processor, used in the Apple G2 Macintosh computers and still thriving as Freescale's e300 embedded cores, retained block and page memory structure, but forwent the Book III hardware translation model by defining instructions for accessing the TLBs directly—Load Data TLB (**tlbld**) and Load Instruction TLB (**tlbli**) instructions.

Although the e600 family continues to implement the PowerPC 1.10 Book III MMU, the software page address translation begun in the 5xx and MPC603 was refined and systematically described by Book E and by Freescale's EIS (Book E implementation standards), which now comprise the Freescale MMU component (category.Embedded.MMU Type FSL) of the Power ISA definition.

Architectural Extensibility—AltiVec Technology

While PowerPC technology proved itself to be a workhorse architecture in the scientific, workstation, and embedded environments, it also proved to be a playful one. As computer games, 3D animation, and video processing placed a new set of demands on personal computing, the architects responded with the AltiVec SIMD (single-instruction/multiple-data) instruction set. This first major extension to the PowerPC architecture came with the introduction of the MPC74xx processors that powered Apple's G4 Macintosh computers.

It is worth noting that AltiVec technology was never formally a part of the PowerPC architecture, although it used PowerPC instruction formats and syntax and occupied the opcode space expressly allocated for such purposes. AltiVec extended the PowerPC programming model to provide 128-bit, multi-element, vector operations. In the Power ISA definition, the AltiVec technology is referred to as the Vector category. For details, see "AltiVec Technology (Category.Vector)."

The fourfold, parallel replication allows execution of the same computational operation across four parallel data elements. Here, computational logic of scalar execution units is replicated, but the resulting performance increases have proven to greatly outweigh the increase in die size and microarchitectural complexity for those environments that need such improved performance.

The AltiVec programming model provided a prototype for the concept of the auxiliary processing unit (APU), a concept central to Book E, in that it made it possible to create special-purpose extensions to the base PowerPC architecture without permanently committing limited architectural resources, such as opcode and register space. After the concept of APUs was introduced, the AltiVec technology became a Freescale APU and part of Freescale's EIS.

Architectural Extensibility, Phase II—Book E, APUs, and Freescale's EIS

The sustained growth of the embedded market drove a need for an alternative architecture to the desktop-oriented PowerPC architecture. This took the form of Book E, which, unlike the three-book structure of the PowerPC desktop architecture, was a single book, making it possible to define functionality with both user- and supervisor-level components in a single place.

Book E also provided alternatives to the MMU model defined in Books II and III, specifically to address those issues that guided the 5xx cores to use a software-oriented, page-based translation mechanism that strayed from Book III, and similarly gave rise to the MPC603's definition of new instructions for configuring TLBs directly via software.

Where the PowerPC 1.10 architecture supports hardware-based page address translation with fixed 4-Kbyte pages, the Book E MMU, which is now part of the Power Architecture Book III-E definition, is strictly software managed and supports fixed and variable page sizes. Where the PowerPC 1.10 architecture defined block address translation (BAT) SPRs that could provide a single translation for large blocks of memory space, in Book E processors this is done with variable sized pages. For more information, see "Memory Management Unit (MMU) Model" on page 44.

Auxiliary Processing Units (APUs)

The ability to extend the ISA systematically became a greater consideration in the PowerPC Book E architecture design philosophy. Book E defined reusable opcode space that can be used by multiple APUs. For example, opcodes for instructions defined by the signal-processing engine (SPE) implemented on the e500 and e200 cores overlap the opcode space defined by the AltiVec extension.

In Book E, APUs could be as simple as a single instruction or register, or a set of fields within a register defined by the PowerPC architecture. They could also be as rich as the SPE APU, which defines new instructions, new registers, new fields within existing registers, and new interrupts. Many APUs have become part of the Power ISA as special-purpose categories.

The Freescale Book E Implementation Standards (EIS)

The PowerPC 1.10 architecture was broken into three books to allow devices to implement the application-level features of Book I UISA common across the spectrum of computing environments, without having to adhere strictly to the desktop-specific features defined in Book III. Likewise, leaving space in the opcode and SPR maps for implementation-specific purposes was to provide a platform for extending the program model.

Book E formally addressed the need for consistency among such devices, defining a framework for a non-segmented, page-based MMU and defining allocated space for programming model extensions such that consistency and reuse could be enforced without restricting innovation. Book E defined many features in a more general way, leaving many details to the implementation. For example, the Book E MMU model defined the TLB Read Entry and TLB Write Entry instructions (**tlbre** and **tlbwe**) for reading and writing the TLBs in software. However, details of how this is accomplished are defined as implementation dependent. Rather than leaving this up to individual implementations, the Freescale architects defined more specifically that these instructions transfer the contents of a set of MMU assist (MAS) SPRs into the TLBs. This definition became part of the Freescale Book E Implementation Standards (EIS).

These EIS-defined MAS registers provide the translation, protection, byte-ordering, and cache characteristics for the relevant pages, and the exact behavior of the **tlbre** and **tlbwe** instructions was defined by the Freescale Book E implementation standards (EIS). These registers are now part of the Embedded.MMU Type FSL.

The following EIS-defined features are now categories in the Power ISA definition:

- The MMU model, in particular the use of MAS registers (Embedded.MMU Type FSL)
- The cache-locking APU (Embedded.Cache Locking)
- Major extensions to the ISA
  - The AltiVec APU (Vector category)
  - Signal-processing engine APU (Signal Processing Engine category)
  - Embedded floating-point APUs (Now the Single-Precision Vector and Scalar categories (SP.FV and SP.FS) and the Double-Precision Scalar category (SP.FD)

Although much of the EIS defined under Book E is now part of the Power Architecture definition, the EIS continues to evolve, allowing Freescale to define its own categories that address the continually diversifying needs of the embedded ecosystem. As new features are added to the EIS, many will be submitted to the Power Architecture Advisory Council for consideration for inclusion in future releases of the Power ISA.

Architectural Extensibility Phase III—The Power ISA Definition

While Book E and Freescale's EIS were putting a finer point on the PowerPC architecture to allow the UISA to provide more detailed programming options within the variety of embedded niches, IBM was extending the architecture into higher-end server devices by developing an alternative to the desktop-oriented resources of the PowerPC 1.10 architecture, Book III. The merging of those two versions of the architecture into a broader and much more modular architecture based around the UISA ensures architectural hardiness and vitality.

The Power ISA retains the three-book structure of the original PowerPC architecture, further organizing the common functionality into general-purpose categories. More significantly, the Power ISA now incorporates those extensions defined in Book E and the EIS as special-purpose categories.

The architectural components common to all Power ISA–compliant devices comprise the broadest of these categories, the Base category. For example, load and store instructions and standard integer computational instructions defined in the original PowerPC Book I definition are now part of the Base category, along with most other UISA features, such as the GPRs, condition register (CR), and link register (LR). These particular features of the Base category are defined in Book I, but the Base category defines some features outside of Book I, such as the Book II time base.

Because of the very different requirements for memory management and interrupt handling in embedded and server environments, the Power ISA defines a separate Book III for each category—Book III-E for Embedded category processors and Book III-S, for Server category processors.

Many special-purpose categories include features defined across multiple books. For example, user instructions defined by the Vector and Signal Processing Engine categories extend Book I, while the interrupt resources associated with these categories are defined in Book III-E.

Some of these special-purpose categories are specific to either the Server or Embedded environments. For example, the Embedded Performance Monitor category (E.PM) can be implemented only in Embedded category devices. Embedded-specific categories are described in the section, "The Embedded Category," on page 13.

The Power Architecture technology of today is the product of the vision of the PowerPC architects of 1990—an architecture with a sound, reliable, and pervasive application base; an architecture well positioned to adapt as the computing environment broadens into even more diverse ecosystems; an architecture that is stable, flexible, and familiar.

Stability, Flexibility, Familiarity

Although there is very little in the way of newly architected features in the embedded environment of the Power ISA, the reorganization makes the specification quite different from both the PowerPC 1.10 and Book E architecture specifications. The resources that are now part of this merged architecture are not new to the thousands of software and hardware designers and tool vendors who have been familiar with PowerPC devices for the past 16 years.

Book I and Book II have been reorganized and amended, with features essentially unchanged from Book E, the EIS, and IBM's PowerPC 2.02 specification for server processors. Although Book III-E and Book III-S still bear a family resemblance to the original PowerPC Book III definition, they differ in very significant ways both from one another and from the PowerPC 1.10 specification.

The Power ISA definition organizes the specification into shared Books I and II (what the Power Architecture definition refers to as the Base category because these resources are common to both), and separate Book IIIs, as shown in Figure 1.

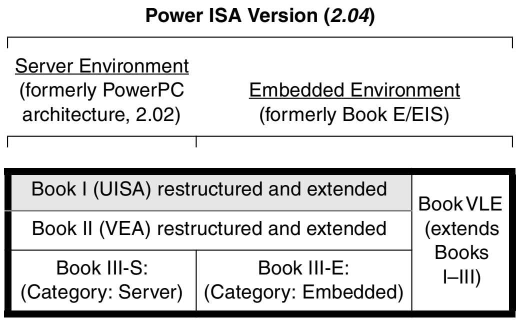

Power ISA Version (*2.04*)
Server Environment
(formerly PowerPC architecture, 2.02)
Embedded Environment
(formerly Book E/EIS)
Book I (UISA) restructured and extended
Book II (VEA) restructured and extended
Book III-S: (Category: Server)
Book III-E: (Category: Embedded)
Book VLE (extends Books I–III)

Figure 1. Power ISA Version (*2.04*)

The PowerPC architecture is the grandfather to the latest generation of the Power Architecture technology. Figure 2 shows the relationship between the different environments.

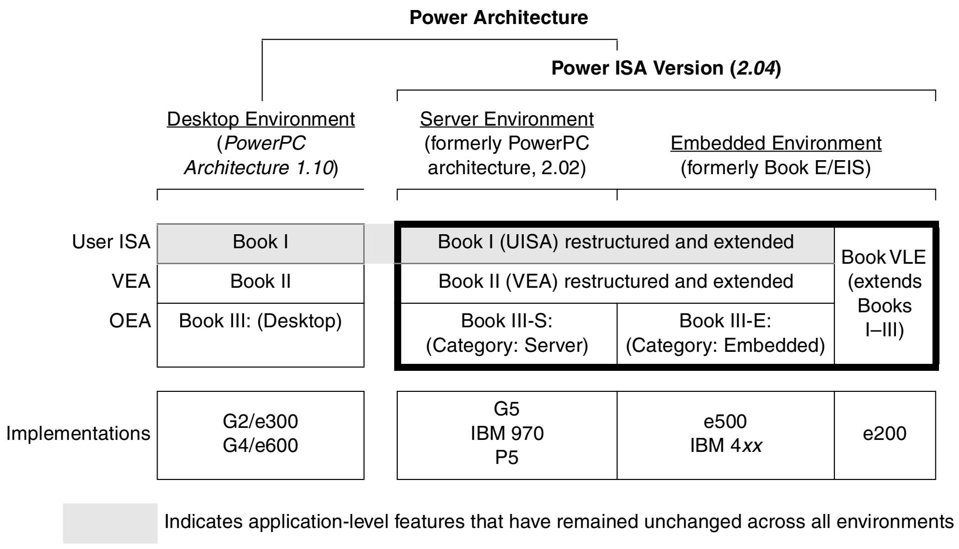

Power Architecture
Power ISA Version (*2.04*)
Desktop Environment
(*PowerPC Architecture 1.10*)
Server Environment
(formerly PowerPC architecture, 2.02)
Embedded Environment
(formerly Book E/EIS)
User ISA
VEA
OEA
Book I
Book II
Book III: (Desktop)
Book I (UISA) restructured and extended
Book II (VEA) restructured and extended
Book III-S: (Category: Server)
Book III-E: (Category: Embedded)
Book VLE (extends Books I–III)
Implementations
G2/e300 G4/e600
G5 IBM 970 P5
e500 IBM 4*xx*
e200
Indicates application-level features that have remained unchanged across all environments

Figure 2. Power Architecture Relationships

The definitions that comprise the Power Architecture technology are as follows:

- The PowerPC instruction set architecture (ISA) 1.10. The original architecture defined in the 1990s by Apple, IBM, and Motorola's semiconductor products sector (SPS) (now Freescale). This mature architecture continues to form the basis for developing PowerPC processors that use Freescale's G2, e300, and e600 processor cores.
- The Power ISA 2.04 specification (April 2007). The Power ISA 2.04 specification It brings together the embedded features defined in Book E and the Freescale EIS with the server and desktop resources defined by IBM's PowerPC architecture 2.02 definition.

The first published version of this merged architecture (version 2.03) mostly reflects functionality that has become familiar through the Book E–based devices such as the Freescale e200 and e500 cores and the IBM 970 processor. It includes newly architected features that will appear in forthcoming processors. Subsequent versions of the architecture will include additional features. The Power ISA 2.04 specification differs from Power ISA 2.03 specification with the addition of several Server category features.

As Figure 2 shows, processors designed under the Book E/EIS and PowerPC advanced server 2.02 architectures remain compliant with the restructured Power ISA architecture.

A modular specification based around the UISA's sturdy and stable foundation makes the Power Architecture model poised to respond and adapt to, as well as to drive, innovation in a computing environment that continues to grow more diverse.

What's New?

The PowerPC Book E architecture, Freescale's EIS, and the PowerPC 2.02 architecture are merged and reorganized, with several additional extensions (categories) that are described in Power ISA 2.04, which is available for download from power.org.

What Has Changed?

Few of the features in the merged architecture are truly new; rather, most were defined by the PowerPC (AIM), PowerPC 2.02, PowerPC Book E, Freescale EIS architectures, and IBM 4xx designs.

In particular, the EIS-defined MMU model, many APUs, and the VLE extension to the architecture defined by Freescale's EIS have gone from being Freescale-specific architecture to becoming categories within the Power Architecture model. What is new is how these different architectures have been joined under a new name that reflects the expansive reach of the diversified architecture. This section describes how these features are incorporated into the Power Architecture model as categories. Detailed descriptions of these categories are described in "An Overview of Categories" on page 13.

Book I Changes and Extensions

In the Power Architecture model, Book I has been extended with the incorporation of the following categories that were formerly EIS APUs:

- The Integer Select (**isel**) instruction. Analogous to the Floating-Point Select (**fsel**) instruction defined by the PowerPC architecture, **isel** is used to eliminate short conditional branch code segments by specifying two source registers and one destination register for a comparison. Under the control of a specified condition code bit, **isel** copies one or the other source operand to the destination. **isel** reduces program latency and code footprint.

- Signal processing engine (SPE). A comprehensive set of 64-bit, two-element, SIMD instructions that share the Book I–defined GPRs extended by the SPE to 64 bits, as shown in Figure 3.

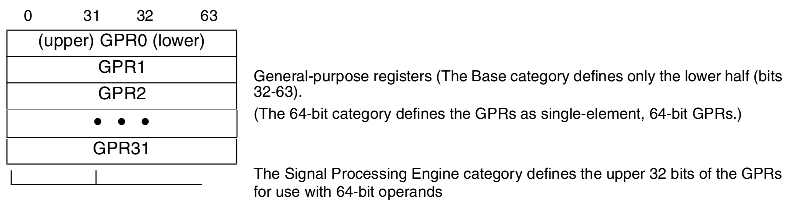

0
31
32
63
(upper) GPR0 (lower)
GPR1
GPR2
• • •
GPR31
General-purpose registers (The Base category defines only the lower half (bits 32-63).
(The 64-bit category defines the GPRs as single-element, 64-bit GPRs.)
The Signal Processing Engine category defines the upper 32 bits of the GPRs for use with 64-bit operands

Figure 3. Extended GPRs

- The SPE defines three dependent embedded floating-point categories:
  — SPE.Embedded float scalar double (SP.FD)
  — SPE.Embedded float scalar single (SP.FS)
  — SPE.Embedded float vector (SP.FV)

  The Signal Processing Engine category also extends Book III–defined features, in particular, the interrupt model.

  This category is implemented in the e200 and e500 cores.
- Variable length encoding (VLE). Variable-length encoding facility that reencodes opcodes from other categories to fit into 16 bits. Although it extends the UISA programming model, the VLE category is specified in a separate book, Book VLE.

  This category is implemented in some e200 cores. See “Book VLE Category” on page 17.
- Vector. AltiVec technology was introduced in 1998 as an extension to (but not formally a part of) the PowerPC architecture. This comprehensive 128-bit, four-operand SIMD ISA consists of 168 instructions, a set of 32, 128-bit vector registers (VRs), the vector save register (VRSAVE), and the vector status and control register (VSCR), which is analogous to the FPSCR. Like VLE and SPE, the Vector category also extends the Book III interrupt model. For more information, see “Architectural Extensibility—AltiVec Technology” on page 4.

The following resources, defined as part of the PowerPC UISA, have been identified as distinct categories and, as such, are not part of the required Base category:

- Floating-point (FP). Consists of the floating-point instructions, registers, and interrupt resources defined in the PowerPC architecture. In the Power Architecture model, the floating-point record forms are defined as a dependent category, Floating-point.Record (FP.R). See the section, “Floating-Point Categories—Floating-Point (FP) and Floating-Point with Record (FP.R),” on page 15.
- Sixty-four bit (64). The 64-bit portion of the PowerPC architecture 1.10 definition has been carried forth as a separate category (64). For Embedded category devices, this moded address mechanism replaces the non-moded 64-bit component of the Book E architecture. This document does not describe features of the 64-bit category.
- Move assist (MA). Consists of the four load store string instructions **lswi**, **lswx**, **stswi**, and **stswx**. Because these instructions duplicate functionality otherwise defined in the architecture, and

because in some environments they may present additional latency problems, they have not been implemented on recent Freescale devices.

Book II Changes

- Alternate time base (ATB). An additional time base analogous to the PowerPC time base (Book II). Implemented on the e500v2.
- External control (EXC). Consists of the External Control In Word Indexed (**eciwx**) and External Control Out Word Indexed (**ecowx**) instructions and the external access register (EAR) defined in the PowerPC Book II definition.
- Support for true little-endian byte ordering, replacing the original little-endian byte ordering, which remains only as part of the PowerPC architecture 1.10. The Embedded category defines the per-page specification of endianness defined in Book E; in the Server category, endianness is specified by a mode, as it was in the PowerPC architecture, 1.10.
- The Book E–defined **msync** instruction has reverted to being the Synchronize instruction **sync** defined by the PowerPC architecture. In the Embedded category, **msync** is a simplified mnemonic for the **sync** instruction to ensure compatibility with the Book E **msync**. The **mbar** instruction, defined in Book E, remains as part of the Embedded category; the equivalent PowerPC architecture 1.10 instruction **eieio**, which shares the opcode, is defined as part of the Server category.

Book III Changes

- The following categories are dependent categories of the Embedded category (abbreviated as E). For example, E.MF identifies that the embedded memory management model, originally defined by Book E and the EIS, are now a category that can only be implemented as part of an Embedded category device. These categories in particular indicate functional characteristics specific to the existing families of Power Architecture processors.
  - Embedded.MMU type FSL (E.MF). Inherited from Book E and Freescale’s EIS. Defines MMU assist (MAS*n*) SPRs (from the EIS) for loading and storing configuration information into the TLBs using the Book E–defined TLB write and read entry instructions (**tlbwe** and **tlbre**) (Book III-E). Implemented in the e500 cores. The section, “Memory Management Unit (MMU) Model,” on page 44, compares the PowerPC architecture 1.10 MMU with the Book III-E category.
  - Embedded.cache locking (E.CL). Defines a set of instructions for locking and clearing cache lines. Implemented in the e500 cores.
  - Embedded.enhanced debug (E.ED). Defines a separate set of interrupt save and restore to provide greater responsiveness for debug interrupts. Implemented in e200 and e500 cores.
  - Embedded.performance monitor (E.PM). Consists of the instructions, registers, and interrupt model defined by the EIS performance monitor APU. Includes definition of separate performance monitor registers (PMR) (Book III-E). Implemented in the e200 and e500 cores.

— Interrupt-related features associated with non-base categories such as vector, SPE, VLE, and performance monitor.

- Additional software-use SPRs (XSR). Extends the number of software-use SPRs (SPRG8–SPRG9). The Base category defines SPRG0–SPRG3; the Embedded category defines SPRG4–SPRG7.

The Server category includes additional categories not described here.

Power Architecture Details

This section provides an overview of the programming, interrupt, cache, and MMU models as they are defined by the PowerPC architecture and Power ISA architecture, noting any differences either in how the resources are defined in the different versions of the architecture or in how those definitions are structured.

The Common User Instruction Set Architecture

The original UISA, Book I, as it was defined in the PowerPC architecture, was consistent with the Book E user-level programming model and now comprises most of the Base category. This ensures binary compatibility across the 15-year legacy of applications and across the many families of desktop, embedded, and server processors.

Users can rely on the foundation laid down by the UISA. Book I remains as part of the Power ISA definition, with the few additions and structural adjustments described in "Book I Changes and Extensions" on page 9. This new Book I fosters the development of further extensions as SoC-specific features, with the incorporation of categories such as SPE and Vector.

The UISA's integer and floating-point register files—32 GPRs and 32 FPRs—have provided a model for expanding the ISA for AltiVec's 128-bit vector registers (VRs) and the SPE's 64-bit, two-element GPRs.

Likewise, the original PowerPC architecture defined special-purpose registers (SPRs) and the Move to Special-Purpose Register (**mtspr**) and Move from Special-Purpose Register (**mfspr**) to access them. Book E defined device control registers (DCRs) accessed by **mtdcr** and **mfdcr** instructions. The EIS defined the performance monitor register file (PMRs) and **mtpmr** and **mfpmr** instructions, now part of the Embedded.Performance Monitor category.

All Power Architecture instructions have the following characteristics:

- Data organization in memory and data transfers—Bytes in memory are numbered consecutively starting with 0. Each number is the address of the corresponding byte.

  Memory operands can be bytes, half words, words, double words, or, for the load/store multiple instruction type and load/store string instructions, a sequence of bytes or words. The address of a memory operand is the address of its first byte (that is, of its lowest numbered byte). Operand length is implicit for each instruction.
- Alignment and misaligned accesses—The operand of a single-register memory-access instruction has an alignment boundary equal to its length. An operand's address is misaligned if it is not a multiple of its width.

Some instructions require their memory operands to have certain alignment. Also, alignment can affect performance. For single-register memory access instructions, the best performance is obtained when memory operands are aligned.

The VLE category, described in "Book VLE Category" on page 17 introduces 16-bit encodings of some UISA-defined instructions; these instructions are defined in a separate book, Book VLE.

An Overview of Categories

This section provides an overview of the categories as they are defined by the Power ISA.

Note that some categories are defined as dependent; that is, they can be implemented only if the category they are dependent on is also implemented. Dependent categories are identified by a dot (.) in their category name. For example, a processor cannot implement the Floating-Point.Record category (FP.R) without the Floating-Point category (FP).

An implementation that supports a facility or instruction in a given category supports all facilities and instructions in that category.

All devices implement the facilities defined by the Base category. This is the largest category, encompassing all of the components common across the computing environments; for example, these include the integer computational and load store instructions, and the GPRs that are essential to the user-level programming model for all devices. Although the Base category largely consists of the features defined in Book I (the user ISA), like many categories, it extends beyond Book I to include those Book II and Book III features common to all Power Architecture devices, such as the machine state register (MSR), the time base, the interrupt model's save and restore registers, and the instructions required for accessing them.

The next largest categories are those that support the two computing environments to which the Power ISA is written, the server and embedded environments. The following section gives a high-level description of the Embedded category; the remaining categories are defined in the sections that follow.

The Embedded Category

As described above, the Embedded category largely consists of features formerly defined by the PowerPC Book E architecture and the Freescale EIS. This section describes the components as they are defined in the Power ISA definition. Note that the high-level Embedded category passes on resources defined as part of Book E, including the following:

- Write MSR External Enable instructions (**wrtee**[**i**]), which can be used to update only MSR[EE].
- The software-use SPRs (SPRG4–SPRG7).
- Device control registers (DCRs), which are used in e200 cores.

Other categories are dependent categories of the Embedded category. These include the following:

- Embedded.Cache Locking—Category E.CL. Originally defined by the EIS and implemented in the e500 and e200 cores, cache locking allows instructions and data to be locked into their respective caches on a cache line basis. Locking is performed by a set of touch and lock set instructions.

- Embedded.Enhanced Debug—Category E.ED. The enhanced debug definition is drawn from the EIS and is implemented in the e200 and e500 cores. It defines a separate set of save and restore resources—DSRR0 and DSRR1 and the Return from Debug Interrupt instruction (**rfdi**).
- Embedded.MMU Type FSL—Category E.MF. The embedded MMU consists primarily of the storage architecture defined by Book E and the Freescale EIS. It includes the following SPRs, all of which are supervisor registers:
  - — MMU assist registers MAS0–MAS4 and MAS6–MAS7.
  - — Process identification registers PID1 and PID2. PID0 is defined (as PID) in Book III-E.
  - — The TLB configuration registers, TLB0CFG–TLB3CFG
  - — The MMU control and status register, MMUCSR0
  - — The MMU configuration register MMUCFG
- Embedded.Performance Monitor—Category E.PM. The performance monitor facility is used for characterizing behavior within the microarchitecture of the core and is especially useful in system bring-up and debugging and for optimizing task-scheduling and data-distribution algorithms.

  The events that are monitored are specific to each device. They typically characterize traffic within the instruction pipeline (counts of instructions of different types that are fetched, decoded, or finished; number of cycles of inactivity due to stalls at various points in the pipeline) and operations at the cache and memory interface (hits, reloads, and retries).

  The performance monitor had traditionally been a standard implementation-specific feature on Freescale PowerPC devices using SPRs to configure the facility and to hold the event counters. The performance monitor became a formal part of the EIS with the introduction of Book E, at which point SPRs were replaced with performance monitor registers (PMRs), which function analogously to SPRs. The PMRs are accessed explicitly with the Move to PMR and Move From PMR instructions.

  Note that the Power ISA technology defines an alternate performance monitor model for Server category devices. Note also that many devices that integrate a PowerPC or Power ISA core also implement an additional performance monitor that can be used to characterize and optimize SoC-level behavior.

  For more information, see the "Performance Monitor Registers (PMRs)" on page 37.
- Embedded.Processor Control—Category E.PC. Provides a way for processors within a coherency domain to send messages to all devices in that domain. It also provides a way to send interrupts that are not dependent on the interrupt controller to processors, and it allows message filtering by the processors that receive the message. It can be used to send messages to a device that provides specialized services such as secure boot operations controlled by a security device.
- Embedded.Process ID—Category E.PD. Provides capabilities for loading and storing GPRs and performing cache management operations using a supplied context other than the context normally used by translation.

The subcategories of the Embedded category cannot be implemented by Server category devices. However, there are other former EIS-defined APUs that are not categories and that are not restricted to either the Server or Embedded categories, for example, the **isel** instruction.

AltiVec Technology (Category.Vector)

The AltiVec technology provided a prototype for the APUs that were developed in conjunction with Book E, in that it added to the programming and interrupt models. It is primarily an extension of Book I, but it identifies some resources for interrupt handling in Book III-E.

The Vector category not only makes use of architecture-defined resources; it creates a 32-entry vector register (VR) file, similar to the GPRs and FPRs, but widened to 128 bits to accommodate multiple-element vector operands. The AltiVec extension was developed to provide the following:

- The ability to equip a single high-performance RISC microprocessor with DSP-like computing power for controller and signal-processing functions
- Highly parallel computational operations, allowing simultaneous execution of up to 16 operations per clock cycle
- Wide data paths and wide field operations that can accelerate a broad array of traditional computing and embedded processing operations
- A programmable solution that can easily migrate by using software upgrades to respond to changing standards and customer requirements
- An integrated solution, 100% compatible with the PowerPC architecture

Although AltiVec technology initially targeted high-intensity graphics and scientific calculations, the ability to perform mathematical computation, logical operations, and bit manipulation simultaneously provided a competitive edge in realms of computing far removed from those envisioned by the AltiVec architects.

AltiVec technology defines the following:

- 162 instructions that are an extension to the PowerPC definition
- Four-operand, non-destructive instructions
  - Up to three source operands and a single destination operand
  - Support for advanced “multiply-add/sum” and permute primitives
- Simplified load/store architecture
  - Simple byte, half-word, word and quad-word loads and stores
  - Virtually no misaligned accesses—software managed via permute instruction

The AltiVec technology introduced an important concept—the value of making architectural extensions that provide a powerful suite of special-purpose functionality critical to certain computing segments. This concept has provided a framework for an architecture that can broaden its diversity to support niche computing without sacrificing consistency across its many environments.

Floating-Point Categories—Floating-Point (FP) and Floating-Point with Record (FP.R)

The Floating-Point categories consist of the instructions, registers, and interrupt resources originally defined by the PowerPC architecture to support single- and double-precision floating-point instructions.

The definition of these resources has not changed. Defining these resources as a separate category underscores the advantages of providing a modular architecture, providing greater leeway in balancing power, thermal, size, and price constraints for very specific environments.

Move Assist (Category.MA)

The move assist instructions (load and store string instructions **lswi**, **lswx**, **stswi**, and **stswx**) are defined as part of the integer instruction set in the UISA. These instructions have typically not been supported on recent Freescale devices.

Signal Processing Engine (Category.SPE)

The SPE is a 64-bit, two-element, single-instruction multiple-data (SIMD) ISA, originally designed to accelerate signal processing applications normally suited to digital signal processing (DSP) operations. The two-element vectors fit within GPRs extended to 64 bits. SPE also defines an accumulator register (ACC) to allow for back-to-back operations without loop unrolling. Like the Vector category, SPE is primarily an extension of Book I but identifies some resources for interrupt handling in Book III-E.

In addition to add- and subtract-accumulate operations, the SPE supports a number of multiply-accumulate operations, including negative-accumulate forms; as summarized in Table 1. The SPE supports signed, unsigned, and fractional forms. For these instructions, the fractional form does not apply to unsigned forms, because integer and fractional forms are identical for unsigned operands.

Mnemonics for SPE instructions generally begin with the letters ‘ev’ (embedded vector).

Table 1. SPE Vector Multiply Instruction Mnemonic Structure

Prefix	Multiply Element		Data Type Element		Accumulate Element	
evm	**ho**	half odd (16x16->32)	**usi**	unsigned saturate integer	**a**	write to ACC
	he	half even (16x16->32)	**umi**	unsigned modulo integer	**aa**	write to ACC & added ACC
	hog	half odd guarded (16x16->32)	**ssi**	signed saturate integer	**an**	write to ACC & negate ACC
	heg	half even guarded (16x16->32)	**ssf**[1]	signed saturate fractional	**aaw**	write to ACC & ACC in words
	wh	word high (32x32->32)	**smi**	signed modulo integer	**anw**	write to ACC & negate ACC in words
	wl	word low (32x32->32)	**smf**[1]	signed modulo fractional		
	whg	word high guarded (32x32->32)				
	wlg	word low guarded (32x32->32)				
	w	word (32x32->64)				

[1] Low word versions of signed saturate and signed modulo fractional instructions are not supported. Attempting to execute an opcode corresponding to these instructions causes boundedly undefined results.

Embedded Vector and Scalar Single-Precision Floating-Point Categories

The embedded floating-point categories are dependent categories of the Signal Processing Engine category. These include the following:

- Single-precision scalar (SP.FS)
- Single-precision vector (SP.FV)
- Double-precision scalar (SP.FD)

The embedded floating-point categories provide IEEE-compatible floating-point operations to power- and space-sensitive embedded applications. As is true for all Signal Processing Engine categories, rather than implementing the FPRs defined by the PowerPC architecture, these categories share the GPRs used for integer operations, extending them to 64 bits to accommodate vector single-precision and scalar double-precision categories. These extended GPRs are described in "Register Files" on page 28.

Book VLE Category

There is perhaps no clearer evidence of the breadth and adaptability of the Power Architecture model than the variable length encoding (VLE) category. VLE redefines encodings for many UISA-based instructions to fit into 16-bit opcodes, which allows the UISA to be presented into environments where there is a driving need for a small code footprint. Like the Vector and Signal Processing Engine categories, the VLE category extends Book I–, II–, and III–level resources, although it is defined separately as Book VLE.

The option of using 16-bit encodings offers more efficient binary representations of applications for the embedded processor spaces where code density plays a major role in overall system cost. This alternate encoding can also improve performance. The purpose of the VLE extension is neither to define an entirely different ISA nor to supplant the PowerPC ISA; instead, the VLE extension is a supplement that can improve code density to an application or to part of an application.

The VLE set of alternate encodings is selected on an instruction-page basis. A single page-attribute bit selects between standard instruction encodings and VLE instructions for that page of memory. Pages of either configuration can be intermixed freely, allowing a mixture of both types of encodings in an application.

Instruction encodings in instruction pages marked as using the VLE extension are either 16 or 32 bits long and are aligned on 16-bit boundaries. Therefore, all pages marked as VLE must use big-endian byte ordering.

The programming model uses the same register set with both instruction encodings, although certain registers are not accessible by VLE instructions using the 16-bit formats, and not all condition register (CR) fields are used by condition setting or conditional branch instructions executing from a VLE instruction page. Furthermore, immediate fields and displacements differ in size and use, due to more restrictive encodings imposed by VLE instructions.

Other than the requirement of big-endian byte ordering for instruction pages and the additional page attribute to identify whether the instruction page corresponds to a VLE section of code, VLE complies with the Embedded category memory model. Likewise, the VLE extension complies with the Book III–E definitions of the exception and interrupt models, timer facilities, debug facilities, and SPRs.

Wait (Category WT)

The Wait category, defined in Book II, provides an ordering function for the effects of all instructions executed by the processor executing the **wait** instruction. The **wait** instruction ensures that all previous instructions complete before it does and that no subsequent instructions are initiated until an interrupt occurs. Any prefetched instructions are discarded and instruction fetching is suspended until an interrupt occurs.

Instruction Model

This section describes the instructions and instruction classes as they are defined as part of the Power ISA 2.04 definition. Features that are defined only for the PowerPC architecture are indicated as such. These instructions are grouped by function, as follows:

- Integer instructions—These include arithmetic, logical, and integer load/store instructions. See "Integer Instructions" on page 19.
- Floating-point instructions—These include the floating-point instructions defined by the PowerPC architecture and the floating-point vector and scalar arithmetic instructions defined as part of the Signal Processing Engine category. See "Floating-Point Instructions (Category FP, FP.R)" on page 21.
- Branch and flow control instructions—These include branching instructions, CR logical instructions, trap instructions, and other instructions that affect instruction flow. See "Branch and Flow Control Instructions (Base Category)" on page 22.
- Processor control instructions—These instructions are used for accessing architecturally defined registers, such as SPRs, the condition register (CR), and the machine state register (MSR). See "Processor Control Instructions (Base Category)"on page 23.
- Memory synchronization instructions—These ensure that accesses to memory and memory resources occur in correct order with respect to memory operations generated by other instructions or by other memory devices. See "Memory Synchronization Instructions" on page 24.
- Memory control instructions—These instructions provide control of caches and TLBs. See "Memory Control Instructions" on page 24.

Integer instructions operate on word operands and use the GPRs. Floating-point instructions operate on single- and double-precision floating-point operands. Floating-Point category instructions use FPRs, while SP.FP, SP.FD, and SP.FV category instructions use the GPRs. Instructions are 4 bytes (one word) long and must be word-aligned. The architecture provides byte, half-word, word, and double-word operand loads and stores between memory and a set of 32 GPRs and provides for word and double-word operand loads and stores between memory and a set of 32 floating-point registers (FPRs). When data is loaded from memory to an FPR, the architecture requires that both double-precision and single-precision data be internally kept in double-precision format.

Arithmetic and logical instructions do not read or modify memory. To use the contents of a memory location in a computation and then modify the same or another location, the memory contents must be loaded into a register, modified, and then written to the target location using load and store instructions.

Note that the PowerPC architecture allows out-of-order, parallel execution but requires in-order completion. Some operations, especially those that update the processor state, must be performed in an order that guarantees that adjacent instructions complete execution and make results available in the proper context. Such serialization is handled by the instruction pipeline microarchitecture.

Similarly, it is sometimes necessary to insert synchronization instructions into the program flow to guarantee that accesses to memory and memory resources such as TLBs complete in order. These memory synchronization instructions control the order in which memory operations complete with respect to asynchronous events, and the order in which memory operations are seen by other mechanisms that access memory.

Simplified Mnemonics

The description of each instruction in the architecture includes the mnemonic and a formatted list of operands. To simplify assembly language programming, a set of simplified mnemonics and symbols is provided for some of the frequently used instructions such as branch conditional, compare, trap, and rotate and shift instructions. These simplified mnemonics redefine the mnemonics to incorporate numerical information provided in operands. For example, there are simplified mnemonics for the **mtspr** and **mfspr** instructions that, instead of requiring the SPR number as operand, incorporate the name of the SPR into the mnemonic. To load a value from a GPR into the count register, **mtctr r**S can be used instead of **mtspr 9,r**S. The Power ISA definition extends the set of simplified mnemonics to include new SPRs that are being phased in.

Simplified mnemonics for individual processors are listed in each reference manual.

Instruction Set Overview

This section provides a brief overview of the Power ISA defined for Embedded category devices.

Architected instructions occupy specifically defined spaces in the opcode space. Because they are defined for a variety of specific environments, some categories are mutually exclusive, so their opcodes may overlap. For example, the Vector and Signal Processing Engine categories are both SIMD instruction sets that target distinctly different markets and so they have many overlapping opcodes. An implementation that attempts to execute a reserved instruction, or any other instruction that is not implemented, generates an interrupt.

Integer Instructions

This section describes the integer instructions, all of which are defined in Book I. All are part of the Base category except for the load/store string and multiple instructions, which make up the move assist category. and load/store double word instructions defined as part of the 64-bit category.

These integer instructions are grouped as follows:

- Integer arithmetic instructions
- Integer compare instructions
- Integer logical instructions
- Integer rotate and shift instructions
- Integer select instruction (formerly the EIS integer select APU)

Integer instructions use GPRs for source operands and place results into GPRs and the XER and CR fields. Integer instructions are shown in Table 2.

Table 2. Integer Computational Instructions

Instructions	Instruction Name	Options
Integer arithmetic (**add***x*, **div***x*, **mul***x*, **neg***x*, **sub***x*)	Add	Carrying, extended, immediate, shifted, minus one, zero
	Divide	Word, unsigned
	Multiply	High word, low word, unsigned, immediate
	Negate	—
	Subtract	From, carrying, extended, immediate, minus one, zero
Integer compare (**cmp***x*)	Compare	Immediate, logical
Integer logical (**and***x*, **cnt**, **eqv**, **ext***x*, **nand**, **nor***x*, **or***x*, **xor***x*)	AND	Immediate, shifted, with complement
	Count	Leading zeros, word
	Equivalent	—
	Extend	Sign, byte, half word
	NAND	—
	NOR	—
	OR	Immediate, shifted, complement
	XOR	Immediate, shifted
Integer rotate and shift (**rlw***x*, **slw***x*, **srw***x*, **sraw***x*)	Rotate left word	Immediate, then AND with mask, then mask insert
	Shift	Left word, right word, algebraic word, immediate
Integer select (**isel**)	Integer Select	—

Integer load and store instructions, shown in Table 3, are issued and translated in program order; however, the accesses can occur out of order. Synchronizing instructions (see Table 8) are defined in Book II and are provided to enforce strict ordering.

Table 3. Integer Load/Store Instructions

Instruction	Instruction Name	Options/Comments
Integer load (**lb***x*, **lh***x*, **lw***x*, **ld***x*[1])	Load	Byte, word, double word[1], half word, algebraic (half word), byte reverse, and zero, with update, indexed
Integer load multiple/string word: **lmw**, **lswi**	Load multiple word	Base category
	Load string word	Move Assist category
Integer store (**stb***x*, **sth***x*, **stw***x*, **std***x*[1])	Store	Byte, word, half word, double word[1], byte-reverse, with update, indexed
Integer store multiple/string word: **stmw**, **stswi**	Store multiple word	Base category
	Store string word	Move Assist category

[1] Category 64-bit only

Floating-Point Instructions (Category FP, FP.R)

The floating-point model is written to the IEEE® Std. 754™, which defines conventions for single- and double-precision arithmetic. The standard requires that single-precision arithmetic be provided for single-precision operands.

The instructions follow these IEEE-754 guidelines:

- Double-precision arithmetic instructions can have single-precision operands but always produce double-precision results.
- Single-precision arithmetic instructions require all operands to be single-precision and always produce single-precision results.
- Conversion from double- to single-precision must be done explicitly by software; conversion from single- to double-precision is done implicitly by the processor.

The floating-point computational, move, and select instructions operate on data kept in FPRs and, except for the compare instructions whose results are reported to the condition register (CR), place the result value into a target FPR specified as part of the instruction syntax. Instruction forms with Rc=1 place additional status information into the CR and are part of the dependent Floating-Point.Record category.

The signal processing engine (SPE) category defines an alternative floating-point instruction set that uses GPRs rather than FPRs. See "Embedded Vector and Scalar Single-Precision Floating-Point Categories" on page 16.

Table 4 provides an overview of the floating-point computational instructions.

Table 4. Floating-Point Computational Instructions

Instructions	Instruction Name	Options
Floating-point elementary arithmetic (**fadd***x*, **fdiv***x*, **fmul***x*, **fsub***x*, **fsqrt***x*, **fres***x*, **fabs**, **fmr**, **fnabs**, **fneg**)	Add	Single, double
	Divide	Single, double
	Multiply	Single, double
	Reciprocal	Estimate single, square root estimate
	Square root	Single, double
	Subtract	Single, double
	Absolute value	—
	Move register	—
	Negative absolute value	—
	Negate	—
Floating-point multiply-add (**fmadd***x*)	Multiply-add	Single, double
	Multiply-subtract	Single, double
	Negative multiply-add	Single, double
	Negative multiply-subtract	Single, double

Table 4. Floating-Point Computational Instructions (continued)

Instructions	Instruction Name	Options
Floating-point rounding and conversion (**fcti***x*, **fr***x*)	Convert from integer	Double word
	Convert to integer	Word, double word, round to zero
	Round to single-precision	—
Floating-point compare and select (**fcm***x*)	Compare	Ordered, unordered
	Select	—
Floating-point status and control register (**mtf***x*, **mff***x*)	Move from FPSCR	—
	Move to FPSCR	Bit 0, Bit 1, fields, immediate

The floating-point load and store instructions are required to transfer operands between memory and the FPRs.

Table 5. Floating-Point Load and Store Instructions

Instructions	Instruction Name	Options
Floating-point load (**lf***x*)	Load floating-point	Double, single, with update, extended, indexed
Floating-point store (**stf***x*)	Store floating-point	Double, single, with update, extended, indexed, as integer word

Branch and Flow Control Instructions (Base Category)

Branch instructions are used to redirect program flow. Usually, this is done conditionally based on CR bit values. If a previous instruction in progress may affect the particular CR bit, the branch is considered unresolved. The direction of the branch may be predicted either using the static branch prediction that can be encoded as part of the branch syntax, or through some hardware mechanism specific to the device. Implementations can begin executing instructions fetched according to the prediction, but the results of this execution cannot update architected registers or memory unless and until the value of the CR bits determines a prediction is correct, at which point results can be committed. If the prediction is incorrect, the fetched instructions and any of their results are purged, and the instruction fetching continues along the alternate path.

Branch instruction functions include the following:

- Branch instructions redirect instruction execution conditionally based on the value of bits in the CR. For branch conditional instructions, the BO operand specifies the conditions under which the branch is taken.
- CR logical instructions perform logical operations on CR contents that help determine branching conditions.
- Trap instructions test for a specified set of conditions. If any of the tested conditions is met, a system trap type interrupt is taken.
- Executing a System Call (**sc**) instruction lets a user program call on the system to perform a service by invoking a system call interrupt. System Call instructions can be either user- or supervisor-level.

For branch conditional instructions, the BO operand specifies the conditions under which the branch is taken. The BI operand specifies which of the 32 CR bits to test.

Because it can be cumbersome for a programmer to remember the various meanings of BO and BI encodings, the architecture provides simplified mnemonics that allow conditions specified by BO and BI to be incorporated into the mnemonic. For example, the Branch Conditional instruction, whose syntax is **bc** BO**,**BI, *target address*, can be coded to decrement the count register (CTR) and branch as long as the CTR is not zero (closure of a loop controlled by a count loaded into CTR). To specify this condition, the BO field must be coded as 16. Alternatively, a simplified mnemonic is available, **bdnz,** that indicates "branch while the decremented value is non-zero." Using the simplified mnemonic eliminates the BO and BI operands, simplifying '**bc** 16,0,target' to the more easily remembered '**bdnz** *target*', which generates identical machine code.

The supervisor-level **rfi** instruction is used for returning from a standard interrupt handler. The **rfci** instruction is used for critical interrupts and **rfmci** is used for machine-check interrupts (Embedded category), and **rfdi** is used for debug interrupts (Embedded.Enhanced Debug category). See the "Interrupt Model" section on page 40.

Branch and flow control instructions are shown in Table 6.

Table 6. Branch and Flow Control Instructions

Instruction	Name	Options
Branch (**b***x*, **bc***x*)	Branch	Address, and link, conditional, conditional to link register, conditional to count register, if less than, if not less than, if less than or equal to, if equal to, if not greater than, if greater than, if greater than or equal to, if summary overflow, if not summary overflow, if unordered, if not unordered, and LR update
CR logical (**cr***x*, **mcr***x*)	Condition register	AND/AND with complement, OR/OR with complement, XOR, NAND, NOR, Equivalent
Trap (**t***x*, **tw***x*)	Trap	Word, immediate, less than, not less than, equal, less than or equal, not equal, not greater than, greater than, greater than or equal, logically less than, logically not less than, logically less than or equal, logically not greater than, logically greater than, logically greater than or equal
System call (**sc**)	System call	—
Return (**rf***x*)	Return from	Interrupt (Base category), critical and machine-check interrupts (Embedded category), debug (embedded.enhanced debug category

Processor Control Instructions (Base Category)

Processor control instructions are used to read from and write to registers other than GPRs and FPRs that can be accessed specifically. These include CR, XER, MSR, and SPRs. The time base register and some SPRs are accessible at both the user and supervisor levels; separate SPR numbers are used for each.

Note that the Embedded category defines the Write MSR External Enable instructions (**wrtee**[**i**]), which can be used instead of **mtmsr** to update only MSR[EE], which enables or disables external interrupt exception conditions. The **wrtee** instruction has fewer serialization requirements, and therefore shorter latency, than **mtmsr**.

Table 7 summarizes processor control instructions.

Table 7. Processor Control Instructions

Instructions	Name	Options
Move (**mt***x*, **mf***x*)	Move to	SPR, CR fields, CR from XER, DCR, time base, MSR, PMR (Embedded.Performance Monitor category)
	Move from	SPR, DCR, CR, TB, MSR, PMR

Memory Synchronization Instructions

Memory synchronization instructions control the order in which memory operations execute with respect to asynchronous events and the order in which operations are seen by other mechanisms that access memory. Memory synchronization instructions are user-level instructions and are shown in Table 8.

Table 8. Memory Synchronization Instructions

Instructions	Instruction Name	Comments
Load word and reserve index	Load word	**lwarx**
Store word conditional index	Store word	**stwcx.**
Synchronize (**sync**, **eieio**, **isync**, **msync**, **mbar**)	Synchronize (Memory Synchronize)	Book E recast PowerPC architecture–defined **sync** as **msync**. The Power ISA version defines **msync** as a simplified mnemonic, configured to function as the Book E–defined **msync** for Embedded category devices.
	Enforce In-Order Execution of I/O	PowerPC architecture 1.10/Server category
	Memory Barrier	Embedded category. The Server category defines this opcode as **eieio**, as in the PowerPC architecture, 1.10.
	Instruction Synchronize	**isync** synchronizes instruction stream

Memory Control Instructions

Memory control instructions, categorized below, include instructions for cache management and TLB management:

- Cache instructions—Help programs manage on-chip caches if they are implemented. The effects of the cache management instructions on memory are weakly ordered. If the programmer needs to ensure that cache or other instructions have been performed with respect to all other processors and system mechanisms, a **sync** or **msync** (a simplified mnemonic in the Power ISA definition) must be placed in the program following those instructions.
- Segment register instructions—Defined by the PowerPC architecture 1.10 and not part of the embedded environment.
- TLB management instructions—Among the resources that the Embedded category defines to support software address translation are the **tlbwe** and **tlbre** instructions, which are used to directly configure the TLBs with translation and memory protection information. Additional instructions are provided for searching and invalidating entries and for synchronizing TLB accesses.

For performance reasons, many processors implement one or more TLBs on-chip. These are caches of portions of the page table. As changes are made to the address translation tables, it is necessary to maintain coherency between the TLB and the updated tables. This is done by invalidating TLB entries, or occasionally by invalidating the entire TLB and allowing the translation caching mechanism to refetch from the tables.

Memory control instructions are listed in Table 9.

Table 9. Memory Control Instructions

Instructions	Name	Options
User-level cache (**dcb***x*, **icb***x*)	Data cache block	Touch, touch for store, allocate, clear, zero, store, flush (Embedded category)
	Instruction cache block	Invalidate, touch (Embedded category)
Supervisor-level cache (**dcbi**)	Data cache block	Invalidate (Embedded category)
Supervisor-level cache (**dcb***x*)	Data cache block	Invalidate
dcbtls, **dcbtstls**, **icbtls** **dcblc**, **icblc**	Data/instruction cache block touch (for store) and lock set	Embedded.Cache Locking category
dcb*x***ls**, **icb***x***ls**	Data/instruction cache block lock touch and set	
TLB management (**tlb***x*)	TLB invalidate	Entry, all, virtual address indexed (Note that Embedded category versions of these instructions differ from the Server category versions).
	TLB synchronize	—
	TLB read entry	Embedded category
	TLB search indexed	Embedded category
	TLB write entry	Embedded category

Register Model

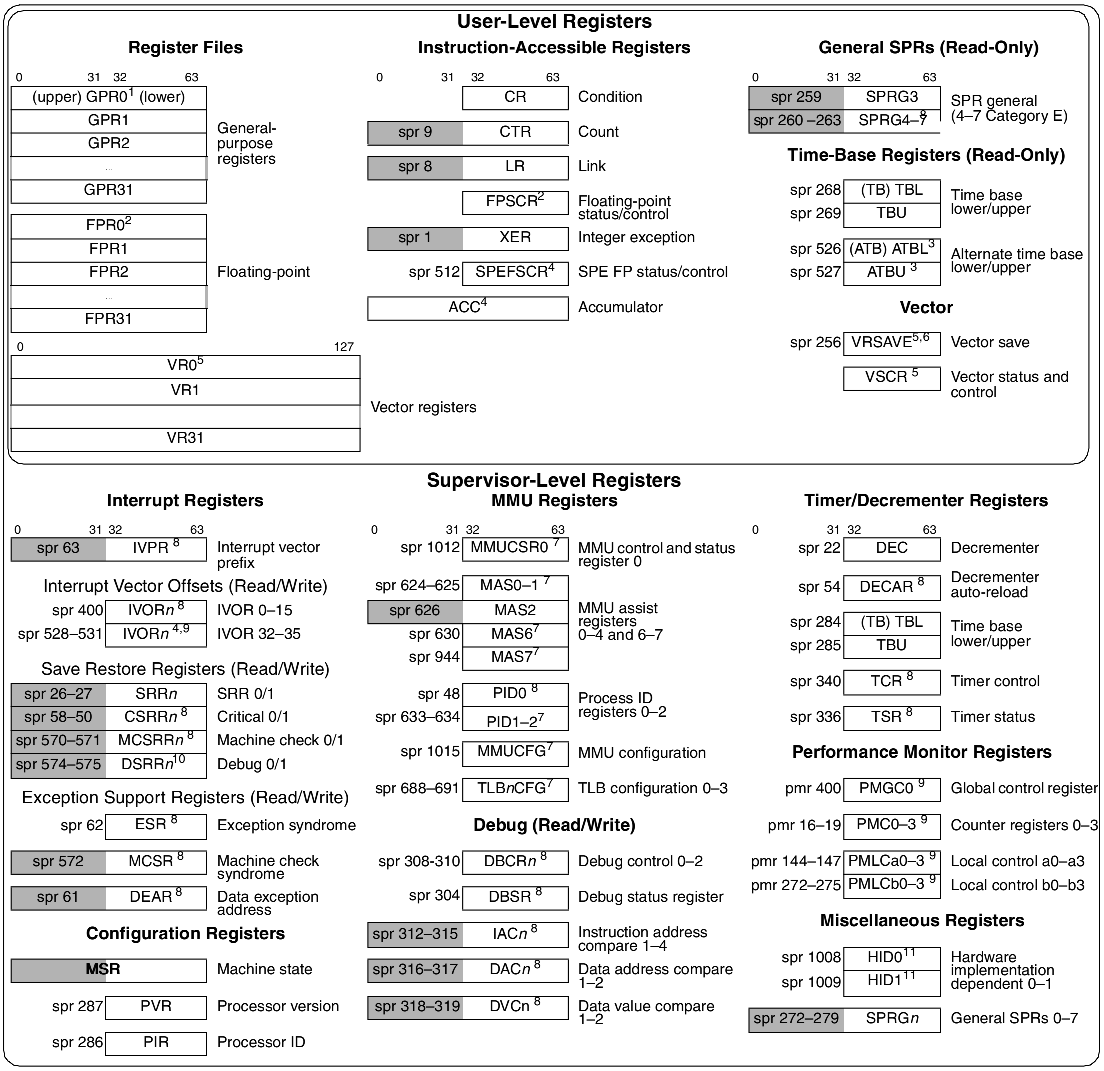
The Power Architecture model defines register-to-register operations for all computational instructions. Source data for these instructions is accessed from the on-chip registers or is provided as immediate values embedded in the opcode. The Power Architecture model allows specification of a target register distinct from the two source registers, preserving the original data for use by other instructions and reducing the number of instructions for some operations. Data is transferred between memory and registers with explicit load and store instructions only. Registers hold the source or destination of an instruction, or they are accessed as a by-product of execution.

Most registers defined in the PowerPC architecture 1.10 are unchanged in the Power Architecture 2.04 model. A few have been replaced by other registers, and in some cases new fields are defined, primarily to support functionality defined by categories that have been added to the architecture.

Registers include the following:

- Register files—General-purpose registers (GPRs) and floating-point registers (FPRs) are accessed as either the source or destination of an instruction. Likewise, the Vector category uses vector registers (VRs), and the SPE uses the 32-bit GPRs extended to 64-bits. GPRs are often used to generate the effective address for instructions that access memory (because GPRs are used to hold addresses, 64-bit implementations require 64-bit GPRs).
- Instruction-accessible registers—Registers such as the condition register (CR), the floating-point status and control register (FPSCR), and some SPRs are accessed as by-products of executing certain instructions.
- Special-purpose registers (SPRs)—On-chip registers that are part of the processor core. They control the use of the debug facilities, timers, interrupts, memory management unit, and other processor resources. They include the hardware implementation-dependent registers (HIDs), not defined by the architecture, that are used for configuration and control. SPRs are accessed with the Move to SPR (**mtspr**) and Move from SPR (**mfspr**) instructions.
- Performance monitor registers, or PMRs (Embedded.Performance Monitor category), and device control registers, or DCRs (Embedded category), offer large sets of on-chip registers similar to SPRs.

Figure 4 shows the Power ISA 2.04 register model for the embedded environment.



▨ Ranges marked in gray are supported only by 64-bit processors.

[1] On 32-bit implementations, only category SPE instructions can access the upper word of the 64-bit GPRs.
[2] Floating-point category (FP)
[3] Alternate time-base category (ATB)
[4] Signal Processing Engine category
[5] Vector category (VEC), formerly AltiVec technology
[6] Formerly USPRG0
[7] Embedded.Freescale MMU Type FSL (E.MF)
[8] Embedded category (E)
[9] Embedded.Performance Monitor category (E.PM)
[10] Embedded.Enhanced Debug category
[11] Defined by the EIS.

Figure 4. Power ISA 2.04 Register Model (Embedded Environment)

Power Architecture user instruction and register models are fully binary-compatible with those of the PowerPC architecture 1.10 UISA.

The UISA registers, shown in Figure 4, can be accessed by user- or supervisor-level instructions; the VEA introduces the time base facility as user-level registers, also shown in Figure 4. The OEA defines the registers that an operating system uses for memory management, configuration, and interrupt handling. The OEA register model includes only supervisor-level registers. The following describes specific registers for both PowerPC architecture 1.10 and the Power Architecture 2.04 register model.

Register Files

Figure 5 shows a comparison of PowerPC architecture 1.10 and Power Architecture register files.

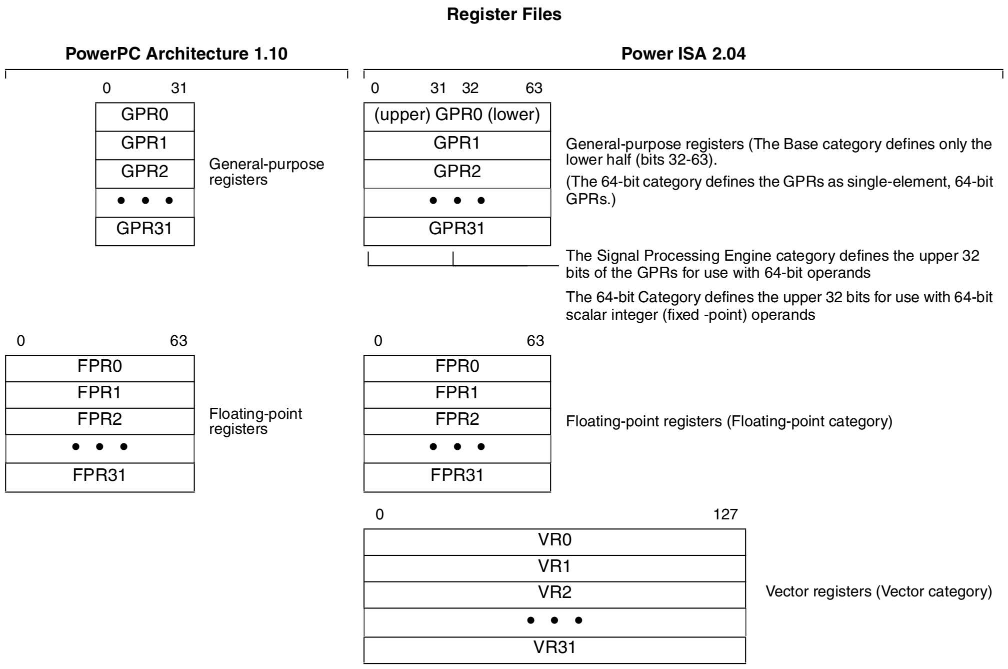

Register Files
PowerPC Architecture 1.10
Power ISA 2.04
0
31
GPR0
GPR1
GPR2
• • •
GPR31
General-purpose registers
0
31
32
63
(upper) GPR0 (lower)
GPR1
GPR2
• • •
GPR31
General-purpose registers (The Base category defines only the lower half (bits 32-63).
(The 64-bit category defines the GPRs as single-element, 64-bit GPRs.)
The Signal Processing Engine category defines the upper 32 bits of the GPRs for use with 64-bit operands
The 64-bit Category defines the upper 32 bits for use with 64-bit scalar integer (fixed -point) operands
0
63
FPR0
FPR1
FPR2
• • •
FPR31
Floating-point registers
Floating-point registers (Floating-point category)
0
127
VR0
VR1
VR2
• • •
VR31
Vector registers (Vector category)

Figure 5. Register File Comparison

PowerPC architecture 1.10 and the Power Architecture 2.04 model both include GPR and FPR files necessary for instruction computation:

- General-purpose registers (GPRs)—GPRs serve as the data source or destination for all integer and non-floating-point load/store instructions and provide data for generating addresses. The PowerPC architecture 1.10 and Book E define a GPR file that consists of thirty-two GPRs designated as GPR0–GPR31. Instructions defined by the Base category for 32-bit implementations use GPRs that are 32 bits wide.

  The Signal Processing Engine category defines a set of thirty-two 64-bit GPRs for use with 64-bit instructions; scalar double-precision embedded floating-point instructions treat the 64 bits as a single operand; SPE vector instructions break the 64-bit registers into two 32-bit elements, which for some instructions are broken into half-word elements.

The 64-bit category, drawn from the PowerPC architecture 1.10, defines 64-bit addressing modes and instructions for loading and storing double-word operands. Because many registers have to be wide enough to hold an address, the sizes of some register resources, including the GPRs, save restore register 0 (SRR0), and the data address register (DAR), are defined to be 32 bits wide in 32-bit implementations and 64 bits wide in 64-bit implementations.

- Floating-point registers (FPRs)—The floating-point category, drawn from the PowerPC architecture 1.10, defines an FPR file that consists of thirty-two 64-bit FPRs, FPR0–FPR31. The FPRs use double-precision operand format for both single- and double-precision data. See "Floating-Point Instructions (Category FP, FP.R)" on page 21.
- Vector registers (VRs)—VRs act as either the source or destination of vector (AltiVec) instructions. The Power Architecture 2.04 model defines a VR file that consists of thirty-two 128-bit VRs, which typically are configured to hold four 32-bit operands to support SIMD operations.

Instruction-Accessible Registers

Figure 6 shows a comparison of PowerPC architecture 1.10 and Power Architecture instruction-accessible registers.

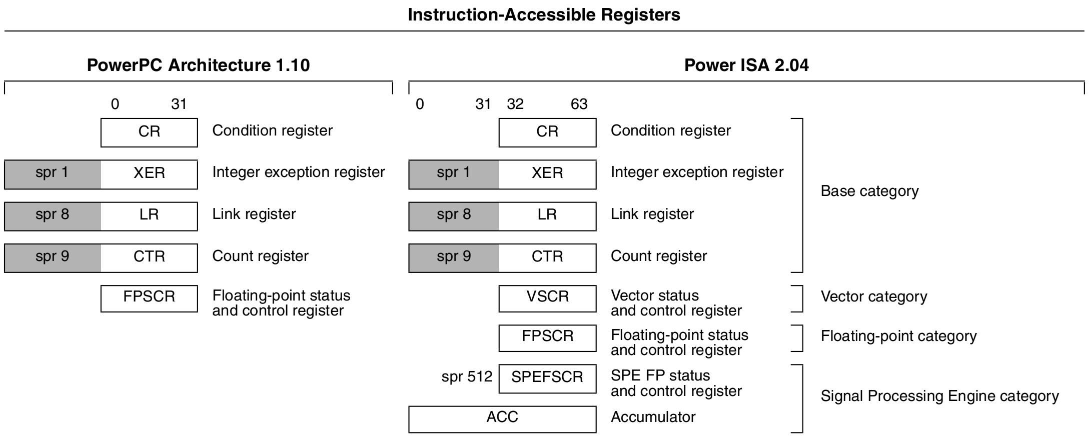

Instruction-Accessible Registers
PowerPC Architecture 1.10
Power ISA 2.04
0
31
32
63
CR
Condition register
spr 1
XER
Integer exception register
spr 8
LR
Link register
spr 9
CTR
Count register
FPSCR
Floating-point status and control register
VSCR
Vector status and control register
spr 512
SPEFSCR
SPE FP status and control register
ACC
Accumulator
Base category
Vector category
Floating-point category
Signal Processing Engine category

Figure 6. Instruction-Accessible Registers Comparison

PowerPC architecture 1.10 and the Power Architecture 2.04 registers contain instruction-accessible registers that can be accessed as the by-product of executing certain instructions:

- Condition register (CR). Reflects the results of testing and branching. It is used to record conditions such as overflows and carries. A specified CR field can be set as the result of either an integer or a floating-point compare instruction.
- Integer exception register (XER). Indicates overflow and carries for integer operations. XER status bits overflow, summary overflow, and carry are set based on the operation of an instruction considered as a whole, not on intermediate results. For example, the subtract from carrying

instruction, which produces a result specified as the sum of three values, sets XER bits based on the entire operation, not on an intermediate sum.

- Link register (LR). Provides the branch target address for the branch conditional to link register (**bclr***x*) instructions and can be used to hold the logical address (also called the effective address) of the instruction that follows a branch and link instruction, typically used for linking to subroutines.
- Count register (CTR). Can be used to hold a loop count, which can be decremented during execution of branch instructions. The entire count register can also be used to provide the branch target address for the branch conditional to count register (**bcctr***x*) instruction.
- Floating-point status and control register (FPSCR). Controls the handling of floating-point exceptions and records status resulting from the floating-point operations. The register includes status bits and control bits needed for compliance with the IEEE 754 floating-point standard.

The following registers support SPE and embedded floating-point instructions:

- SPE floating-point status and control register (SPEFSCR). Used for status and control of SPE and embedded floating-point instructions. It controls the handling of floating-point exceptions and records status resulting from the floating-point operations.
- Accumulator register (ACC). Holds the results of the multiply accumulate (MAC) forms of SPE integer instructions. The ACC allows back-to-back execution of dependent MAC instructions, something that is found in the inner loops of DSP code such as finite impulse response (FIR) filters. The accumulator is partially visible to the programmer in that its results do not have to be explicitly read to use them. Instead, they are always copied into a 64-bit destination GPR specified as part of the instruction. Based upon the type of instruction, this register can hold either a single 64-bit value or a vector of two 32-bit elements.

Time Base Registers

Figure 7 shows a comparison of PowerPC architecture 1.10 and Book E PowerPC architecture time base registers.

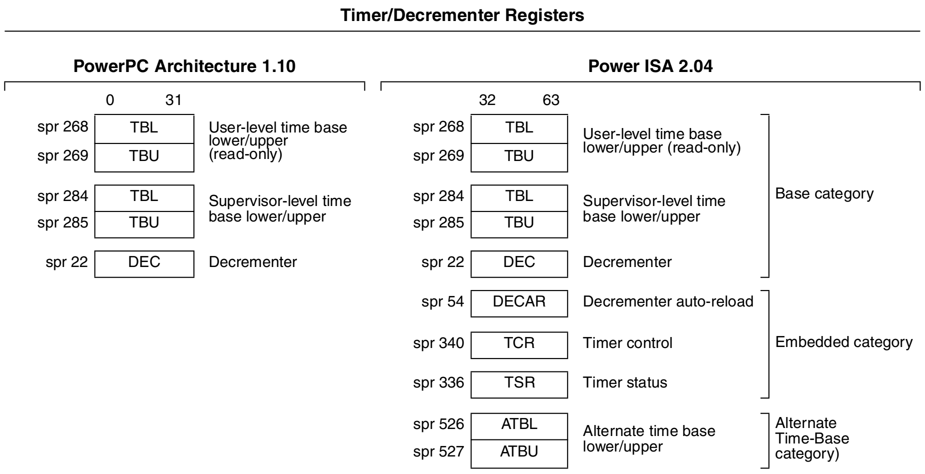


Figure 7. Time/Decrementer Registers Comparison

The following hardware and software timer registers are derived from PowerPC architecture 1.10 and Book E:

- Time base (TBU and TBL). Provides timing functions for the system. TB is composed of two 32-bit registers, the time base upper (TBU) concatenated on the right with the time base lower (TBL). The two 32-bit TB registers count at an implementation-specific rate like a 64-bit counter. User-level applications have read-only access to the TB while supervisor-level applications have read/write access. The time base count is used, among other functions, to trigger interrupts.
- Decrementer register (DEC). Typically used as a general-purpose software timer. It is updated at the same rate as the TB and provides a way to signal a decrementer interrupt after a specified period.

The Book E definition provides registers that incorporate timing mechanisms for the fixed-interval and watchdog timer interrupts defined in Book E:

- Decrementer auto-reload register (DECAR). Used to automatically reload a programmed value into DEC. In the PowerPC architecture 1.10, a value has to be explicitly programmed into DEC.
- Timer control register (TCR). Provides control information for the decrementer. It controls features such as auto-reload enable and decrementer interrupt enable.
- Timer status register (TSR). Contains status on timer events and the most recent watchdog timer-initiated processor reset. It controls features such as watchdog timer, fixed-interval interrupt enable, and watchdog timer interrupt status.

MMU Control and Status Registers

Because the PowerPC architecture MMU specification was cumbersome for embedded applications, many embedded processors were designed with alternate features, such as variable-sized pages and software-managed page tables. These features are now part of the Power ISA 2.04 definition.

The complexity of the modal 32-/64-bit MMU model in the PowerPC architecture 1.10 was replaced in Book E by additional supervisor mode registers and instructions in the UISA. The Book E definition resulted in a more embedded-friendly MMU architecture that is simpler and more flexible while implementing software-driven TLBs and per-page properties. Translation lookaside buffers (TLBs) keep recently-used page address translations on-chip. See "Memory Management Unit (MMU) Model" on page 44 for more information on the MMU.

Figure 8 compares the MMU registers defined by the PowerPC architecture 1.10 with those defined by the Power ISA category Embedded.MMU Type FSL version and the Freescale EIS to support embedded devices.

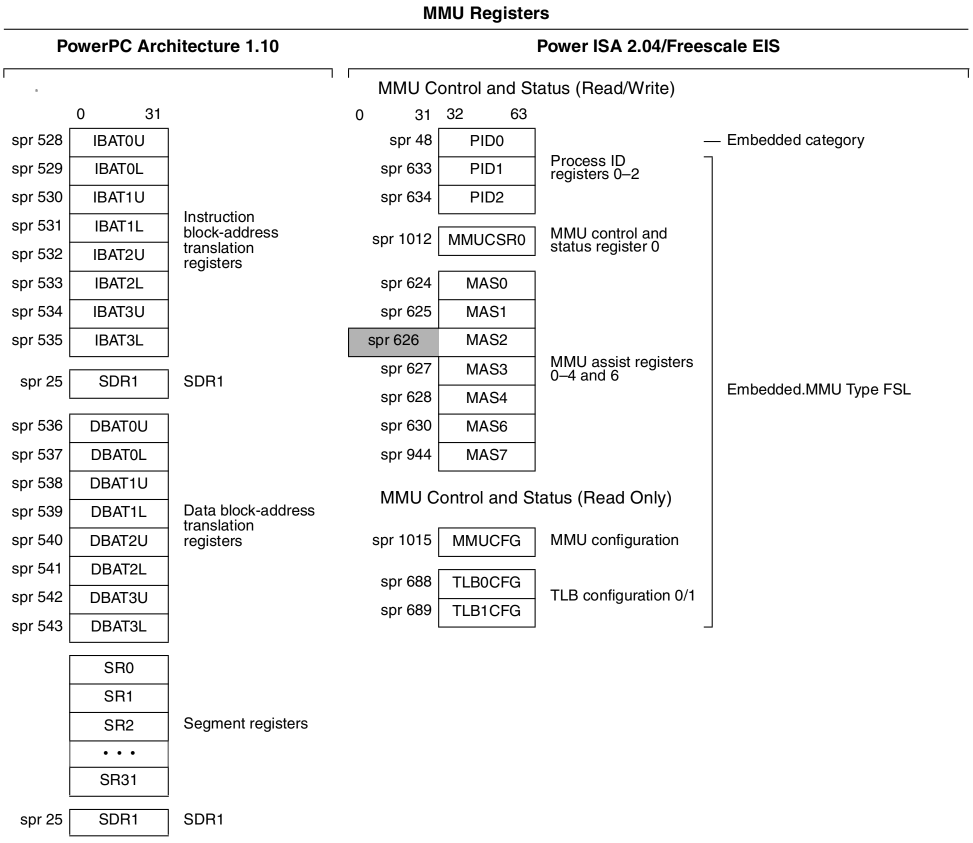


Figure 8. MMU Registers Comparison

The PowerPC architecture definition includes SPRs that are used for address translation:

- Block address translation registers. The PowerPC architecture 1.10 MMU defines BAT registers (BATs) to maintain address translation information for blocks of memory. For each block, a pair of registers is defined, upper and lower, which contain the base address and size of the block, as well as the value of the WIMG bits used to describe cache coherency attributes.

  The architecture defines these BATs as two four-entry fully associative arrays: one array of four pairs for instruction memory (IBATs) and one array of four pairs for data memory (DBATs). Effective addresses are compared simultaneously with all four pairs of registers in the BAT array during block translation.

  The BATs are maintained by the system software and are implemented as eight pairs of SPRs. Each block is defined by a pair of BATs. For example, IBAT0U and IBAT0L provide translation and protection for one block. BAT registers are not part of the PowerPC Book E specification. A more detailed discussion of how BATs function can be found in the section, "Memory Management Unit (MMU) Model" on page 44.
- SDR1 register. SDR1 is a PowerPC 1.10 register that specifies the base address and the size of the page tables in memory. When a table search operation commences, a primary hashing function is performed on the virtual address. The output of the hashing function is then concatenated with bits programmed into SDR1 by the operating system to create the physical address of the primary page table entry group.

The Embedded category implements a register to identify TLB entries used in address translation:

- Process ID register (PID). Provides an identifier value associated with each effective address (instruction or data) generated by the processor. The MMU Type FSL category defines additional PIDs.

The MMU Type FSL category includes MMU assist (MAS) registers, among others, to provide MMU control:

- Process ID registers (PID1–PID2). Provide identifier values associated with each effective address (instruction or data) generated by the processor.
- MMU control and status register 0 (MMUCSR0). Used for general MMU control, for example, to invalidate TLBs.
- MMU assist registers (MAS0–MAS7). Used to configure and manage pages through translation lookaside buffers (TLBs). These registers are used to configure and control MMU read/write and replacement, descriptor configuration, effective page number and page attributes, real page number and access, and hardware replacement assist configuration.
- MMU configuration register (MMUCFG). Provides configuration information for the particular MMU supplied with a version of the core. It is a read-only register that provides information on PID register size and the number of TLBs.
- TLB configuration registers (TLB0CFG–TLB1CFG). These read-only registers provide information about each TLB that is visible to the programming model. They provide configuration information for TLBs and describe aspects such as the associativity, minimum and maximum page sizes of the TLBs, and the number of entries in the TLBs.

L1 Cache Registers (EIS)

The Freescale EIS defines L1 cache configuration and status registers, shown in Figure 9. Neither the PowerPC architecture 1.10 nor the Power Architecture 2.04 specification defines L1 cache registers.

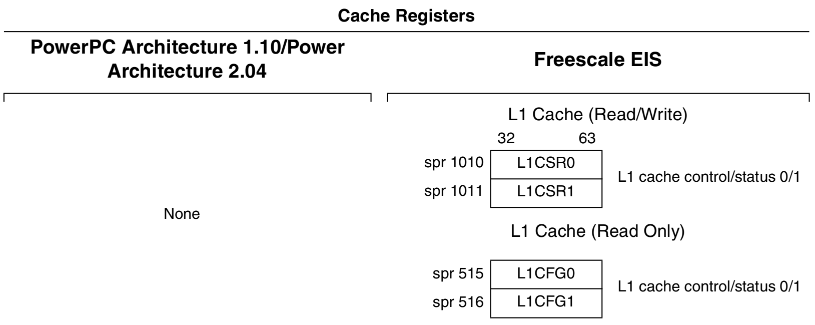


Figure 9. Cache Registers Comparison

The registers in Figure 9 are described as follows:

- L1 cache configuration registers (L1CFG0–L1CFG1). Read-only registers that provide configuration information for the particular L1 data and instruction caches supplied with a version of the core. They include a description of the cache block size, the number of ways, the cache size, and the cache replacement policy, among other features.
- L1 cache control and status registers (L1CSR0–L1CSR1). L1CSRs are used for general control and status of the L1 data and instruction caches and are read/write accessible by supervisor-level programs. They allow the programmer to enable features such as cache parity and the cache itself. They provide status on information such as cache locking and cache locking overflow.

Interrupt Registers

When interrupts occur, information about the state of the processor is saved to certain registers and the processor begins execution at an address (interrupt vector) predetermined for each interrupt. In the PowerPC architecture 1.10 architecture, this interrupt vector consists of a fixed offset prepended with a value as determined by MSR[IP]. Processing of interrupts begins in supervisor mode.

Figure 10 compares the PowerPC architecture 1.10 with the Embedded category interrupt registers.

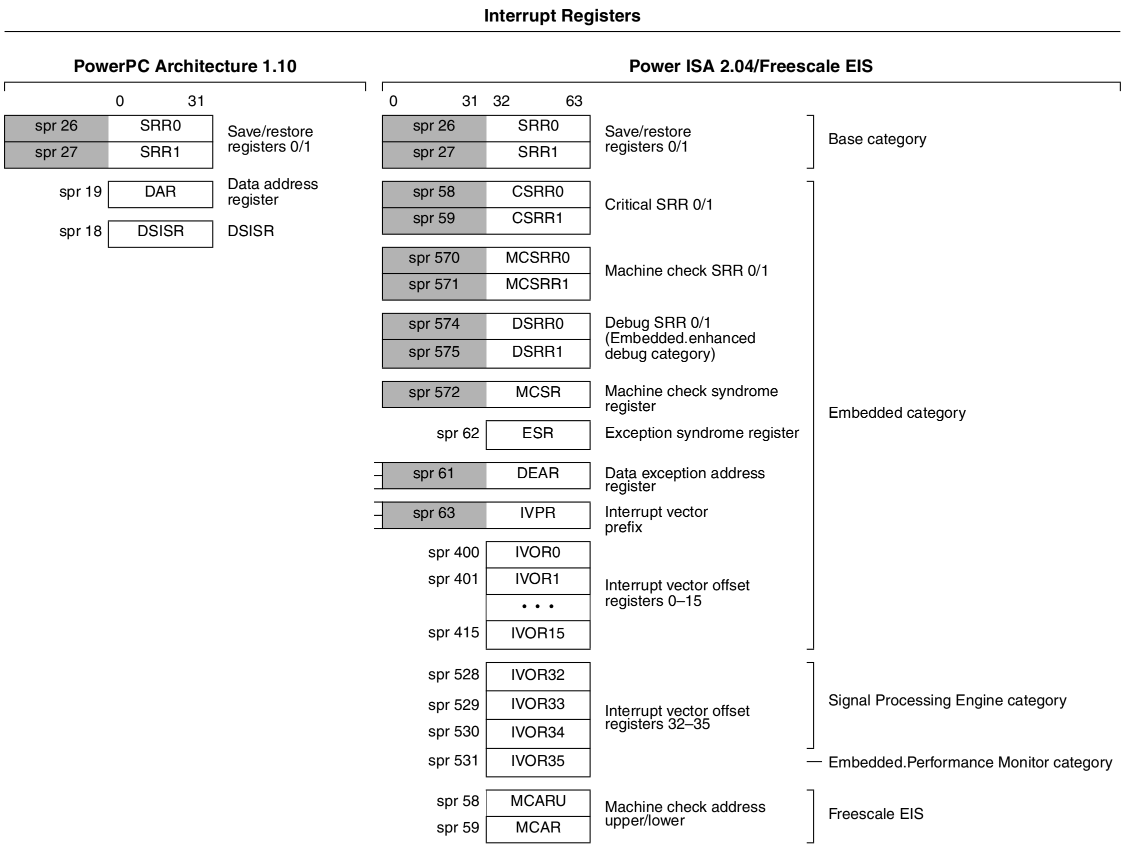


Figure 10. Interrupt Register Comparison

Save/restore registers are automatically updated with machine state information and the return address when an interrupt is taken. The PowerPC architecture 1.10 architecture defines only SRR0 and SRR1. The Embedded category includes Book E–defined critical interrupts and additional interrupt types defined by the EIS that use similar resources. These registers are described below.

- Save/restore registers (SRR0 and SRR1)
  — SRR0 holds the address of the instruction where an interrupted process should resume. For instruction-caused interrupts, it is typically the address of the instruction that caused the interrupt. When **rfi** executes, instruction execution continues at the address in SRR0. In Embedded category devices, SRR0 is used for non-critical interrupts.
  — SRR1 holds machine state information. When an interrupt is taken, MSR contents are placed in SRR1. When **rfi** executes, SRR1 contents are placed into MSR. In Embedded category devices, SRR1 is used for non-critical interrupts.

The PowerPC architecture accounts for DSI (data storage interrupt) and alignment exceptions in its register model:

- Data address register (DAR). The effective address generated by a memory access instruction is placed in the DAR if the access causes an exception (for example, an alignment exception). This register is not supported in Embedded category devices.
- DSISR. The DSISR identifies the cause of DSI and alignment exceptions. It is not supported in Embedded category implementations, although much of its functionality is supported in the ESR.

The Embedded category provides greater flexibility in specifying vectors through the implementation of the interrupt vector prefix register (IVPR) and interrupt-specific interrupt vector offset registers (IVORs):

- Critical save/restore registers (CSRR0 and CSRR1). Defined to save and restore machine state on critical interrupts (critical input, machine check, watchdog timer, and debug) and are analogous to SRR0 and SRR1.
- Exception syndrome register (ESR). The ESR provides a way to differentiate between exceptions that can generate an interrupt type.
- Data exception address register (DEAR). Loaded with the effective address of a data access (caused by a load, store, or cache management instruction) that results in an alignment, data TLB miss, or DSI exception.
- Interrupt vector prefix register (IVPR). Used with IVORs to determine the vector address. The 12-bit vector offsets are concatenated to the right of IVPR to form the address of the interrupt routine.
- Interrupt vector offset registers (IVOR0–IVOR31). IVORs provide the index from the base address provided by the IVPR for its respective interrupt type. IVORs provide storage for specific interrupts. The Power ISA definition allows implementations to define IVORs to support category- and implementation-specific interrupts. For example, the SPE defines IVOR32–IVOR35. Such IVORs are listed at the bottom of Table 11.

The machine check interrupt model, part of the Embedded category, defines the following registers:

- Machine check save/restore registers (MCSRR0 and MCSRR1). Analogous to SRR0 and SRR1.
- Machine check syndrome register (MCSR). When the core complex takes a machine-check interrupt, it updates MCSR to differentiate between machine-check conditions. The MCSR indicates whether a machine-check condition is recoverable.
- Machine check address register (MCAR). When the core complex takes a machine-check interrupt, it updates MCAR to indicate the address of the data associated with the machine check.

Configuration Registers

The PowerPC architecture defines registers that provide control, configuration, and status information of the machine state and process IDs. Figure 11 shows a comparison of PowerPC architecture 1.10 and the PowerPC architecture 2.04/EIS configuration registers.

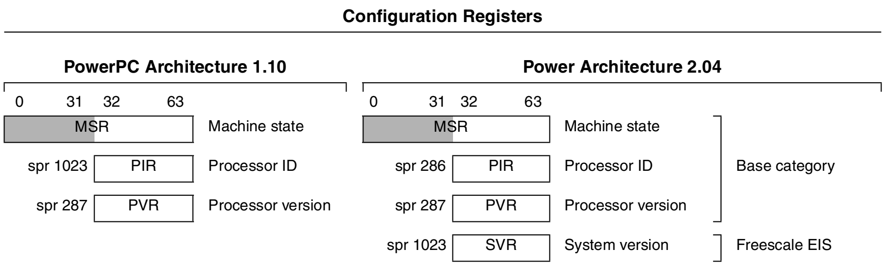


Figure 11. Configuration Registers Comparison

PowerPC architecture 1.10 and the Power Architecture 2.04 specification both include a versatile register that provides control and configuration of interrupts:

- Machine state register (MSR). Defines the state of the processor (that is, enabling and disabling of interrupts and debugging exceptions, enabling and disabling some features, and specifying whether the processor is in supervisor or user mode).

  The PowerPC architecture 1.10 MSR supports bits that enable data address translation (IR and DR) and modal big/little endian byte ordering (LE and ILE). Embedded category devices support configuration of byte ordering as a page attribute through the MAS registers.

  The MSR provides enable bits for machine-check, external, and critical interrupts. MSR contents are automatically saved, altered, and restored by the interrupt-handling mechanism. If an interrupt is taken, MSR contents are automatically copied into the appropriate save/restore register, for example, SRR1. When the corresponding Return from Interrupt instruction, (for example, **rfi**) is executed, MSR contents are restored.
- Processor ID register (PIR). Contains a value that can be used to distinguish the processor from other processors in the system. Note that the PowerPC architecture 1.10 and Embedded category PIR SPR numbers differ.
- Processor version register (PVR). Contains a value identifying the version and revision level of the processor. The PVR distinguishes between processors whose attributes may affect software.

The EIS defines the system version register (SVR), which identifies the integrated device in which the core is implemented.

Performance Monitor Registers (PMRs)

The set of registers shown in Figure 12 are used exclusively by the Embedded.Performance Monitor category. PMRs are similar to the SPRs and are accessed by **mtpmr** and **mfpmr** instructions, which are also part of this category.

The counter registers, global controls, and local controls have alias names that cause the assembler to use different PMR numbers. The names PMC0–PMC15, PMGC0, PMLCa0–PMLCa15, and PMLCb0–PMLCb15 cause the assembler to use the supervisor-level PMR number, and the names UPMC0.–UPMC15, UPMGC0, UPMLCa0–UPMLCa15, and UPMLCb0–UPMLCb15 cause the assembler to use the user-level PMR number. User-level access to these PMRs is read-only.

Although the PowerPC architecture 1.10 does not define performance monitor registers, most PowerPC processors implemented a performance monitor using implementation-specific SPRs rather than PMRs.

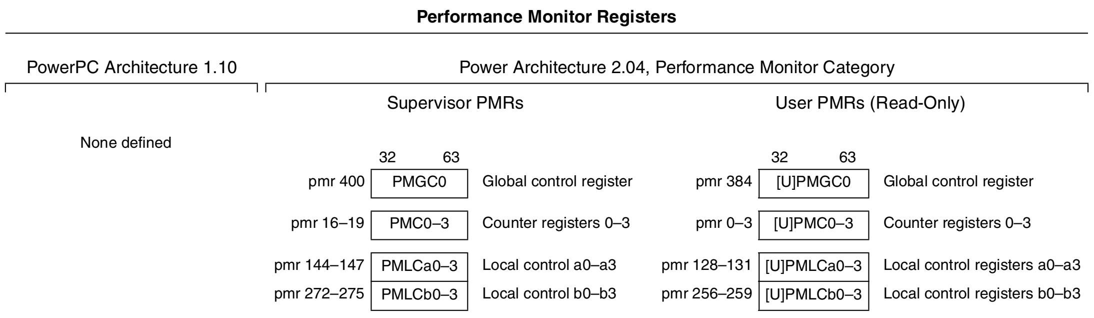


Figure 12. Performance Monitor Registers Comparison

The following describes the PMRs:

- Global control register (PMGC0/UPMGC0). PMGC0 controls all performance monitor counters and is a supervisor-level register. The contents of PMGC0 are reflected to UPMGC0, which is read by user-level software.
- Performance monitor counter registers (PMC0–PMC3/UPMC0–UPMC3). PMC0–PMC3 are 32-bit counters that can be programmed to generate interrupt signals when they overflow. Each counter is enabled to count 128 events. The contents of PMC0–PMC3 are reflected to UPMC0–UPMC3, which are read by user-level software.
- Local control registers facilitate software control of the PMRs:
  - — PMLCa0–PMLCa3/UPMLCa0–UPMLCa3. PMLCa registers function as event selectors and give local control for the corresponding performance monitor counters. Each PMLCa works with the corresponding PMLCb register.

    The contents of PMLCa0–PMLCa3 are reflected to UPMLCa0–UPMLCa3, which are read by user-level software and are read-only.
  - — PMLCb0–PMLCb3/UPMLCb0–UPMLCb3. PMLCb registers specify a threshold value and a multiple to apply to a threshold event selected for the corresponding performance monitor counter. Each PMLCb works with the corresponding PMLCa.

    The contents of PMLCb0–PMLCb3 are reflected to UPMLCb0–UPMLCb3, which are read by user-level software.

Debug Registers

Debug registers are accessible to software running on the processor. These registers are intended for use by special debug tools and debug software, and not by general application or operating system code. Figure 13 shows a comparison of PowerPC architecture 1.10 and the Power ISA debug registers.

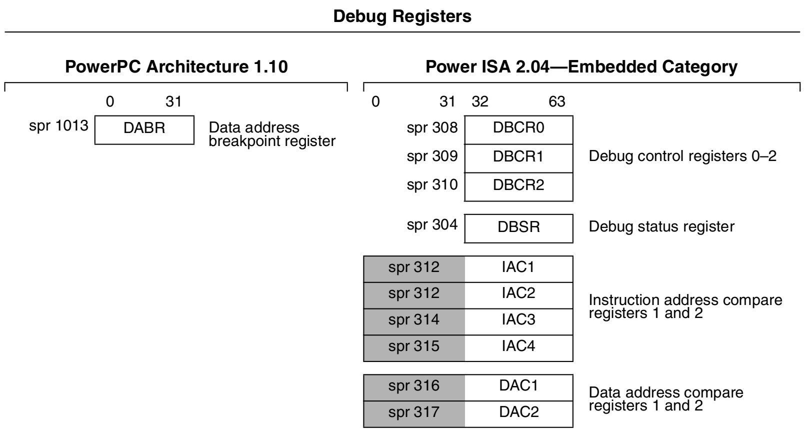

Debug Registers
PowerPC Architecture 1.10
Power ISA 2.04—Embedded Category
0
31
32
63
spr 1013
DABR
Data address breakpoint register
spr 308
DBCR0
spr 309
DBCR1
spr 310
DBCR2
Debug control registers 0–2
spr 304
DBSR
Debug status register
spr 312
IAC1
spr 312
IAC2
spr 314
IAC3
spr 315
IAC4
Instruction address compare registers 1 and 2
spr 316
DAC1
spr 317
DAC2
Data address compare registers 1 and 2

Figure 13. Debug Registers Comparison

The PowerPC architecture 1.10 definition provides one register to facilitate debugging:

- Data address breakpoint register (DABR). The data address breakpoint facility provides a means to detect accesses to a designated word. A data address breakpoint match is detected for a load or store instruction and a match generates a DSI exception. The address comparison is done on an effective address, and it applies to data accesses only.

The Embedded category definition provides debugging support at data and instruction addresses:

- Debug control registers (DBCR0–DBCR1). Enable debug events, reset the processor, control timer operation during debug events, and set the debug mode of the processor.
- Debug status register (DBSR). Provides status information for debug events and for the most recent processor reset. The DBSR is set through hardware but is read and cleared through software.
- Instruction and data address compare registers (IAC1–IAC4, DAC1–DAC2). A debug event may be enabled to occur upon an attempt to execute an instruction or access a data location from an address specified in an IAC/DAC, inside or outside a range specified by the IACs/DACs, or to blocks of addresses specified by the combination of the IACs/DACs.

Note that the embedded.enhanced debug category defines additional interrupt resources, described in "Interrupt Registers" on page 34.

Implementation-Specific Registers

To handle special functions, implementations typically have additional SPRs not defined by the architecture, and some of these registers may appear on multiple implementations with similar functionality. In particular, implementations define hardware implementation-dependent registers (HIDs) that typically control hardware-related functionality.

Interrupt Model

To meet the demands on embedded processors in highly integrated devices, the Embedded category defines an interrupt model that is more agile and responsive. In a real-time OS in which the core supports a complex, integrated system-on-a-chip, system performance is to a large part measured by the efficiency of the response to interrupt requests generated through peripheral logic. To reduce interrupt response time to crucial interrupts, Book E defined a second interrupt type, the critical interrupt, with separate save and restore resources, CSSR0 and CSRR1 the Return from Critical Interrupt instruction (**rfci**). These resources allowed critical-type interrupts to be taken without having to save state of any concurrent non-critical interrupts. The EIS extended this notion by defining similar interrupt types for debug and machine-check interrupts. These are now part of the Embedded category, as shown in Table 10.

Table 10. Further Differentiation of the Book E Critical Interrupt Model

Interrupt	Book E/EIS		Power ISA 2.04
	Book E	EIS	
Critical input (analogous to the non-critical external interrupt)	Critical interrupt	—	Embedded category
Machine check	Critical interrupt	Machine-check APU	Embedded category
Watchdog timer (analogous to non-critical fixed-interval timer interrupt defined in Book E)	Critical interrupt	—	Embedded category
Debug	Critical interrupt	Debug APU	Embedded.enhanced debug category

Most of the features of the interrupt model are common to all architecture versions. The PowerPC interrupt mechanism allows the processor to change to supervisor state as a result of external signals, errors, or unusual conditions arising in the execution of instructions. When interrupts occur, information about the state of the processor is saved to certain registers and the processor begins execution at an address (interrupt vector) predetermined for each interrupt. Processing of interrupts begins in supervisor mode.

Exception conditions may be defined at non-supervisor levels of the architecture. For example, the user instruction set architecture defines conditions that may cause floating-point exceptions; the OEA defines, at the supervisor level, the mechanism by which the interrupt is taken.

The Power Architecture model differentiates between the terms ‘exception’ and ‘interrupt’. Use of these terms in this document are as follows:

- An exception is the event that, if enabled, causes the processor to take an interrupt. The architecture describes exceptions as being generated by signals from internal and external peripherals, instructions, the internal timer facility, debug events, or error conditions.
- An interrupt is the action a processor takes in response to an exception. The processor saves its context (typically the MSR settings and return instruction address) and begins execution at a predetermined interrupt handler address with a modified MSR.

The architecture requires that interrupts be taken in program order; therefore, although a particular implementation may recognize exception conditions out of order, they are handled strictly in order with respect to the instruction stream. When an instruction-caused interrupt is recognized, any unexecuted instructions that appear earlier in the instruction stream, including any that have not yet entered the execute state, are required to complete before the interrupt is taken.

Interrupts can occur while an interrupt handler routine is executing, and multiple interrupts can become nested. It is up to the interrupt handler to save the appropriate machine state if it is desired to allow control to ultimately return to the interrupting program.

In many cases, after the interrupt handler handles an interrupt, there is an attempt to execute the instruction that caused the interrupt. Instruction execution continues until the next exception condition is encountered. This method of recognizing and handling interrupts sequentially guarantees that the machine state is recoverable and processing can resume without losing instruction results.

To prevent the loss of state information, soon after the interrupt is taken, interrupt handlers must save the information stored in save/restore registers (the return address and the MSR settings) as well as other registers (such as the ESR and DBSR) that might be affected by an interrupt to prevent this information from being lost due to another interrupt being taken.

All interrupts except some machine-check interrupts are recoverable. The conditions that cause a machine check may prohibit recovery.

Because multiple exception conditions can map to a single interrupt, a more specific condition may be determined by examining a register associated with the exception—for example, the exception syndrome register (ESR) and the floating-point status and control register (FPSCR). Additionally, certain exception conditions can be explicitly enabled or disabled by software.

Interrupts, their offsets, and conditions that cause them for the PowerPC architecture 1.10 and the Power Architecture 2.04 specification are summarized in Table 11. The Embedded category includes the Book E–defined interrupt vector offset registers (IVORs) to handle each interrupt type, whereas the PowerPC architecture 1.10 implementation uses fixed-location vector offsets. Unless otherwise specified, MSR settings and the return address for every interrupt are stored in save/restore registers.

- Interrupts in the PowerPC architecture 1.10 definition—The PowerPC interrupt model uses fixed addresses as vector offsets to map to physical memory locations with the base address determined by the MSR[IP]. If IP is zero, vector offsets are added to the physical address 0x000n_nnnn. If IP is set, vector offsets are added to the physical address 0xFFFn_nnnn. Table 11 shows the vector offsets associated with each interrupt type. Finally, the PowerPC architecture includes the system reset, trace, and floating-point assist interrupts as part of its definition.
- Interrupts in the Power ISA 2.04 Embedded category. Defines interrupt vector offset registers (IVORs), interrupt vector prefix registers (IVPRs), and critical interrupts. An IVOR is assigned to each interrupt type. The IVPR provides the base address location to which the offset in the IVORs is added. Table 11 shows the IVORs associated with each interrupt type.

  The save and restore resources are part of the Base and Embedded categories and are largely identical to those defined by the OEA. Save and restore registers (shown in Figure 10) save the return address and machine state when they are taken. A return from interrupt instruction (**rfi**, **rfci**, **rfdi**, or **rfmci**) restores state at the end of the interrupt routine.
- In addition to the critical type interrupt originally defined by Book E, the Embedded category defines analogous resources for machine-check and debug interrupts. The Power ISA resources are defined as follows:

- Critical interrupts (Base category)—Higher-priority interrupts that use separate save/restore resources analogous to those defined for non-critical interrupts. These consist of the critical save and restore registers (CSRR0/CSRR1) and the Return from Critical Interrupt instruction **rfci** instruction to restore state.

  The Power ISA version defines the critical input, watchdog timer, and debug interrupts as critical interrupts (although debug interrupts may be implemented as separate interrupt types).
- Machine-check interrupt (Embedded category)—Implements save and restore registers (MCSRR0/MCSRR1) used to save the return address and machine state when machine-check interrupts are taken. The **rfmci** instruction is used to restore state.
- Debug interrupt (embedded.enhanced debug category)—implements save and restore registers (DSRR0/DSRR1) used to save the return address and machine state when debug interrupts are taken. The **rfdi** instruction is used to restore state.

• Other categories, such as the SPE and performance monitor, define non-critical interrupts to handle category-specific program interrupts.

Table 11 shows a comparison of the PowerPC architecture 1.10 and Power ISA 2.04 interrupt models.

Table 11. Interrupts and Conditions—Overview

Interrupt Type	Vector Offset		Causing Conditions
	Power ISA	PowerPC	
Power ISA 2.04 Embedded Category Interrupts			
Critical input	IVOR0	—	Typically caused by assertion of an asynchronous signal; presented to the interrupt mechanism. Similar to external interrupts.
System reset	—	0x100	Caused by implementation-defined asynchronous conditions.
Machine check	IVOR1	0x200	Causes are implementation-dependent but typically related to conditions such as bus parity errors or attempts to access an invalid physical address. Typically, these interrupts are triggered by an input signal to the processor. Disabled when MSR[ME] = 0; if a machine-check interrupt condition exists, the processor goes into checkstop. Machine-check interrupts have separate resources MCSRR0 and MCSRR1 and the Return from Machine Check Interrupt instruction (**rfmci**). An address related to the machine check may be stored in MCAR. MCSR reports the cause of the machine check.
DSI	IVOR2	0x300	A data memory access cannot be performed. Such accesses can be generated by load/store instructions and by certain memory control and cache control instructions. The ESR reports the cause; DEAR holds the effective address of the data access. PowerPC architecture 1.10: DSISR reports the cause; DAR is set based on DSISR settings.

Table 11. Interrupts and Conditions—Overview (continued)

Interrupt Type	Vector Offset		Causing Conditions
	Power ISA	PowerPC	
ISI	IVOR3	0x400	Instruction fetch cannot be performed. Causes include the following: • The effective address cannot be translated. For example, when there is a page fault for this portion of the translation, an ISI must be taken to retrieve the page (and possibly the translation), typically from a storage device. • An attempt is made to fetch an instruction from a no-execute memory region or from guarded memory when MSR[IR] = 1. • The fetch access violates memory protection. Embedded category devices use ISI to assist implementations that cannot dynamically switch byte ordering between consecutive accesses, do not support the byte order for a class of accesses, or do not support misaligned accesses using a specific byte order. ESR reports the cause. The VLE category defines additional misaligned, mismatched, and byte-ordering storage exceptions.
External interrupt	IVOR4	0x500	Generated only when an external interrupt is pending (typically signaled by a signal specified by the implementation) and the interrupt is enabled (MSR[EE]=1).
Alignment	IVOR5	0x600	The processor cannot perform a memory access because of one of the following: • The operand of a load or store is not aligned. • a **dcbz** instruction referenced storage that is write-through required or cannot be established in the data cache. Embedded category devices use ESR to report the interrupt cause; DEAR holds the effective address of the data access. PowerPC architecture 1.10: DSISR reports the cause; DAR is set based on DSISR.
Floating-point unavailable	IVOR7	0x800	Caused by an attempt to execute a floating-point instruction (including floating-point load, store, and move instructions) when the floating-point available bit is cleared, MSR[FP] = 0.
Decrementer	IVOR10	0x900	The most-significant DEC bit changes from 0 to 1 and the interrupt is enabled (MSR[EE] = 1). If it is not enabled, the interrupt remains pending until it is taken. In Embedded category devices, TSR records status on timer events. An auto-reload value in the DECAR is written to DEC when it decrements from 0x0000_0001 to 0x0000_0000.
System call	IVOR8	0xC00	Occurs when a System Call (**sc**) instruction is executed.
Trace	—	0xD00	Optional. Either MSR[SE] = 1, almost any instruction successfully completed, or MSR[BE] = 1, and a branch instruction is completed.
Floating-point assist	—	0xE00	Optional. Can be used to provide software assistance for infrequent and complex floating-point operations such as denormalization.
Auxiliary processor unavailable	IVOR9	—	An attempt is made to execute an auxiliary processor instruction (including loads, stores, and moves), the target auxiliary processor is implemented, but is configured as unavailable.
Fixed interval timer	IVOR11	—	A fixed-interval timer exception exists (TSR[FIS] = 1), and the interrupt is enabled (TCR[FIE] = 1 and MSR[EE] = 1).
Watchdog timer	IVOR12	—	Critical interrupt. Occurs when a watchdog timer exception exists (TSR[WIS] = 1), and the interrupt is enabled (TCR[WIE] = 1 and MSR[CE] = 1).
Data TLB error	IVOR13	—	A virtual address associated with an instruction fetch does not match any valid TLB entry.

Table 11. Interrupts and Conditions—Overview (continued)

Interrupt Type	Vector Offset		Causing Conditions
	Power ISA	PowerPC	
Instruction TLB error	IVOR14	—	A virtual address associated with a fetch does not match any valid TLB entry.
Debug	IVOR15	—	Critical interrupt. A debug event causes a corresponding DBSR bit to be set and debug interrupts are enabled (DBCR0[IDM] = 1 and MSR[DE] = 1).
Reserved	IVOR16–IVOR31	—	Additional IVORs not listed below are reserved for future architectural use.
SPE Category Interrupts			
Vector unavailable	IVOR32	—	MSR[VEC] is cleared and a Vector category instruction is executed.
SPE Category Interrupts			
SPE unavailable	IVOR32	—	MSR[SPE] is cleared and an SPE or embedded floating-point instruction is executed.
Embedded floating-point data	IVOR33	—	Embedded floating-point invalid operation, underflow or overflow exception
Embedded floating-point round	IVOR34	—	Embedded floating-point inexact or rounding error
Performance Monitor Category Interrupts			
Performance monitor	IVOR35	—	An interrupt-enabled event defined by the performance monitor occurred. Embedded performance monitor category.

Memory Management Unit (MMU) Model

The MMU, together with the exception-processing mechanism, make it possible for an operating system to implement a paged virtual-memory environment and to define and enforce characteristics of that memory space, such as cache coherency and memory protection. Virtual memory management permits execution of programs larger than the size of physical memory; the term ‘demand-paged’ implies that individual pages are loaded into physical memory from backing storage only as they are accessed by an executing program.

Generally, the address translation mechanism is defined in terms of mapping an effective-to-physical address for memory accesses. The effective address is converted to an interim virtual address and a page table is used to translate the virtual address to a physical address.

In addition to instruction accesses and data accesses generated by load and store instructions, addresses specified by cache instructions also require address translation. This section describes how such translations are managed on a per-page basis using the resources defined in a general way by the Power ISA, and more specifically in the category.Embedded.MMU Type FSL.

Translation lookaside buffers (TLBs) are commonly implemented to keep recently used page address translations on-chip. Although their exact characteristics are not specified in the architecture, the general concepts pertinent to the system software are described.

The model described here shares some general characteristics of the MMU model definition in the PowerPC architecture 1.10, particularly those related to memory protection and cache coherency and the general concepts of pages as TLBs. Differences are described in "MMU Features in the PowerPC Architecture 1.10 Definition," on page 46.

MMU Features in the Embedded Category Definition

The simplicity and flexibility of the category.Embedded.MMU Type FSL is suited especially for embedded applications. It supports demand-paged virtual memory as well as a variety of management methods that depend on precise control of effective-to-real address translation and configurable memory protection. Address translation misses and protection faults cause precise exceptions. Sufficient information is available to correct the fault and restart the faulting instruction.

Each program on a 32-bit implementation can access 2^{32} bytes of effective address space, subject to limitations imposed by the operating system. In a typical system, each program's EA space is a subset of a larger virtual address (VA) space managed by the operating system.

Each effective (logical) address is translated to a real (physical) address before being used to access physical memory or an I/O device. The operating system manages the physically addressed resources of the system by setting up the tables used by the address translation mechanism.

The effective address space is divided into pages. The page represents the granularity of effective address translation, permission control, and memory/cache attributes. Multiple page sizes may be simultaneously supported. They can be as small as 1 Kbyte. The maximum size depends on the implementation. For an effective-to-real address translation to exist for a page, a valid entry containing the effective address must be in a translation lookaside buffer (TLB). Addresses for which no TLB entry exists cause TLB miss exceptions (instruction or data TLB error interrupts).

The MMU model defines a set of MMU assist (MAS) registers that can be programmed via the **mtspr** instructions to update the TLBs directly with translation and configuration information. The configuration data in the MAS registers is written to the TLBs on the execution of a TLB Write Entry (**tlbwe**) instruction. Likewise, when a TLB needs to be reprogrammed, the contents can be saved back to the MAS registers by executing a TLB Read Entry (**tlbre**) instruction.

The instruction addresses generated by a program and the addresses used by load, store, and cache management instructions are effective addresses. However, in general, the physical memory space may not be large enough to map all the virtual pages used by the active applications. With support provided by hardware, the operating system can attempt to use the available real pages to map a sufficient set of virtual pages for an application. If a sufficient set is maintained, paging activity is minimized, improving performance.

The operating system can restrict access to virtual pages on a per-page basis by selectively granting permissions for user state read, write, and execute; and supervisor state read, write, and execute. These permissions can be set up for a particular system (for example, program code might be execute-only, data structures may be mapped as read/write/no-execute) and can also be changed by the operating system based on application requests and operating system policies.

MMU Features in the PowerPC Architecture 1.10 Definition

The PowerPC architecture 1.10 supports three types of address translation: page-address translation, block address translation and real mode (where the hardware translation mechanism is turned off and the effective address is used as the physical address).

Page address translation is defined in terms of segment descriptors. The segment information translates the effective address to an interim virtual address, and the page-table information translates the virtual address to a physical address. Effective address spaces are divided into 256-Mbyte segments. Segments that correspond to memory-mapped areas are divided into 4-Kbyte pages.

The definition of the segment and page-table data structures provides significant flexibility for a wide range of computing environments. Therefore, the methods for storing segment or page-table information on-chip vary from implementation to implementation.

The segment information, used to generate the interim virtual addresses, is stored in on-chip, software-accessible segment registers. The MMU then uses these descriptors to generate the physical address, the protection information, and other access-control information each time an address within the page is accessed. Address descriptors for pages reside in tables (as PTEs) in physical memory; for faster accesses, the MMU often caches on-chip copies of recently used PTEs in an on-chip TLB.

The PowerPC architecture 1.10 also defines the block address translation (BAT) mechanism. Simpler than page-translation, block-address translation allows the operating system to configure attributes for blocks of memory through a set of 16 SPRs. As Figure 8 shows, there are four pairs of BAT SPRs for instruction memory (IBATs) and four pairs for data memory (DBATs). The BAT registers include fields for configuring the size (128 Kbytes to 256 Mbytes) and location of the blocks. Some implementations extend the number of BAT registers beyond those defined by the architecture.

Differences between the MMU Models

The flow diagram in Figure 14 describes the address translation mechanisms of the Power ISA 2.03 and PowerPC architecture 1.10.

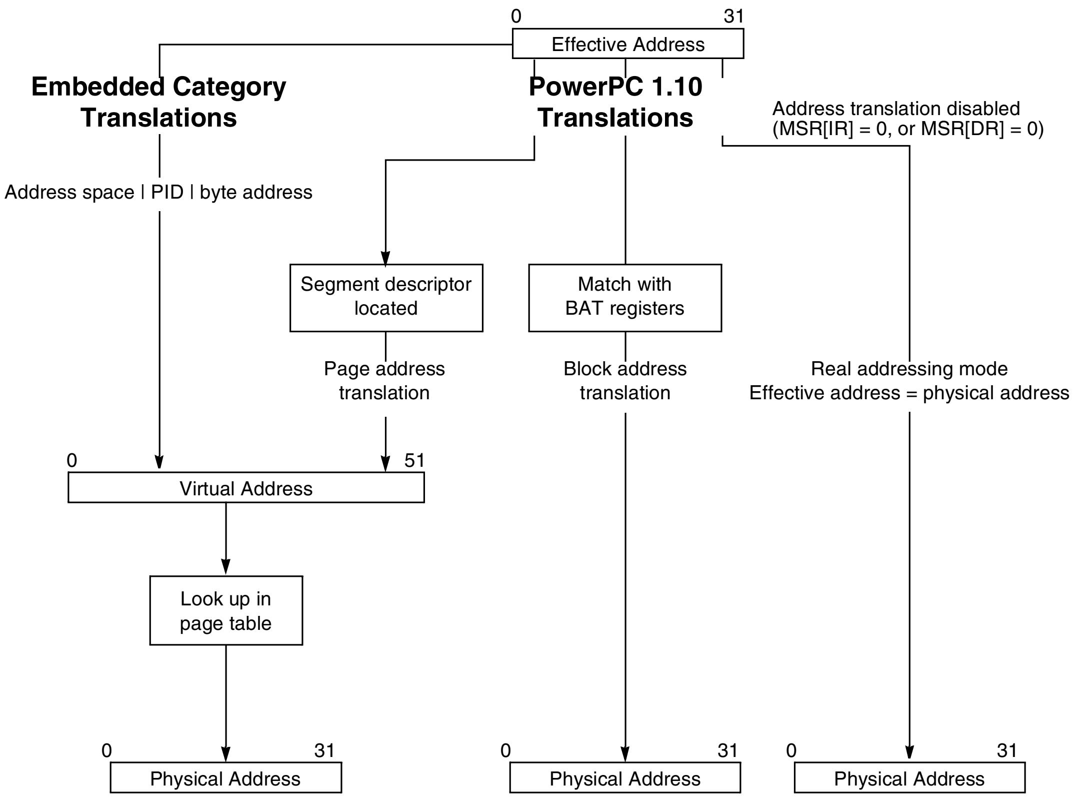

0
31
Effective Address
Embedded Category Translations
PowerPC 1.10 Translations
Address translation disabled (MSR[IR] = 0, or MSR[DR] = 0)
Address space | PID | byte address
Segment descriptor located
Match with BAT registers
Page address translation
Block address translation
Real addressing mode Effective address = physical address
0
51
Virtual Address
Look up in page table
0
31
Physical Address
0
31
Physical Address
0
31
Physical Address

Figure 14. Address Translation Types

Differences between the PowerPC architecture 1.10 and the Power Architecture 2.04 embedded category MMU models are outlined in Table 12.

Table 12. PowerPC Architecture 1.10 and Power Architecture 2.04 Embedded MMU Models

PowerPC Architecture 1.10	Power ISA 2.04 Embedded Category
Support for block address translation, page address translation, and real mode	Enhanced page address translation, no block address translation or real mode
Fixed 4-Kbyte pages	Support for both fixed and variable-sized page address translation mechanisms
Segmented memory model	No segments defined

Table 12. PowerPC Architecture 1.10 and Power Architecture 2.04 Embedded MMU Models (continued)

PowerPC Architecture 1.10	Power ISA 2.04 Embedded Category
Hardware page address translation definition with little architected support for software management	Hardware table hashing is not defined. Additional features are defined that support management of page translation and protection in TLBs in software. Two instructions, TLB Read Entry (**tlbre**) and TLB Write Entry (**tlbwe**), are defined that provide direct software access to page translation and configuration.
Byte ordering. Modal, big- endian and (munged) little-endian support provided through the MSR	Support for big- and true little-endian byte ordering provided on a per-page basis, programmed through the TLBs
DSI and ISI interrupts taken when an address cannot be translated or a protection violation occurs	In addition to the DSI and ISI interrupts, data and instruction TLB error interrupts are taken if there is a TLB miss.

For example, the Embedded category defines the TLB Read Entry and TLB Write Entry instructions (**tlbre** and **tlbwe**) for reading and writing the TLBs in software but does not specify how this is to be accomplished. Freescale processors execute these instructions by reading or writing the contents of a set of MMU assist (MAS) SPRs into the TLBs. These MAS registers, which provide the translation, protection, byte-ordering, and cache characteristics for the relevant pages, and the exact behavior of the **tlbre** and **tlbwe** instructions, are defined by the Freescale embedded MMU category. Table 13 shows how the Power ISA 2.04 MMU features are defined.

Table 13. Embedded and Freescale Embedded MMU Categories

PowerPC ISA Embedded Category	PowerPC ISA Embedded MMU Category
TLB Read Entry (**tlbre**) and TLB Write Entry (**tlbwe**) give software direct access to page translation, protection, and configuration.	MMU assist registers (MAS*n*) defined as SPRs that hold translation, configuration, and protection information copied to and from the TLBs by executing **tlbwe** and **tlbre**.
A single process ID register (PID) used by system software to identify TLB entries used by the processor to accomplish address translation for loads, stores, and instruction fetches.	The Freescale embedded MMU category defines additional PID registers. (the Power ISA Embedded category–defined PID is treated as PID0). An implementation may choose to provide any number of PIDs up to a maximum of 15. The number of PIDs implemented is indicated by the value of MMUCFG[NPIDS].
	Additional MMU registers: • MMU configuration register (MMUCFG). Identifies the number of bits in a real address supported by the implementation, the number and size of PID registers, and the number of TLBs. • TLB*n*CFG registers (one for each implemented TLB array). Indicates the number of ways of associativity, minimum and maximum page size, page-size availability, number of entries, and invalidate protection capability in each TLB array. • MMUCFG0 used for general control of the MMU including flash invalidation of the TLB arrays and page sizes for programmable fixed size arrays

Power Architecture™ technology is incredibly efficient for general-purpose computing as well as an ideal platform for embedded applications. The minimal silicon requirements of the instruction set architecture enable high levels of integration, making it possible to pack RISC processor core and multiple peripheral functions on a single chip with low levels of power consumption and heat generation. Microprocessors built on Power Architecture technology also offer a compelling price/performance ratio, extended temperature options, multiprocessing capabilities, instruction set compatibility across the entire product line and a broad selection of development tools. Take a look inside Power Architecture technology and discover how it can open up possibilities for your designs.

Document Number: PWRARCPRMRM
Rev. 1