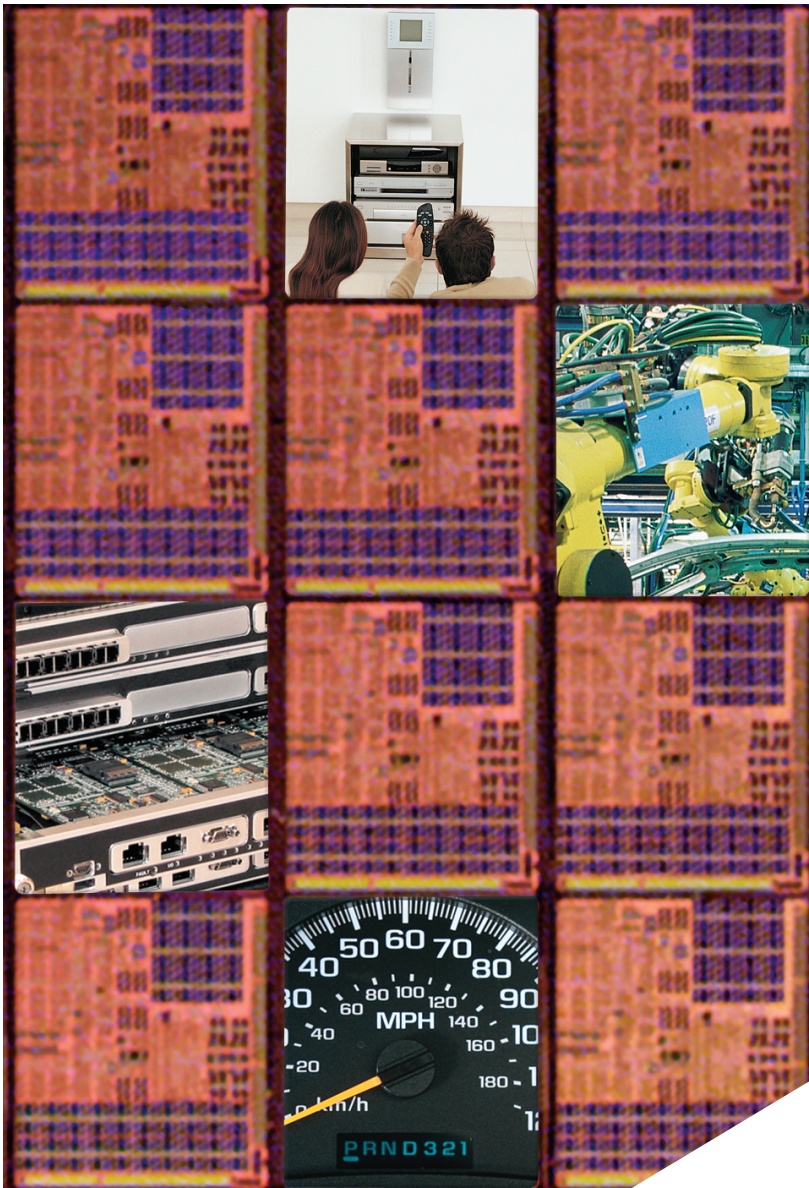


Power Architecture™ Technology Primer



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.

Freescal^e Power Architecture™ Primer

PWRARCPRMRM
Rev. 0, 07/2006



How to Reach Us:

Home Page:

www.freescale.com

email:

support@freescale.com

USA/Europe or Locations Not Listed:

Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
(800) 521-6274
480-768-2130
support@freescale.com

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)
support@freescale.com

Japan:

Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064, Japan
0120 191014
+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
(800) 441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. The PowerPC name is a trademark of IBM Corp. and is used under license. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© Freescale Semiconductor, Inc., 2006. All rights reserved.

Document Number: PWRARCPRMRM
Rev. 0, 07/2006



Contents

Section	Title	Page
	Coevolution—Power Architecture™ Technology and its Ecosystems	1
	The Power.org Community	2
History: The PowerPC Architecture		2
	The PowerPC Architecture Matures	3
	Architectural Extensibility—Alternatives to Book III’s Hardware-Based MMU Model.....	3
	Architectural Extensibility—AltiVec Technology	4
	Architectural Extensibility, Phase II—Book E, APUs, and Freescale’s EIS	4
	Auxiliary Processing Units (APUs)	5
	The Freescale Book E Implementation Standards (EIS)	5
	Architectural Extensibility Phase III—The Power ISA Definition	6
	Stability, Flexibility, Familiarity	6
	What’s New?.....	9
	What Has Changed?.....	9
	Book I Changes and Extensions	9
	Book II Changes	10
	Book III Changes	10
Power Architecture Details		11
	The Common User Instruction Set Architecture	11
	Categories, from the General to the Specific	12
	The Embedded Category	13
	AltiVec (Category.VEC)	14
	Floating-Point Categories—Floating-Point (FP) and Floating-Point with Record (FPR)	15
	Move Assist (Category.MA).....	15
	Signal Processing Engine (Category.SPE).....	15
	Embedded Vector and Scalar Single-Precision Floating-Point Categories	16
	Book VLE Category	16
Instruction Model.....		17
	Simplified Mnemonics.....	18
	Instruction Set Overview	18
	Integer Instructions	19
	Floating-Point Instructions (Category FP, FPR)	20
	Branch and Flow Control Instructions (Base Category).....	21
	Processor Control Instructions (Base Category).....	23
	Memory Synchronization Instructions.....	23
	Memory Control Instructions.....	24
Register Model.....		25
	Register Files	27
	Instruction-Accessible Registers.....	28
	Time Base Registers.....	29

Contents

Section	Title	Page
	MMU Control and Status Registers	30
	L1 Cache Registers	32
	Interrupt Registers.....	33
	Configuration Registers	35
	Performance Monitor Registers (PMRs)	36
	Debug Registers	38
	Implementation-Specific Registers	39
Interrupt Model		39
Memory Management Unit (MMU) Model.....		43
	MMU Features in the PowerPC Architecture 1.10 Definition	44
	MMU Features in the Embedded Category Definition.....	44
	Address Translation Mechanisms	46
	Interrupt Model	48

Coevolution—Power Architecture™ Technology and its Ecosystems

When computing first slipped away from the centralized mainframe, the design focus was primarily on two things—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.

Computing migrated from the vaults of tapes and mainframes, first into beige office appliances, and now into our kids’ pockets and grandmothers’ purses. As typewriters, floppies, adding machines, and the plain-old telephone become extinct, they are replaced by whole systems on chips—digital minotaurs, sphinxes, jackalopes, and the other fantastic electronic chimeras. These useful monsters make it possible for us to phone, email, text message, surf the web, calculate, watch videos, listen to music, check the scores and check our stocks, and play games—all without wires. Less visible, but no less influential on us, are their more reclusive cousins that have taken up residence in the telecommunications and networking infrastructures, and in our cars making them safer, more fuel-efficient, and even more fun to drive.

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

There is such burgeoning speciation and hybridization that the buzzword ‘ecosystem’ has become unavoidably apt. The improvements in semiconductor processes have opened opportunities to integrate more and more functionality, giving rise to the system on a chip (SoC).

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 technology of 2006 represents a merging of the existing PowerPC™ architecture specifications. 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™ 2.03 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 6](#).

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 introducing a new concept—categories. Every feature defined in the architecture is assigned to one (and only one) category.

The broadest category is the base category, which defines all of those elements common to all Power Architecture processors. Although the base category includes functionality defined in all three books, it most notably preserves almost all of the user, application-level resources defined in the original PowerPC

Book I technology, 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.

Where it is appropriate to do so, the Power Architecture technology also defines separate resources for embedded and server devices. These resources are identified as the embedded and server categories. This document focusses on the common functionality defined in the base category and the functionality specific to the embedded category as implemented on Freescale's 32-bit processors.

The concept of categories, which encompasses and extends the concept of auxiliary processing units (APUs), introduced in the PowerPC Book E architecture, makes it possible for the Power Architecture technology to further extend its programming model to create 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 Power Architecture specification. And it is that binary-compatible, application-level programming model, coupled with the ability to extend the architecture into new computing spaces, where 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 allow other types of implementations:

- 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, and 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 modularity made it possible for other architecture-compliant devices to leverage the PowerPC UISA without being constrained to Book III’s desktop-oriented operating system features, and the first embedded PowerPC processors did just that.

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, frequency of cache misses, interrupts, page fault, and other behaviors. Capturing such information made it possible to fine-tune software and hardware designs. The Power Architecture 2.03 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 an exception would be handled automatically by hardware or cause an interrupt. Often, new functionality, such as the performance monitor and the AltiVec™ technology (now the vector category), added interrupts.

Many of these implementation features were passed on to other processors to become family traits. In turn, some of these gradually became part of the architecture, as is the case with many EIS-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 the EIS, which now comprise the embedded.Freescale MMU component of the Power ISA 2.03 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 made 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 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 becomes category VEC. For details, see [“AltiVec \(Category.VEC\).”](#)

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 technology was 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. Befitting the APU concept, although it followed PowerPC rules for instruction syntax and form, it was never part of the PowerPC specification. After the concept of APUs was introduced, the AltiVec technology became an APU and was specified as 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 the 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 43.

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. This notion of APUs is part of the concept of categories in the Power ISA technology.

The Freescale Book E Implementation Standards (EIS)

Specifying the PowerPC 1.10 architecture in three books was to allow devices to implement the Book I UISA without adhering strictly to Book III. Likewise, leaving space in the opcode and SPR map 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 Power ISA Embedded.Freescale MMU category. Note that, because individual Freescale processors have different requirements, which MAS registers and which fields within those registers are used is left up to the individual implementation or implementation family (for example e500v2 supports 36-bit addressing, which requires an additional MAS register).

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

- Much of the MMU model, in particular the use of MAS registers (Embedded.Freescale MMU category)
- The cache-locking APU (now the cache-locking category)

- Major extensions to the ISA
 - The AltiVec APU (now the vector category)
 - Signal-processing engine APU (now the SPE category)
 - Embedded floating-point APUs (now the SPE.floating-point category)
 - Single-precision vector and scalar
 - Double-precision scalar

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 address the continually diversifying needs of the embedded ecosystem, ensuring consistency across all Freescale-specific categories, and providing a common layer of architecture between the Power Architecture definition and the implementations.

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 reinforces hardiness and vitality of the architecture.

The new Power Architecture definition represents a merging of Book E, Freescale's EIS, and IBM's server architecture, preserving the UISA, but defining separate Book III's—Book III-E for embedded category processors and Book III-S, for server category processors.

In the Power ISA definition, the Book E concept of APUs has been replaced by the more general concept of categories. Whereas Book E's APUs were all extensions to the architecture, in the Power ISA definition, every resource, including former APUs, falls into exactly one category. The **add** instruction defined in the original PowerPC Book I definition is now part of the base category, along with most other standard 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. However, a category may include features that are defined in multiple books. For example, the time base is part of the base category defined in Book II. Likewise, user instructions defined by the vector and SPE categories extend Book I, while the interrupt resources associated with these categories are defined in Book III-E.

The Power Architecture technology of 2006 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 Power Architecture 2.03, 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 the PowerPC 2.02 definition. 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 original 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 III's. It's as simple as [Figure 1](#).

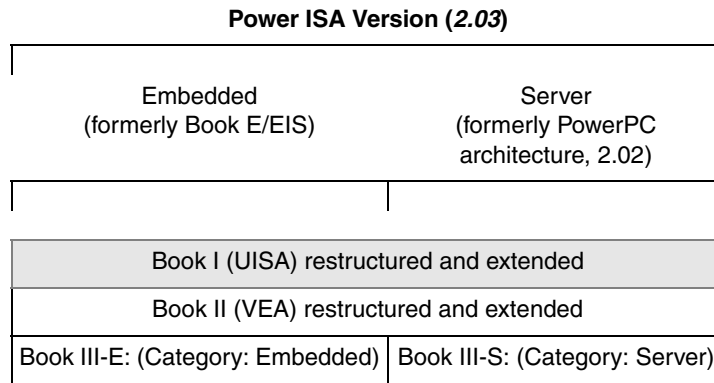


Figure 1. Power ISA Version (2.03)

The PowerPC architecture is the grandfather to the latest generation of the Power Architecture technology, the Power ISA version. [Figure 2](#) shows the relationship between the different components of the Power Architecture technology.

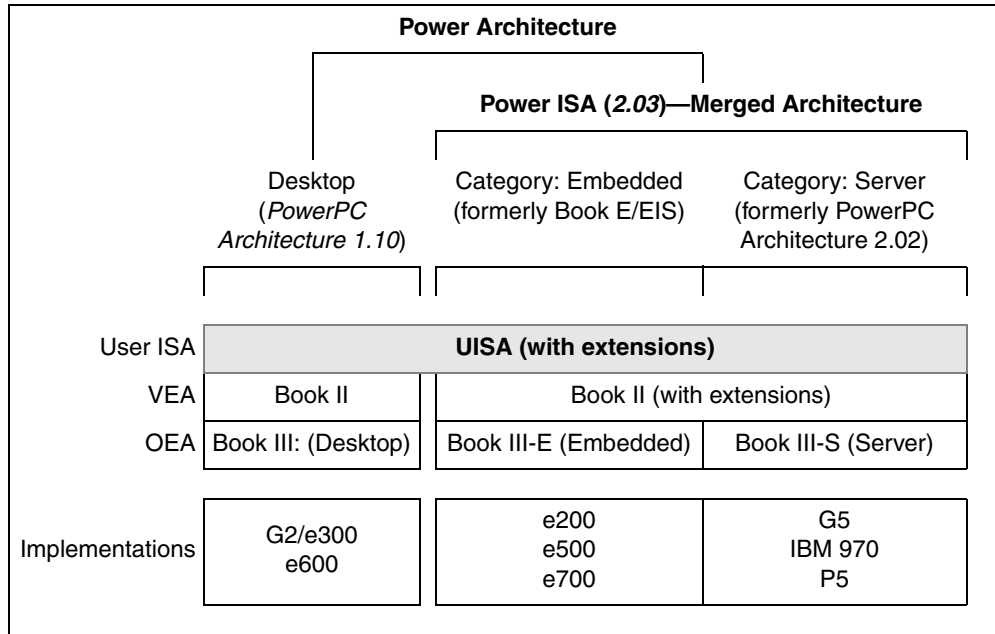


Figure 2. Power Architecture Relationships

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 2.03 specification. This definition is referred to as the merged architecture. It brings together the embedded features defined in Book E and the Freescale EIS with the server resources defined by IBM’s PowerPC architecture 2.02 definition.

The first published version of this merged architecture (version 2.03) will mostly reflect 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 will include new architected features that will appear in forthcoming processors. Subsequent versions of the architecture will include additional features.

As [Figure 2](#) shows, processors designed under the Book E/EIS and PowerPC advanced server 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 Advanced Server (AS) architecture 2.02 have been merged and reorganized, with several additional extensions (categories) that will be disclosed when the architecture is published.

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, and Freescale EIS architectures.

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 [“Categories, from the General to the Specific” on page 12](#).

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. 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 SPE category also extends Book III–defined features, in particular, the interrupt model. Implemented in the e200 and e500 cores.

- Variable length encoding (VLE). Variable-length encoding facility that reencodes some PowerPC opcodes to fit into 16 bits. Although it extends the UISA programming model, the VLE category is specified in a separate book, Book VLE. Implemented in some e200 cores. See [“Book VLE Category” on page 16](#).
- Vector (VEC). The vector category (nee AltiVec) 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, register, and interrupt models defined in the Power PC 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 in some environments 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 **sync** instruction. The **mbar** instruction, defined in Book E, remains as part of the embedded category; the equivalent PowerPC architecture 1.10 instruction **eiemo**, which shares the opcode, is defined as part of the server category.

Book III Changes

- The following categories are dependent categories of category 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 a category E 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 43, 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 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, and 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 and 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 continues to foster the development of further extensions as device-specific features, as EIS-defined categories (such as the L2 cache control category), and as additional categories in the Power Architecture model itself.

The UISA has provided the model for architecture expansion. The UISA defined the integer and floating-point register files—32 GPRs and 32 FPRs. The structure of these register files was extended, first as AltiVec’s vector registers (VRs), and then, as the SPE’s 64-bit GPRs, widened to accommodate two-element 64-bit operands.

Likewise, the PowerPC architecture defined special-purpose registers (SPRs) and two instructions to access them (Move to Special-Purpose Register (**mtspr**) and Move from Special-Purpose Register (**mfspr**)), and Book E defined device control registers (DCRs) accessed by **mtdcr** and **mfdcr** instructions. Likewise, the performance monitor category defines the performance monitor register (PMRs) and **mtpmr** and **mfpmr** instructions.

All instructions defined by the PowerPC architecture and by extensions to the architecture 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 introduces 16-bit encodings of some UISA-defined instructions not specified as part of Book I; these instructions are in a separate book, Book VLE.

Categories, from the General to the Specific

In the Power ISA definition, the Book E concept of APUs has been replaced by the more general concept of categories. Where Book E's APUs were all extensions to the architecture, in the Power ISA definition, every resource, including former APUs, falls into exactly one category. The **add** instruction defined in the original PowerPC Book I definition is now part of the base category, along with most other standard 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. However, a category may include features that are defined in multiple books. For example, the time base is part of the base category defined in Book II. Likewise, user instructions defined by the vector and SPE categories extend Book I, while the interrupt resources associated with these categories are defined in Book III-E.

Some categories are defined as dependent; that is, they are supported only if the category they are dependent on is also supported. Dependent categories are identified by the “.” in their category name. For example, a processor cannot implement the floating-point.record category (FP.R) without implementing 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.

The base category, Category.B, is the largest, encompassing all of the components that lie at the heart of the architecture regardless of the computing environment, the integer computational and load store

instructions, and the GPRs. Like many categories, the base category extends beyond Book I to include Book II and Book III features common to all Power Architecture devices, including the machine state register (MSR), the time base, the interrupt model's save and restore registers, and the instruction required for accessing them—all family traits that have been passed down from the parent PowerPC architecture, and in many cases, from the POWER architecture that preceded the PowerPC architecture.

All devices implement the facilities defined by the base category.

The next largest categories are the two that support the two computing environments to which the architecture is written, the server and embedded environments. The following gives a high-level description of the embedded category.

The Embedded Category

As described above, the embedded category 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 (**wrtee**) instruction, which can be used to update only MSR[EE].
- The software-use SPRs (SPRG4–SPRG7). The XSL category defines two more, SPRG8–SPRG9.
- Device control registers (DCRs), 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. The cache locking category also provides resources for detecting and handling overlocking conditions.
- Embedded.enhanced debug—Category E.ED. The enhanced debug category definition, drawn from the EIS and implemented in the e500 and e200 cores, defines a separate set of save and restore resources (DSRR0 and DSRR1, which save the return address and MSR settings, and the Return from Debug Interrupt instruction (**rfdi**) to restore those values at the end of the interrupt handler).
- 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. MAS5 is defined by the EIS.
 - 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 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 device-level behavior.

- For more information, see the [“Performance Monitor Registers \(PMRs\)” on page 36](#).

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

AltiVec (Category.VEC)

Although it precedes Book E and its definition of APUs, the AltiVec technology provides a prototype for the category concept in that it extends the instruction set, register, 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, but it also creates a 32-entry vector register (VR) file, similar to the GPRs and FPRs but widened to 128 bits to accommodate the 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
- A one-part/one-code-base approach to product design that also boosts performance
- A vector processing engine that provides highly parallel operations, allowing simultaneous execution of up to 16 operations per clock cycle
- Wide data paths and wide field operations that offer acceleration for a broad array of traditional computing and embedded processing operations
- A programmable solution that can easily migrate by using software upgrades to follow changing standards and customer requirements
- An integrated solution 100% compatible with the industry-standard PowerPC architecture that simplifies design and support

The AltiVec technology began an extension to the PowerPC architecture and is now a category in the Power ISA definition.

The ability to perform mathematical computation, logical operations, and bit manipulation simultaneously can be credited with providing a competitive edge in realms of computing far removed from those envisioned by the AltiVec architects.

AltiVec 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 to the architectural programming and interrupt model that is not part of the architecture itself, although its instructions and registers adhere to the conventions laid out in the PowerPC architecture.

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 spirit of providing an architecture that is scalable and extensible, 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) operation. 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 VEC 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 to accumulator operations, the SPE supports a number of forms of multiply and multiply-accumulate operations, as well as negative accumulate forms. These instructions are 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 SPE 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 SPE 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 support the vector single-precision and scalar double-precision categories. These extended GPRs are described in [“Register Files” on page 27](#).

Book VLE Category

There is perhaps no stronger 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 introduced into environments where there is driving need for a small code footprint. Like the vector and SPE 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 affecting 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.

Instruction Model

This section describes the 32-bit instructions and instruction classes as they are defined as part of the Power ISA 2.03 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 SPE category. See [“Floating-Point Instructions \(Category FP, FP.R\)” on page 20](#).
- 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 21](#).
- 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 23](#).
- 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. The PowerPC architecture 1.10–defined floating-point instructions use FPRs, while SPE embedded floating-point instructions share the GPRs. Instructions are 4 bytes (one word) long and word-aligned, and there is a penalty if instructions are not word-aligned. The architecture provides byte, half-word, and word operand loads and stores between

memory and a set of 32 general-purpose registers (GPRs). The PowerPC architecture 1.10 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 stored 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.

Programs written to be portable across the various assemblers for the PowerPC architecture should not assume the existence of instructions not described in that processor's reference manual.

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 **mfspir** 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 rS** can be used instead of **mtspr 9,rS**. 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 32-bit Power ISA version that is 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 considered to be mutually exclusive, so their opcodes may overlap. For example, the vector and SPE categories are both SIMD instruction sets that target distinctly different markets and so their opcodes may overlap. 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 defined as part of the base category except for the load/store string and multiple instructions, which make up the move assist 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 (addx , divx , mulx , negx , subx)	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 (cmpx)	Compare	Immediate, logical
Integer logical (andx , cnt , eqv , extx , nand , norx , orx , xorx)	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 (rlwx , slwx , srwx , srawx)	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 provided to enforce strict ordering.

Table 3. Integer Load/Store Instructions

Instruction	Instruction Name	Options/Comments
Integer load (l <i>bx</i> , l <i>hx</i> , l <i>wx</i>)	Load	Byte, word, half word, algebraic (half word), byte reverse, and zero, with update, indexed
Integer load multiple/string word: l <i>mw</i> , l <i>swi</i>	Load multiple word	Move assist category
	Load string word	
Integer store (st <i>bx</i> , st <i>hx</i> , st <i>wx</i>)	Store	Byte, word, half word, byte-reverse, with update, indexed
Integer store multiple/string word: st <i>mw</i> , st <i>swi</i>	Store multiple word	Move assist category
	Store string word	

Floating-Point Instructions (Category FP, FPR)

The floating-point model defined in the Power Architecture technology is written to the IEEE-754 standard, which defines conventions for 64- and 32-bit 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 computational, move, and select instructions operate on data located in FPRs and, except for the compare instructions whose results are reported to the condition register (CR), place the result value into an FPR. 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 the GPRs rather than a separate set of FPRs. See [“Embedded Vector and Scalar Single-Precision Floating-Point Categories”](#) on page 16.

The floating-point instructions are shown in [Table 4](#).

Table 4. Floating-Point Computational Instructions

Instructions	Instruction Name	Options
Floating-point elementary arithmetic (faddx , fdivx , fmulx , fsubx , fsqrtx , fresx , 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 (fmaddx)	Multiply-add	Single, double
	Multiply-subtract	Single, double
	Negative multiply-add	Single, double
	Negative multiply-subtract	Single, double
Floating-point rounding and conversion (fctix , frx)	Convert from integer	Double word
	Convert to integer	Word, double word, round to zero
	Round to single-precision	—
Floating-point compare and select (fcmx)	Compare	Ordered, unordered
	Select	—
Floating-point status and control register (mtfx , mffx)	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 (lfx)	Load floating-point	Double, single, with update, extended, indexed
Floating-point store (stfx)	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.
- The **trap** instruction tests for a specified set of conditions. If any of the conditions tested by a trap instruction are met, the system trap type program interrupt is taken. If the tested conditions are not met, instruction execution continues normally.
- The System Call (**sc**) instruction lets a user program call on the system to perform a service and causes the processor to take a system call interrupt. Executing this instruction causes the system call interrupt handler to be invoked. 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 is used to specify which of the 32 CR bits to test. Because it can be cumbersome for a programmer to remember the meanings of the various 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.” So instead of coding the instruction as ‘**bc** 16,0,target’ it can be coded ‘**bdnz** *target*.’

The supervisor-level **rfi** instruction is used for returning from a standard (base category) interrupt handler. The **rfdi** instruction is used for critical interrupts and **rfdci** is used for machine-check interrupts (embedded category), and **rfdi** is used for debug interrupts (embedded.enhanced debug category). See the section “[Interrupt Model](#)” on page 39.

Branch and flow control instructions are shown in [Table 6](#).

Table 6. Branch and Flow Control Instructions

Instruction	Name	Options
Branch (bx , bcx)	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 (crx , mcrx)	Condition register	AND/AND with complement, OR/OR with complement, XOR, NAND, NOR, Equivalent

Table 6. Branch and Flow Control Instructions (continued)

Instruction	Name	Options
Trap (tx , twx)	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 (rfx)	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 the CR and XER at the user level, as well as the machine state register (MSR) and most special-purpose registers (SPRs) at the supervisor level. 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 (**wrttee**) instruction, which can be used to update only MSR[EE].

Processor control instructions are shown in [Table 7](#).

Table 7. Processor Control Instructions

Instructions	Name	Options
Move (mtx , mfX)	Move to	SPR, CR fields, CR from XER, DCR, time base, MSR, PMR (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.

Table 8. Memory Synchronization Instructions (continued)

Instructions	Instruction Name	Comments
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 Power ISA 2.03 definition.
- 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 (dcbx , icbx)	Data cache block	Touch, touch for store, allocate, clear, store, flush (Embedded category)
	Instruction cache block	Invalidate, touch (Embedded category)
Supervisor-level cache (dcbi)	Data cache block	Invalidate (Embedded category)
Supervisor-level cache (dcbx)	Data cache block	Invalidate

Table 9. Memory Control Instructions (continued)

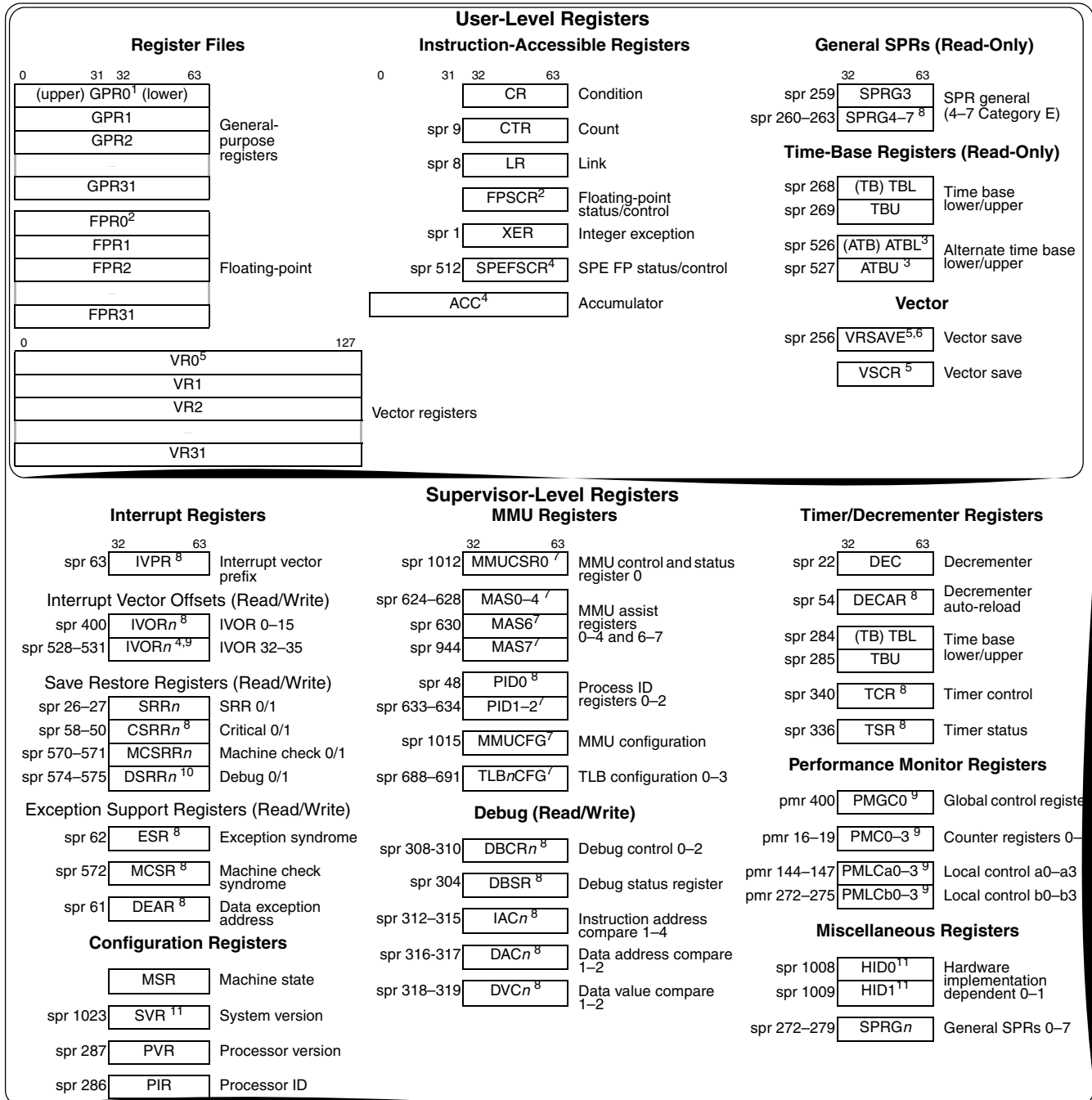
Instructions	Name	Options
dcbtls, dcbtstls, icbtls	Data/instruction cache block touch (for store) and lock set	Embedded.cache locking category
dcbxls, icbxls	Data/instruction cache block lock touch and set	
Segment register (mtsr_x/mfsr_x)	Move to/from SR	Indirect (PowerPC architecture 1.10)
TLB management (tlb_x)	TLB invalidate	Entry, all, virtual address indexed
	TLB synchronize	—
	TLB read entry	Embedded category
	TLB search indexed	Embedded category
	TLB write entry	Embedded category

Register Model

This section describes the Power Architecture 32-bit 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 defined in the PowerPC architecture 1.10 are essentially unchanged in the Power Architecture 2.03 model. Registers hold the source or destination of an instruction, or they are accessed as a by-product of execution.

- 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.
- Instruction-accessible registers—Registers such as the condition register (CR), the floating-point status and control register (FPSCR), and some SPRs are accessed as the 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 (performance monitor category), and device control registers, or DCRs (embedded category), offer large sets of on-chip registers similar to SPRs.



¹ 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 SPE category
⁵ Vector category (VEC), formerly AltiVec technology
⁶ Formerly USPRG0
⁷ Embedded.Freescale MMU category (E.MF)
⁸ Embedded category (E)
⁹ Embedded.performance monitor category (E.PM)
¹⁰ Embedded.enhanced debug category
¹¹ Defined by the EIS. Note that the HID registers contain fields defined by Power ISA categories

Figure 3. Power ISA 2.03 Register Model

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 3, can be accessed by user- or supervisor-level instructions; the VEA introduces the time base facility as user-level registers, also shown in Figure 3. 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 2.03 model.

Register Files

Figure 4 shows a comparison of PowerPC architecture 1.10 and Power Architecture register files.

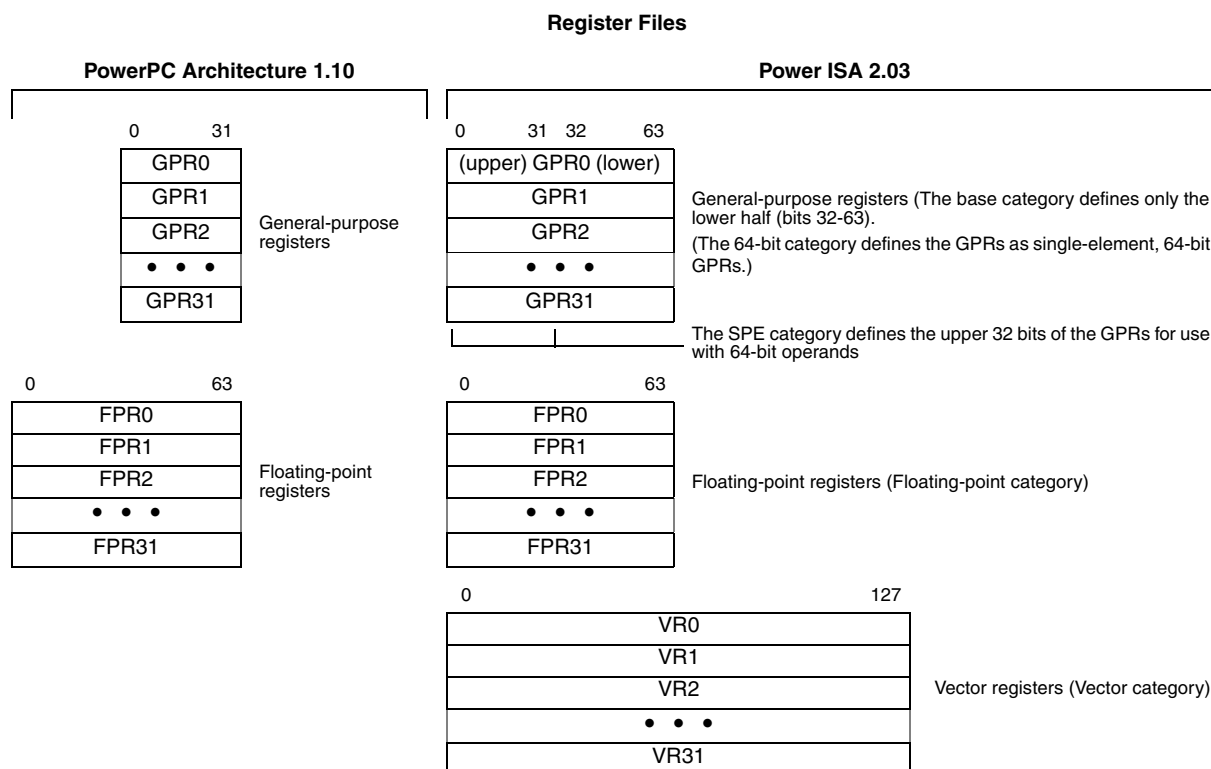


Figure 4. Register File Comparison

PowerPC architecture 1.10 and the Power Architecture 2.03 model both include GPR and FPR register 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 32-bit GPRs designated as GPR0–GPR31.

The SPE category defines a set of thirty-two 64-bit GPRs for use with the SPE category and its dependent floating-point categories.

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, FPR)” on page 20.
- Vector registers (VRs)—VRs act as either the source or destination of vector (AltiVec) instructions. The Power Architecture 2.03 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 5 shows a comparison of PowerPC architecture 1.10 and Power Architecture instruction-accessible registers.

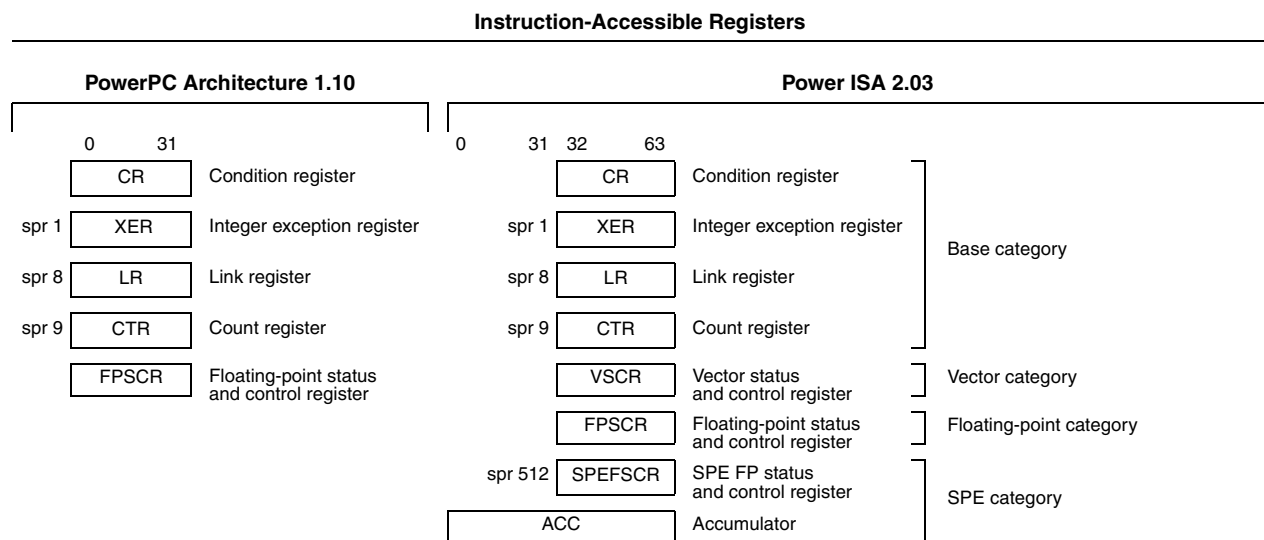


Figure 5. Instruction-Accessible Registers Comparison

PowerPC architecture 1.10 and the Power Architecture 2.03 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)

PowerPC architecture 1.10 and Book E registers include hardware and software timers:

- 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 2.03 definition.

The complexity of the modal 32-/64-bit MMU model in the PowerPC architecture 1.10 presented impediments to portability and was replaced in Book E by additional registers and instructions in the UISA, and by slight modifications to existing instructions to extend the addressability. 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 43](#) for more information on the MMU.

Figure 7 compares the MMU registers defined by the PowerPC architecture 1.10 with those defined by the Power ISA version and the Freescale EIS to support embedded devices.

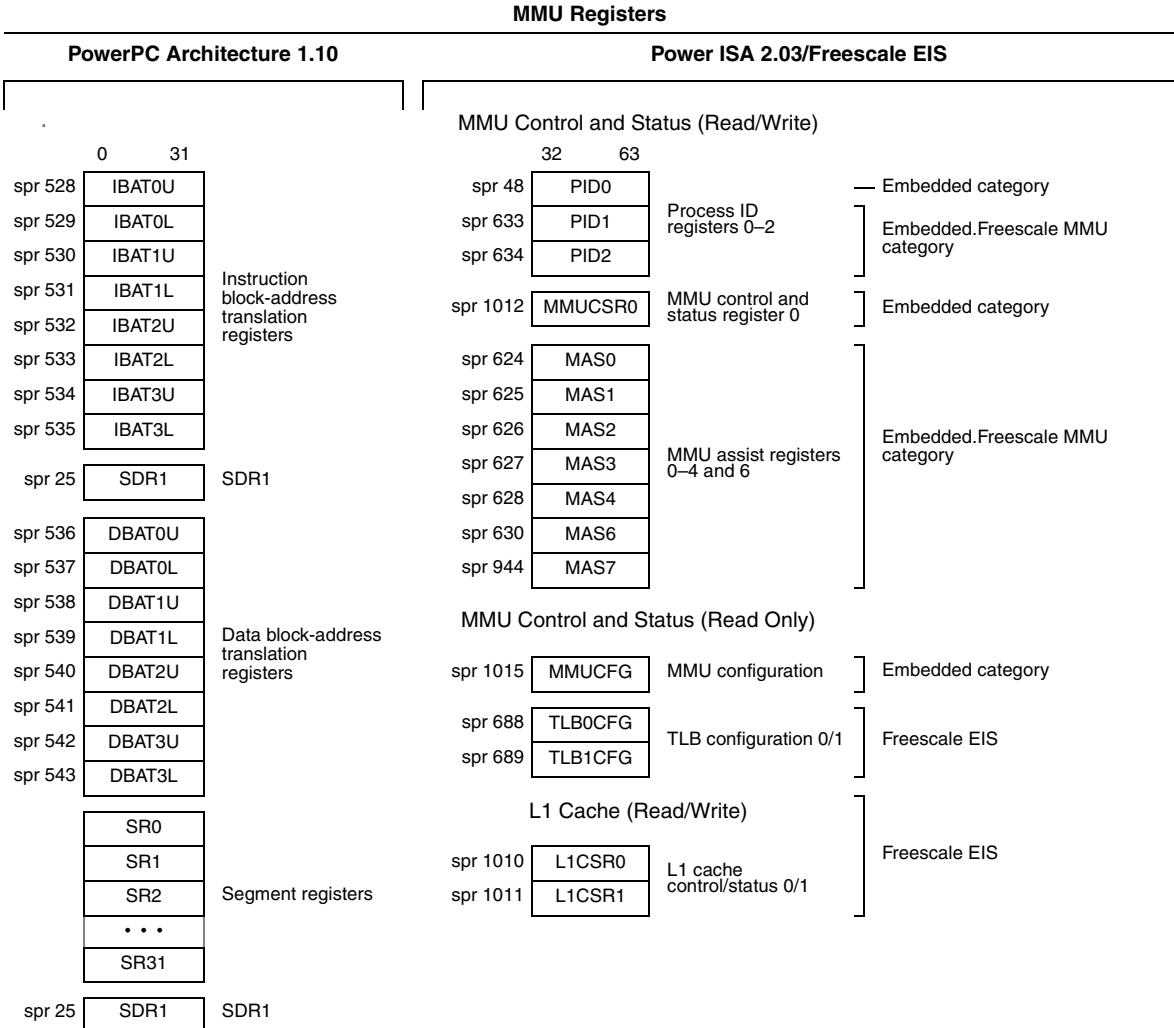


Figure 7. MMU Registers Comparison

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 43.

- 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 track effective address generation:

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

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

- Process ID registers (PID1–PID2). PID values are used to construct virtual addresses for accessing memory.
- MMU control and status register 0 (MMUCSR0). Used for general MMU control, for example, to flash invalidate TLBs of the caches.
- MMU assist registers (MAS0–MAS6). 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

The Freescale EIS defines L1 cache configuration and status registers, shown in [Figure 8](#). Neither the PowerPC architecture 1.10 nor the Power Architecture 2.03 specification defines L1 cache registers.

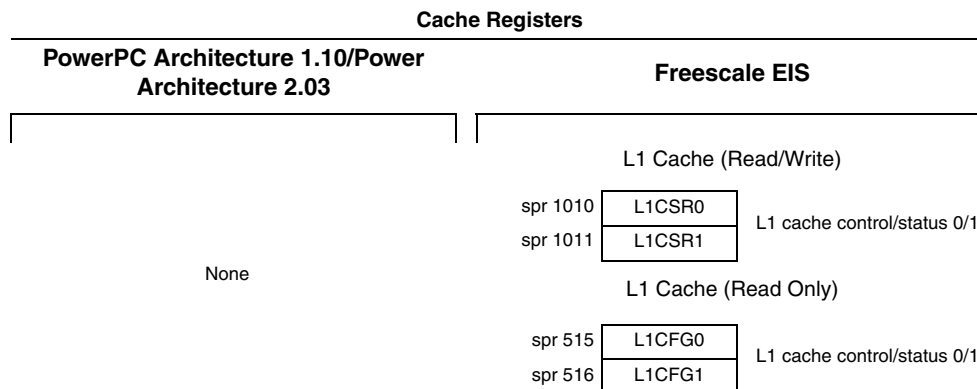


Figure 8. Cache Registers Comparison

The registers in [Figure 8](#) 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 9 compares the PowerPC architecture 1.10 with the embedded category interrupt registers.

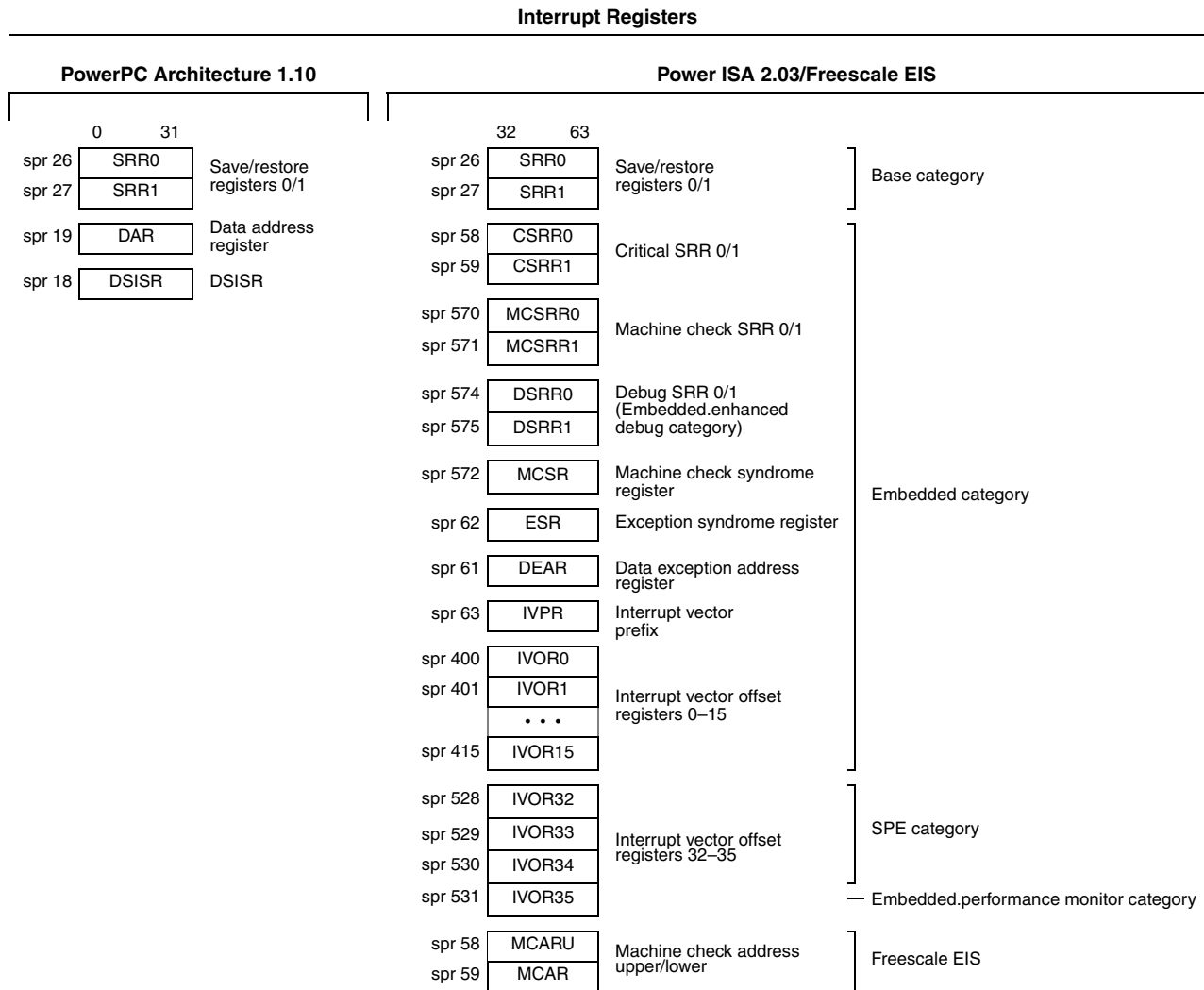


Figure 9. Interrupt Register Comparison

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 16-bit vector offsets are concatenated to the right of IVPR to form the address of the interrupt routine.
- Interrupt vector offset registers (IVOR0–IVOR15). 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 additional IVORs to support implementation-specific interrupts. For example, the SPE defines IVOR32–IVOR35. Such IVORs are listed at the bottom of [Table 10](#).

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 10](#) shows a comparison of PowerPC architecture 1.10 and the PowerPC architecture 2.03/EIS configuration registers.

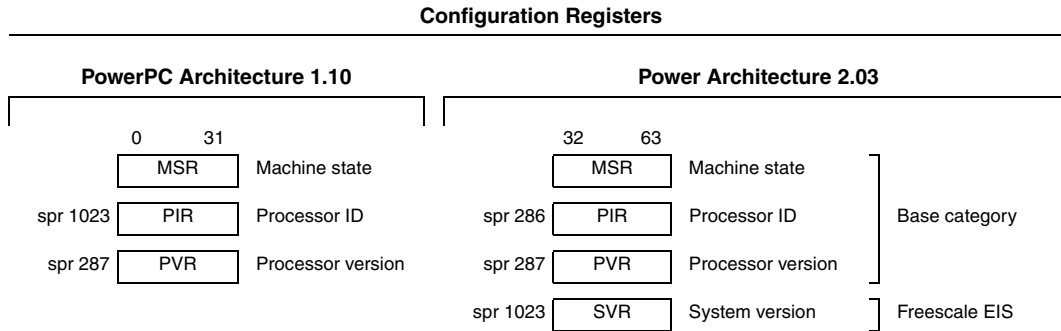


Figure 10. Configuration Registers Comparison

PowerPC architecture 1.10 and the Power Architecture 2.03 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 do not support modal big/little endian byte ordering or real mode (IR=0 and DR=0).

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 Book E 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 11](#) are used exclusively by the performance monitor category. PMRs are similar to the SPRs and are accessed by **mtpmr** and **mfpmr** instructions, which are also part of the performance monitor category. User-level PMRs are 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.

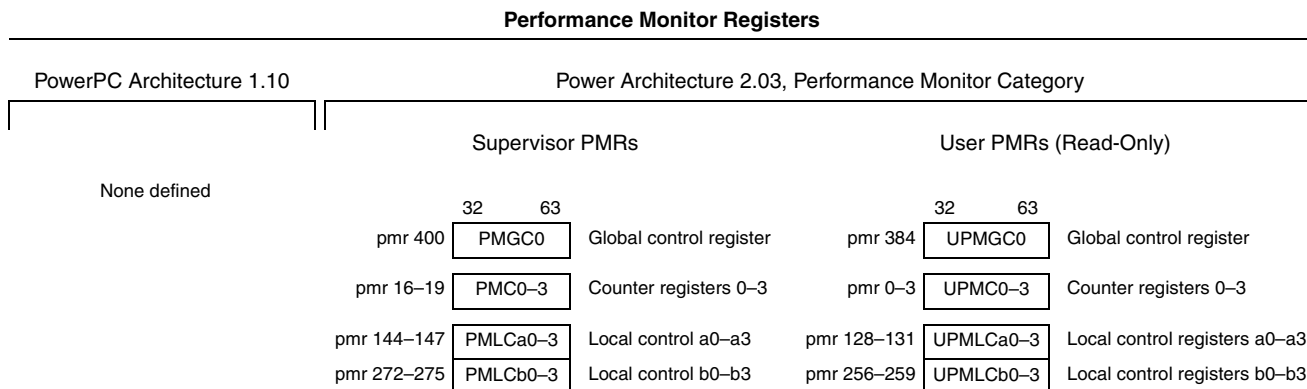


Figure 11. Performance Monitor Registers Comparison

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 12 shows a comparison of PowerPC architecture 1.10 and the Power ISA debug registers.

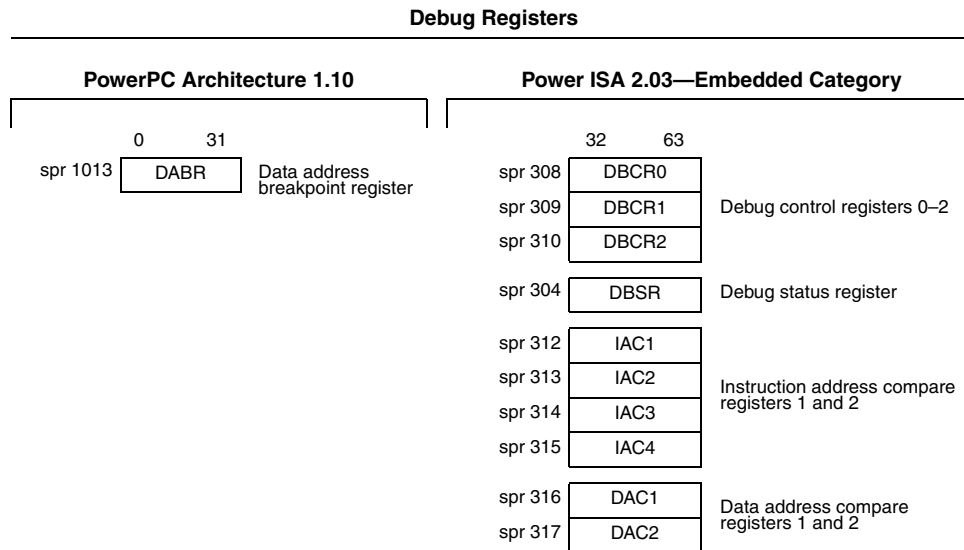


Figure 12. Debug Registers Comparison

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 33.

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

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, interrupt handlers must save the information stored in SRR0 and SRR1 (the return address and the MSR settings) soon after the interrupt is taken 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, in PowerPC architecture 1.10, the DSISR and the floating-point status and control register (FPSCR). Additionally, certain exception conditions can be explicitly enabled or disabled by software.

Invocation of an interrupt is precise, unless one of the imprecise modes for invoking the floating-point enabled exception-type program interrupt is in effect. When the interrupt is invoked imprecisely, the excepting instruction does not appear to complete before the next instruction starts (because the invocation of the interrupt required to complete execution has not occurred).

Interrupts, their offsets, and conditions that cause them for the PowerPC architecture 1.10 and the Power Architecture 2.03 specification are summarized in [Table 10](#). 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 SRR0 and SRR1.

- 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 0xFFFFn_nnnn. [Table 10](#) 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 2.03 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 10](#) shows the IVORs associated with each interrupt type.

The save and store resources are part of the base category and are largely identical to those defined by the OEA. Save and restore registers (SRR0/SRR1) save the return address and machine state when they are taken. The **rfi** instruction is used to restore 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 **rftci** instruction to restore state.
The Power ISA version defines the critical input, watchdog timer, debug, and machine-check interrupts as critical interrupts.
 - 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 10 shows a comparison of the PowerPC architecture 1.10 and Power ISA 2.03 interrupt models.

Table 10. Interrupts and Conditions—Overview

Interrupt Type	Vector Offset		Causing Conditions
	PowerPC	Power ISA	
Power ISA 2.03 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	0x200	IVOR1	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 and ME is cleared, the processor goes into checkstop. MSR settings and return address are stored in SRR0 and SRR1. Embedded-specific features: 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	0x300	IVOR2	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. PowerPC architecture 1.10: DSISR reports the cause; DAR is set based on DSISR settings. Embedded-specific features: ESR reports the cause; DEAR holds the effective address of the data access.
ISI	0x400	IVOR3	Instruction fetch cannot be performed. Causes include the following: <ul style="list-style-type: none"> • 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 segment or from guarded memory when MSR[IR] = 1. • The fetch access violates memory protection. Embedded-specific features: Book E also uses 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.
External interrupt	0x500	IVOR4	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).

Table 10. Interrupts and Conditions—Overview (continued)

Interrupt Type	Vector Offset		Causing Conditions
	PowerPC	Power ISA	
Alignment	0x600	IVOR5	<p>The processor cannot perform a memory access for one of these reasons:</p> <ul style="list-style-type: none"> • The operand of a load or store is not aligned. • The instruction is a move assist, load multiple, or store multiple. • A dcbz operand is in write-through-required or caching-inhibited memory, or dcbz is executed in an implementation with no data cache or a write-through data cache. • The operand of a store, except store conditional, is in write-through required memory. <p>PowerPC architecture 1.10: DSISR reports the interrupt cause; DAR is set based on DSISR settings.</p> <p>Embedded-specific features: ESR reports the interrupt cause; DEAR holds the effective address of the data access.</p> <p>EIS-specific features: EIS defines additional exception conditions.</p>
Program	0x700	IVOR6	<p>One of the following conditions occurs during instruction execution:</p> <ul style="list-style-type: none"> • Floating-point enabled exception—Generated when MSR[FE0,FE1] ≠ 00 and FPSCR[FEX] is set. • Illegal instruction—Generated when execution of an instruction is attempted with an illegal opcode or illegal combination of opcode and extended opcode fields, or when execution of an optional instruction not provided in the specific implementation is attempted (these do not include optional instructions treated as no-ops). • Privileged instruction—User-level code attempts execution of a supervisor instruction. • Trap—Any of the conditions specified in a trap instruction is met. <p>PowerPC architecture 1.10: Caused when a floating-point instruction causes an enabled exception or by the execution of a Move to FPSCR instruction that sets both an exception condition bit and its corresponding FPSCR enable bit. DSISR reports the cause of the program interrupt; DAR is set based on DSISR settings.</p> <p>Embedded-specific features: An unimplemented operation exception may occur if an unimplemented defined or allocated instruction is encountered. Otherwise, an illegal instruction interrupt occurs. ESR reports the cause.</p>
Floating-point unavailable	0x800	IVOR7	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	0x900	IVOR10	<p>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.</p> <p>Embedded-specific features: The 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.</p>
System call	0xC00	IVOR8	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).

Table 10. Interrupts and Conditions—Overview (continued)

Interrupt Type	Vector Offset		Causing Conditions
	PowerPC	Power ISA	
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.
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	Reserved for future architectural use.
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, provides the necessary support for the operating system to implement a paged virtual-memory environment and for enforcing protection of designated memory areas. The MSR controls some of the critical functionality of the MMU.

The MMU and exception models support demand-paged virtual memory. 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.

The memory management model includes the concept of a virtual address space that is not only larger than that of the maximum physical memory allowed, but is also larger than the effective address space. Effective addresses are 32 bits wide. In the address-translation process, the processor converts an effective address to a virtual address of up to 52 bits in size.

Two general types of processor-generated memory accesses require address translation: instruction accesses and data accesses generated by load and store instructions. Additionally, the addresses specified by cache instructions also require translation. 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.

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.

MMU Features in the PowerPC Architecture 1.10 Definition

In the PowerPC architecture 1.10 definition, the address-translation mechanism is further defined in terms of segment descriptors. The segment information translates the effective address to an interim virtual address, and, as previously mentioned, the page-table information translates the virtual address to a physical address.

Effective address spaces are divided into 256-Mbyte segments or into other large regions called blocks (128 Kbyte–256 Mbyte). Segments that correspond to memory-mapped areas can be further subdivided into 4-Kbyte pages.

The definition of the segment and page-table data structures provides significant flexibility for the implementation of performance enhancement features in a wide range of processors. Therefore, the performance enhancements used to store the segment or page-table information on-chip vary from implementation to implementation.

The segment information, used to generate the interim virtual addresses, is stored as segment descriptors. These descriptors may reside in on-chip segment registers (32-bit implementations) or as segment table entries (STEs) in memory (64-bit implementations).

The PowerPC architecture 1.10 architecture also defines the block address translation (BAT) mechanism, which is a software-controlled array that stores the available block address translations on-chip. BAT array entries are implemented as pairs of BAT registers that are accessible as supervisor special-purpose registers (SPRs).

For each block or page, the operating system creates an address descriptor (page-table entry (PTE) or BAT-array entry); 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 block or 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 MMU keeps the block information on-chip in the BAT array (comprised of the BAT registers).

MMU Features in the Embedded Category Definition

Book E defined an alternative to the MMU model defined in PowerPC architecture 1.10. Where the PowerPC architecture supports hardware-based page address translation with fixed 4-Kbyte pages, the Book E MMU is strictly software-managed and supports fixed and variable page sizes. The PowerPC architecture also defined block address translation (BAT) SPRs that could provide a single translation for

large blocks of memory space; in embedded-category processors, this is done with variable page sizes. For more information, see “[Interrupt Model](#)” on page 39.

Differences between the PowerPC architecture 1.10 and the Power Architecture 2.03 MMU models are outlined in [Table 11](#).

Table 11. PowerPC Architecture 1.10 and Power Architecture 2.03 MMU Models

PowerPC Architecture 1.10	Power ISA 2.03 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
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 12](#) shows how the Power ISA 2.03 MMU features are defined.

Table 12. 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 .

Table 12. Embedded and Freescale Embedded MMU Categories (continued)

PowerPC ISA Embedded Category	PowerPC ISA Embedded MMU Category
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].
	<p>Additional MMU registers:</p> <ul style="list-style-type: none"> • 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. • TLBnCFG 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

Address Translation Mechanisms

The following types of address translation are supported:

- Page address translation—translates the page frame address for a 4-Kbyte page size. The embedded category provides enhanced page address translation that eliminates the need for block address translation and real addressing mode address translation.
- Block address translation—translates the block number for blocks that range in size from 128 Kbytes to 256 Mbytes (PowerPC architecture 1.10)
- Real addressing mode address translation—when address translation is disabled, the physical address is identical to the effective address. (PowerPC architecture 1.10)

Figure 13 shows the address translation mechanisms provided by the MMU.

The segment descriptors shown in the figure control the page address translation mechanisms in PowerPC processors. When an access uses the page address translation, the appropriate segment descriptor is required. One of the 16 on-chip segment registers (which contain the segment descriptors) is selected by the highest-order effective address bits.

In PowerPC processors, for memory accesses translated by a segment descriptor, the interim virtual address is generated using the information in the segment descriptor. In Freescale embedded-category processors, the virtual address is determined by concatenating the address space bit and the Process ID (PID) to the effective address.

In both PowerPC architecture 1.10 and Power Architecture 2.03 processors, page address translation corresponds to the conversion of this virtual address into the 32-bit physical address used by the memory subsystem. In some cases, the physical address for the page resides in an on-chip TLB and is available for quick access. However, if the page address translation misses in a TLB, the MMU searches the page table in memory (using the virtual address information and a hashing function) to locate the required physical address. Some PowerPC implementations may have dedicated hardware to perform the page table search

automatically, while all embedded category processors and some PowerPC implementations, such as the e300 cores, perform the page table searches with software.

Because blocks are larger than pages, there are fewer upper-order effective address bits to be translated into physical address bits (more low-order address bits, at least 17, are untranslated to form the offset into a block) for block address translation. Also, instead of segment descriptors and a page table, block address translations use the on-chip BAT registers as a BAT array. If an effective address matches the corresponding field of a BAT register, the information in the BAT register is used to generate the physical address; in this case, the results of the page translation (occurring in parallel) are ignored. Note that a matching BAT array entry takes precedence over a translation provided by the segment descriptor in all cases.

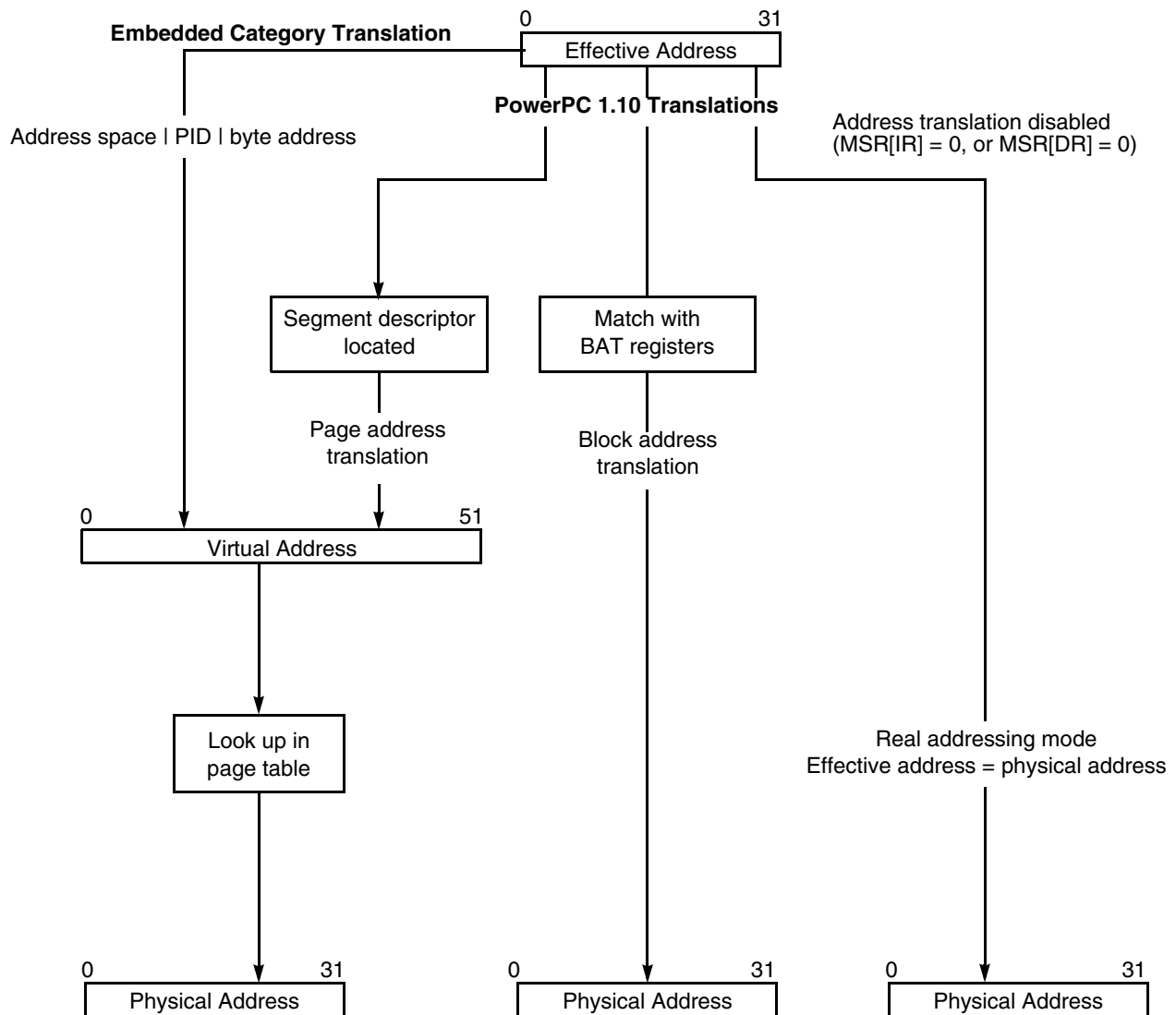


Figure 13. Address Translation Types

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 13](#).

Table 13. Further Differentiation of the Book E Critical Interrupt Model

Interrupt	Book E/EIS		Power ISA 2.03
	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

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 a 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.



Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.
© Freescale Semiconductor, Inc. 2006.

Document Number: PWRARCPMRM
Rev 0

