Freescale Semiconductor
Application Note

Document Number: AN3531
Rev. 0, 10/2007

Migrating from e600- to e500-Based Integrated Devices

by Jerry Young
Networking and Multimedia Group
Freescale Semiconductor, Inc.
Austin, TX

This application note outlines general, high-level, architectural differences between the e600 and e500 family processors. It does not address microarchitectural differences, such as the sizes of caches, bus architecture, or instruction pipeline design. It does not attempt to describe all differences among the members of these two families of processors, but is intended as a general guideline for programmers and system designers who are assessing the efforts required in migrating to e500-based devices.

For specific details, refer to the user documents for the respective devices.

Contents

1 The e600, the e500, and Changes to the Architecture Definition

This section describes the changes to the architectural specifications especially as they relate to the e600 and e500 processor families.

NOTE

The term 'PowerPC™ architecture' has come to refer strictly to the original architecture definition for desktop processors that is implemented on the e300, e600, and others and sometimes is referred to as the classic or AIM (Apple, IBM, Motorola) version of the architecture.

The term 'Power ISA™' refers to the current architecture specification that is implemented on e500 cores.

Both the PowerPC architecture and the Power ISA are part of the more general Power Architecture™ model, as described below.

Many of the differences between the e600 and e500 processor families exist because they were designed to somewhat different versions of the PowerPC architecture, as follows:

- The e600 family was designed to the original PowerPC architecture definition. The functionality of the e600 family cores is described in the following Freescale documents:
  - The *e600 Power Architecture™ Core Family Reference Manual*, which describes functionality specific to the e600.
  - The *Programming Environments Manual for 32-Bit Implementations of the PowerPC™ architecture* (referred to as the PEM), which describes the functionality common to all PowerPC devices.
- The e500v1 and e500v2 processors are designed to what was originally the PowerPC Book E architecture and Freescale's embedded implementation standards (EIS). Together, they replaced many of the original architecture's desktop-centered features (most notably, operating system-level features such as the MMU and interrupt models, as well as true little-endian as part of a storage model in which byte ordering is configured on a per-page basis) with features more suited to the embedded environment for which Book E was intended. The functionality of the e500 family cores is described in the following Freescale documents:
  - The *e500 Power Architecture™ Core Family Reference Manual*, which describes functionality specific to the e500 cores.
  - The *EREF: a Programmer's Reference Manual for Freescale Embedded Devices*, which describes the functionality common to all Freescale Power ISA embedded devices.

  Note that in this document, references to the e500 refer to all e500 devices. Any device-specific differences are noted.

NOTE

For any of the cores referenced in this document, resources that may be defined at the processor level may not be fully implemented, or implemented at all, in the system-on-a-chip (SoC) device that integrates the core. Also, in some cases, and in particular with register fields, functionality may be defined at a general level by the architecture and core reference manuals, and more specifically by the SoC. Because of such differences, it is important to consult the core register summary chapter in the reference manual for the integrated device.

Both families include many extensions to the architecture versions to which they were designed, such as the performance monitor, cache management features, and the signal processing engine (SPE). The SPE, implemented on e500v1 and e500v2, defines an extensive set of 64-bit, two-element vector instructions and includes a set of floating-point instructions as an alternative to the one defined by the PowerPC architecture. To facilitate such special-purpose extensions, Book E introduced the concept of auxiliary processing units (APUs) and allocated resources such as instruction opcode space and SPRs that encouraged the development of such functionality.

Since the restructuring of the architecture (now referred to collectively as the Power Architecture model), most of those APUs are now a formal part of the portion of the architecture designated for embedded devices and published in the Power ISA specification, released in 2006. Figure 1 shows the relationship between the different environments. Note that the e600 family is part of the Power Architecture model; the e500 family is part of the embedded environment of the Power ISA.

It is especially important to note that, although the structure of the architecture has changed considerably, most of the functionality changes have been relegated to operating system–level features (such as the MMU and interrupt models described above). As Figure 1 illustrates, the application-level programming model, that is the base set of instructions and registers, remains consistent across the e600, e500, and all other Power Architecture devices.

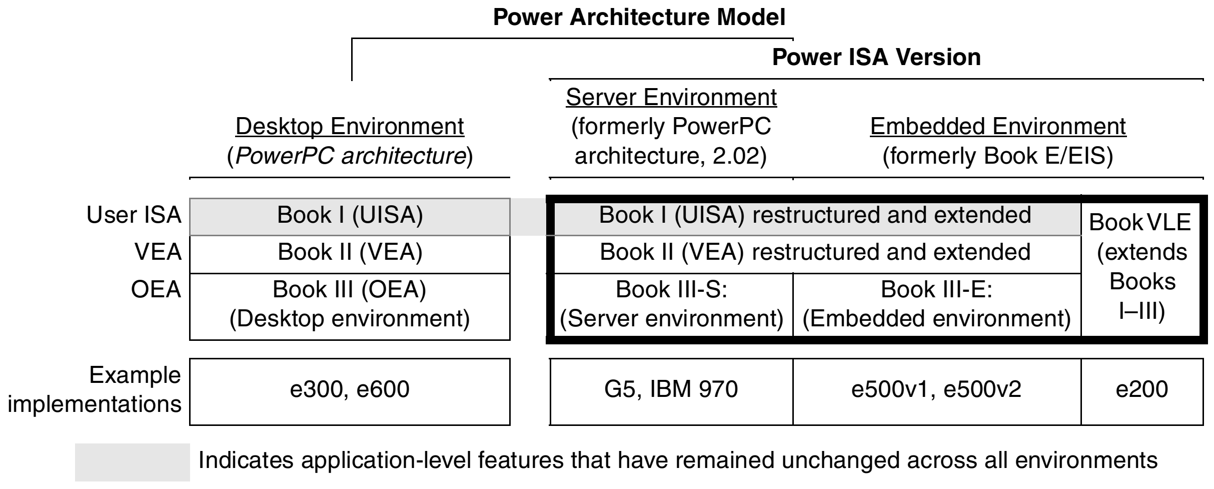


Figure 1. Power Architecture Relationships

The Power ISA extends the modularity of the layered architecture (Books I through III) by breaking the functionality of the architecture into components called ‘categories,’ the broadest of which define basic functionality common across computing environments, as follows:

- The base category defines all of those elements common to all Power Architecture processors. Although it includes functionality defined in all three books, the Base category preserves almost all of the user application-level resources defined in the original PowerPC Book I, the user instruction set architecture (UISA). Other features from the original UISA, such as the floating-point and move assist instructions, are treated as separate categories that are not required for every implementation.
- The embedded and server categories define mutually exclusive resources appropriate for those environments. The e500 family devices implement embedded category resources.

Other categories address more specific features, such as the signal processing engine. Some of these special features were optional in the PowerPC architecture. Others were previously defined as auxiliary processing units (APUs) and were not part of the architecture. Many of those former APUs, began life as part of Freescale’s embedded implementation standards (EIS), a layer of architecture for features common to Freescale processors, but outside of the formal architecture specification. The EIS continues to define such features.

2 Differences between e600 and e500 Cores: Overview

This section provides an overview of differences between the e600 and e500 families and summarizes functionality specific to each; more detailed information about the instruction, register, interrupt, and MMU models is provided in the subsequent sections.

2.1 The Floating-Point Model and Signal Processing Engine (SPE)

The e600 implements the floating-point instruction model defined by the PowerPC architecture, and included as a distinct category the Power ISA. This floating-point model includes a separate register file of 32, 64-bit floating-point registers (FPRs) and a full suite of floating-point computational and load/store instructions that support both single- and double-precision operations. The floating-point status and control register (FPSCR) and condition register (CR) resources enable and track exception conditions.

The e500v1 and e500v2 implement the signal processing engine (SPE), a comprehensive set of 64-bit, two-element, SIMD instructions that share the UISA-defined GPRs extended by the SPE to 64 bits, as shown in Figure 2.

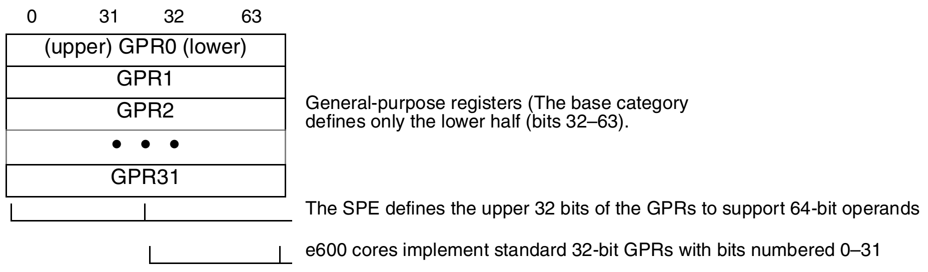

0
31
32
63
(upper) GPR0 (lower)
GPR1
GPR2
• • •
GPR31
General-purpose registers (The base category defines only the lower half (bits 32–63).
The SPE defines the upper 32 bits of the GPRs to support 64-bit operands
e600 cores implement standard 32-bit GPRs with bits numbered 0–31

Figure 2. Extended GPRs

The SPE definition includes three dependent embedded floating-point categories:

- Embedded scalar, single-precision (e500v1/e500v2)
- Embedded scalar, double-precision (e500v2)
- Embedded vector, single-precision (e500v1/e500v2)

For systems that do not require high-end graphics and other floating-point intensive applications, providing a floating-point instruction set that shares the integer-based GPRs rather than requiring the implementation of FPRs simplifies the design of the processor.

2.2 e500-Only Features

The e500 implements the following features, not provided on the e600 and defined by the Power ISA:

- Multiple-level interrupt model. In addition to the standard set of save restore registers (SRR0 and SRR1) and the Return from Interrupt instruction (**rfi**), the Power ISA defines the following separate resources to shorten interrupt latency and provide greater control over interrupt behavior:
  - — Critical interrupts—Uses 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 following interrupts use the critical interrupt resources: critical input and watchdog timer interrupts. On the e500, machine check interrupts may be configured to use critical interrupt resources.
  - — Machine check interrupt—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.
- Programmable interrupt vectors. The Power ISA defines the following SPRs for setting up the interrupt vector table:
  - — Interrupt vector prefix register (IVPR). Provides the high-order bits for placing the interrupt table in memory.
  - — Interrupt vector offset registers (IVORs). Provides the low-order, interrupt-specific bits for placing each interrupt handler into the interrupt table.
- Byte ordering configured on a per-page basis (the E bit in the TLBs) instead of the moded byte ordering determined by the setting of MSR[LE,ILE]. These bits are not implemented on the e500.

  The Power ISA defines true little-endian byte ordering, replacing the version of little-endian byte ordering defined in the PowerPC architecture.
- Cache-line 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 e600 cache locking functionality allows separate locking of the data and instruction cache by setting HID0[DLOCK,ILOCK].
- The e600 implements an L2 cache, which is not supported on the e500v1 or e500v2. Therefore the e600 registers shown in Figure 7 are not supported on the e500v1 and e500v2.

- Freescale MMU. The embedded MMU model defines page-based, software-managed address translation and memory protection using translation lookaside buffers (TLBs). It consists primarily of the storage architecture defined by Book E and the Freescale EIS.

  The MMU model defines the following used to configure and update the TLBs:
  - Machine state register (MSR) fields. MSR[DS] and MSR[IS] are defined as part of the address translation to designate address spaces for data and instruction storage. These bits replace MSR[DR] and MSR[IR], which the PowerPC architecture defined to enable memory translation. Note that translation is always enabled on Power ISA devices.

    Unlike the PowerPC model implemented on the e600, there is no support for real mode; that is, translation is always enabled.
  - MMU assist registers:
    - e500v1: MAS0–MAS4 and MAS6
    - e500v2: MAS0–MAS4 and MAS6–MAS7
  - Process identification registers PID*n*.
  - The TLB configuration registers, TLB0CFG–TLB3CFG
  - The MMU control and status register, MMUCSR0
  - The MMU configuration register MMUCFG
- Expanded hardware and software debug functions. These include instruction and data breakpoints and program single stepping. The debug facilities include debug control registers (DBCR0–DBCR2) and address compare registers (IACs and DACs) for enabling and recording various kinds of debug events and registers that support the debug interrupt-type (DSRR0 and DSRR1).
- Alternate time base. An additional time base analogous to the standard time base defined by both the Power ISA and the PowerPC architecture (Book II). The alternate time base is implemented on the e500v2.
- Additional software-use SPRs. In the Power ISA, the base category defines SPRG0–SPRG3; the embedded category defines SPRG4–SPRG7. The PowerPC architecture defines SPRG0–SPRG3; e600 implements SPRG4–SPRG7, as described in Section 5.10, “Software-Use SPR Comparison.”

3 Power Architecture Details

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

The 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.

3.1 An Overview of Categories Implemented by the e500

This section provides an overview of the categories defined by the Power ISA and implemented on the e500.

All devices implement the facilities defined by the base category. This largest category encompasses all components common across the computing environments; for example, these include the integer computational and load/store instructions and the GPRs. These include devices such as the e600 cores based on the PowerPC architecture. Although the base category largely consists of the features defined in Book I (the user ISA), like many categories, it extends beyond Book I to include those Book II (VEA) and Book III (OEA) features common to all Power Architecture devices, such as the machine state register (MSR), the time base, the interrupt model’s save and restore registers, and the instructions required for accessing them.

The Power ISA floating-point category consists of the resources originally defined by the PowerPC architecture to support single- and double-precision floating-point instructions. The e500v1 and e500v2 do not implement this floating-point model. The functionality of these resources has not changed. Defining them as a separate category underscores the advantages of a modular architecture, providing greater leeway in balancing power, thermal, size, and price constraints for very specific environments.

The Integer Select instruction (**isel**), formerly a Freescale EIS instruction, is now part of the base category. This instruction can be used to more efficiently handle sequences with multiple conditional branches, and is not implemented on the e600.

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

3.1.1 The Embedded Category

As described above, the embedded category largely consists of features formerly defined by the PowerPC Book E architecture and the Freescale EIS. This section describes the components as defined by the Power ISA. Note that the high level embedded category incorporates some resources defined in Book E, including the following:

- Write MSR External Enable instructions (**wrtee**[**i**]), which is implemented on the e500 to update only MSR[EE].
- The software-use SPRs (SPRG4–SPRG9), which are implemented on the e600 as implementation-specific features.

3.1.2 Signal Processing Engine (SPE)

The SPE, implemented on the e500v1 and e500v2, is a 64-bit, two-element, single-instruction multiple-data (SIMD) ISA, originally designed to accelerate signal processing applications normally suited for digital signal processing (DSP) operations. The two-element vectors fit within the GPRs, which the SPE extends to 64 bits. SPE also defines an accumulator register (ACC) to allow for back-to-back operations without loop unrolling. The SPE is primarily an extension of Book I but identifies some resources for interrupt handling in Book III-E.

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

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

Table 1. SPE Vector Multiply Instruction Mnemonic Structure

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

[1] Low word versions of signed saturate and signed modulo fractional instructions are not supported.

3.1.2.1 SPE Embedded Vector and Scalar Floating-Point Categories

The embedded floating-point categories are dependent categories of the SPE. These include the following:

- Single-precision scalar
- Single-precision vector
- Double-precision scalar

The embedded floating-point categories, compatible with IEEE Std. 754™, provide floating-point operations to power- and space-sensitive embedded applications. As is true for all Signal Processing Engine categories, rather than implementing the FPRs defined by the PowerPC architecture, these categories share the GPRs used for integer operations, extending them to 64 bits to accommodate vector single-precision and scalar double-precision categories. These extended GPRs are described in Section 5.1, “Register File Comparison.”

4 Instruction Model

This section describes the instructions and instruction classes as they are defined as part of the Power ISA definition. Features defined only for the PowerPC architecture are indicated as such.

The following instructions are implemented on both the e600 and e500 cores with minimal differences:

- Integer instructions—These include arithmetic, logical, and integer load/store instructions. See Section 4.2.1, “Integer Instructions.” The Power ISA defines and the e500 implements the Integer Select instruction (**isel**), which is neither provided by the earlier PowerPC architecture nor is implemented on the e600.

- Branch and flow control instructions—These include branching instructions, CR logical instructions, trap instructions, and other instructions that affect instruction flow. See Section 4.2.3, "Branch and Flow Control Instructions."

The e500 does not implement the following instructions implemented on the e600:

- AltiVec ISA. The e500 implements the SPE, which defines 64-bit SIMD (single-instruction, multiple data) instructions that operate on integer, fractional, fixed-point, and floating-point operands. The SPE extends the GPRs to 64 bits. See Section 3.1.2, "Signal Processing Engine (SPE)."
- The e500 does not implement the data streaming instructions that are defined as part of the AltiVec ISA.
- Load TLB entry instructions (**tlbld** and **tlbli**) to directly access TLBs. The e500 uses these instructions to directly configure TLBs with translation and memory protection information by loading and storing values defined in the memory assist (MAS) registers. Additional instructions are provided for searching and invalidating entries and for synchronizing TLB accesses.

The following groups of instructions are implemented on both e600 and e500 family devices, but with some differences:

- Floating-point instructions—The e600 family implements the base category floating-point instructions defined by the PowerPC architecture; the e500 family implements floating-point vector and scalar single-precision instructions defined as part of the SPE; the e500v2 implements the embedded double-precision instructions. See Section 4.2.2, "Floating-Point Instructions (e600)," and Section 2.1, "The Floating-Point Model and Signal Processing Engine (SPE)."
- Processor control instructions—These instructions, described in Section 4.3, "Processor Control Instructions," include the instructions that explicitly access registers such as SPRs, MSR, CR, and others. To reduce interrupt latency, the e500 implements the Write MSR External Enable instructions (**wrtee**[**i**]), which can be used instead of **mtmsr** to update only MSR[EE], which enables or disables external interrupt exception conditions. The **wrtee** instruction has fewer serialization requirements, and therefore shorter latency, than **mtmsr**.
- Memory synchronization instructions—These instructions, described in Section 4.3.1, "Memory Synchronization Instructions," 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.
  - Book E recast the PowerPC architecture–defined **sync** as **msync**. However, the Power ISA version defines **msync** as a simplified mnemonic for the **sync** instruction, configured to function as the Book E–defined **msync** for embedded category devices.
  - The **eieio** instruction, Enforce In-Order Execution of I/O, which is defined by the PowerPC architecture and implemented on the e600, shares the same opcode with the **mbar** (Memory Barrier) instruction defined by the Power ISA embedded category.

    Because **eieio** and **mbar** share the same opcode, software designed for both environments must assume that only the **eieio** functionality applies, because the functions provided by **eieio** are a subset of those provided by **mbar**. Refer to the EREF and PEM for details.
- Memory control instructions—These instructions provide control of caches and TLBs. See Section 4.3.2, "Memory Control Instructions."

The standard UISA floating-point instructions use FPRs for single- and double-precision floating-point operands. The SPE embedded floating point instructions, implemented on the e500v1 and e500v2, use GPRs widened to 64 bits to support vector single-precision and scalar double-precision operands.

4.1 Simplified Mnemonics

The simplified mnemonics for instructions common to both versions of the architecture are consistent in all implementations. Note that the Power ISA defines simplified mnemonics for some new instructions.

Also the **msync** instruction in the e500 is a simplified mnemonic for **sync** instruction. See Table 8.

Additional simplified mnemonics are provided to support access to both newly architected and implementation-specific SPRs.

4.2 Instruction Set Overview

The tables in this section provide a general overview of the e500 instruction set, indicating those instructions that either are not supported by the e600 or whose implementations have changed.

4.2.1 Integer Instructions

This section describes the integer instructions, all of which are defined in Book I. All are part of the base category except for the load/store string and multiple instructions.

These integer instructions are grouped as follows:

- Integer arithmetic instructions
- Integer compare instructions
- Integer logical instructions
- Integer rotate and shift instructions
- Integer select instruction (new in the Power ISA and implemented on the e500)

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	Function	Options
Integer arithmetic (**add***x*, **div***x*, **mul***x*, **neg***x*, **sub***x*)	Add, divide, multiply, negate, subtract	Unchanged
Integer compare (**cmp***x*)	Compare	Unchanged
Integer logical (**and***x*, **cnt**, **eqv**, **ext***x*, **nand**, **nor***x*, **or***x*, **xor***x*)	AND, count, equivalent, extend, NAND, NOR, OR, XOR	Unchanged
Integer rotate and shift (**rlw***x*, **slw***x*, **srw***x*, **sraw***x*)	Rotate left word, shift	Unchanged
Integer select (**isel**)	Integer select	Defined by the Power ISA, implemented by the e500 but not the e600.

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

Table 3. Integer Load/Store Instructions

Instruction	Function	Comments
Integer load (**lb***x*, **lh***x*, **lw***x*)	Load byte, word, half word, algebraic (half word), byte reverse, and zero, with update, indexed.	**Note:** The SPE defines instructions for loading and storing double-word operands required for SPE vector instructions and embedded floating-point single-precision vector and double-precision scalar instructions.
Integer load multiple/string word: **lmw**, **lswi**	Load multiple word	Base category. Implemented on both e600 and e500 cores.
	Load string word	Move assist category. Implemented on the e600; not implemented on the e500.
Integer store (**stb***x*, **sth***x*, **stw***x*	Store	**Note:** Byte, word, half word, byte-reverse, with update, indexed. The SPE defines instructions for loading and storing double-word operands required for SPE vector instructions and embedded floating-point single-precision vector and double-precision scalar instructions.
Integer store multiple/string word: **stmw**, **stswi**	Store multiple word	Base category. Implemented on both e600 and e500 cores.
	Store string word	Move assist category. Implemented on the e600; not implemented on the e500.

4.2.2 Floating-Point Instructions (e600)

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

The signal processing engine (SPE), implemented on the e500v1 and e500v2, defines an alternative floating-point instruction set that uses GPRs rather than FPRs. See Section 3.1.2.1, "SPE Embedded Vector and Scalar Floating-Point Categories."

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

Table 4. Floating-Point Computational Instructions

Instructions	Instruction Name	Comments
Floating-point elementary arithmetic (**fadd***x*, **fdiv***x*, **fmul***x*, **fsub***x*, **fsqrt***x*, **fres***x*, **fabs**, **fmr**, **fnabs**, **fneg**)	Add, divide, multiply, reciprocal, square root, subtract, absolute value, move register, negative absolute value, negate	Not on e500v1/e500v2
Floating-point multiply-add (**fmadd***x*)	Multiply-add, multiply-subtract, negative multiply-add, negative multiply-subtract	
Floating-point rounding and conversion (**fcti***x*, **fr***x*)	Convert to/from integer, round to single-precision	
Floating-point compare and select (**fcm***x*)	Compare, select	
FPSCR (**mtf**x, **mff***x*)	Move to/from FPSCR	

Table 5 shows that 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	Comments
Floating-point load (**lf***x*)	Load floating-point	Not on e500v1/e500v2
Floating-point store (**stf***x*)	Store floating-point	

4.2.3 Branch and Flow Control Instructions

Branch instruction functions include the following:

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

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

All processors support simplified mnemonics that allow conditions specified by BO and BI to be incorporated into the mnemonic. For example, the Branch Conditional instruction, **bc** BO**,**BI, *target address*, can be coded to decrement the count register (CTR) and branch as long as the CTR is not zero (closure of a loop controlled by a count loaded into CTR). To specify this condition, the BO field must be coded as 16. Alternatively, a simplified mnemonic is available, **bdnz,** that indicates “branch while the decremented value is non-zero.” Using the simplified mnemonic eliminates the BO and BI operands, simplifying ‘**bc** 16,0,target’ to the more easily remembered ‘**bdnz** *target*’, which generates identical machine code.

The supervisor-level **rfi** instruction is used for returning from a standard interrupt handler.

The differences between the processor families are as follows:

- The **rfci** instruction is part of the embedded category and is used for critical interrupts on e500 cores.
- The e500 implements the Power ISA–defined **rfmci** for machine check interrupts. See Section 6, “Interrupt Model.”

Branch and flow control instructions are shown in Table 6.

Table 6. Branch and Flow Control Instructions

Instruction	Name	Comments
Branch (**b***x*, **bc***x*)	Branch	Unchanged
CR logical (**cr***x*, **mcr***x*)	Condition register	Unchanged
Trap (**t***x*, **tw***x*)	Trap	Unchanged
System call (**sc**)	System call	Unchanged
Return (**rf***x*)	Return from	Interrupt, critical, and machine check interrupts.

4.3 Processor Control Instructions

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

Differences between implementations are as follows:

- The e500 implements the Power ISA–defined Write MSR External Enable instructions (**wrtee**[**i**]), which updates only MSR[EE] with fewer serialization requirements, and therefore shorter latency, than **mtmsr**.

Table 7 summarizes processor control instructions.

Table 7. Processor Control Instructions

Instructions	Name	Comments
Move (**mt***x*, **mf***x*)	Move to SPR, CR fields, CR from XER, time base, MSR, PMR	**Note:** All devices support simplified mnemonics formed by adding the abbreviated name of any SPR to the prefix '**mf**', for example **mfmas0**, **mfivor3**, and **mfcssr1**.
	Move from SPR, CR fields, CR from XER, time base, MSR, PMR.	

4.3.1 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. Differences between processors are highlighted in Table 8.

Table 8. Memory Synchronization Instructions

Instructions	Name	Comments
lwarx	Load word and reserve index	Unchanged
stwcx.	Store word conditional index	Unchanged

Table 8. Memory Synchronization Instructions (continued)

Instructions	Name	Comments
Synchronize (**sync**, **eieio**, **isync**, **msync**, **mbar**)	Memory Synchronize	Book E recast PowerPC architecture–defined **sync** as **msync**. Power ISA defines **msync** as a simplified mnemonic, configured to function as the Book E–defined **msync**.
	Enforce In-Order Execution of I/O (e600)/Memory Barrier (e500)	PowerPC architecture–defined (e600)
		Embedded category **mbar** instruction implemented on the e500. The PowerPC architecture defines this opcode as **eieio**.
	Instruction Synchronize	**isync** synchronizes the instruction stream

4.3.2 Memory Control Instructions

Memory control instructions include instructions for cache management and TLB management. Major differences are as follows:

- The segment register instructions defined by the PowerPC architecture to support the segmented MMU model and implemented on the e600 are not part of the embedded environment and are not implemented on the e500.
- TLB management instructions—Resources defined to support software address translation.

  The e600 defines the Load Data TLB Entry (**tlbld**) and Load Instruction TLB Entry (**tlbli**) instructions to directly access TLBs. The Power ISA defines **tlbwe** and **tlbre**, which the e500 uses to directly configure TLBs with translation and memory protection information by loading and storing values defined in the memory assist (MAS) registers. Additional instructions are provided for searching and invalidating entries and for synchronizing TLB accesses.

Specific differences in these instruction sets are listed in Table 9.

Table 9. Memory Control Instructions

Instructions	Name	Comments
User-level cache (**dcb***x*, **icb***x*)	Data cache block touch, touch for store, allocate, clear, zero, store, flush.	The Power ISA defines additional cache lock instructions, **icblc** and **dcblc,** implemented on the e500.
	Instruction cache block invalidate, touch	The embedded category defines additional cache touch instructions implemented on the e500: **icbtls**, **dcbtls**, and **dcbtstls**.
TLB management (**tlb***x*)	TLB invalidate, synchronize	Unchanged
	TLB read entry	e500. Reads TLB parameters from the TLBs to the MAS registers.
	TLB search indexed	e500. Searches valid TLB arrays for an entry corresponding to the virtual address and reads appropriate values into the MAS registers.
	TLB write entry	e500. Writes TLB parameters from the MAS registers to the TLBs.

4.3.3 Instruction Set Differences

Table 10 lists the instructions implemented in the e600 and e500 processors, where applicable, noting the architecture that defines the instruction.

Table 10. List of Instructions

Mnemonic	PowerPC	Power ISA	e600	e500
addc[o][.]	√	√	√	√
adde[o][.]	√	√	√	√
addi	√	√	√	√
addic[.]	√	√	√	√
addis	√	√	√	√
addme[o][.]	√	√	√	√
addze[o][.]	√	√	√	√
add[o].]	√	√	√	√
andc[.]	√	√	√	√
andi.	√	√	√	√
andis.	√	√	√	√
and[.]	√	√	√	√
b	√	√	√	√
ba	√	√	√	√
bbelr	—	—	—	√
bblels	—	—	—	√
bc	√	√	√	√
bca	√	√	√	√
bcctr	√	√	√	√
bcctrl	√	√	√	√
bcl	√	√	√	√
bcla	√	√	√	√
bclr	√	√	√	√
bclrl	√	√	√	√
bl	√	√	√	√
bla	√	√	√	√
brinc	—	√	—	v1/v2
cmp	√	√	√	√
cmpi	√	√	√	√
cmpl	√	√	√	√
cmpli	√	√	√	√
cntlzw[.]	√	√	√	√
crand	√	√	√	√
crandc	√	√	√	√
creqv	√	√	√	√
crnand	√	√	√	√
crnor	√	√	√	√
cror	√	√	√	√

Table 10. List of Instructions (continued)

Mnemonic	PowerPC	Power ISA	e600	e500
crorc	√	√	√	√
crxor	√	√	√	√
dcba	√	√	√	√
dcbf	√	√	√	√
dcbi	√	√	√	√
dcblc	—	√	—	√
dcbst	√	√	√	√
dcbt	√	√	√	√
dcbtls	—	√	—	√
dcbtst	√	√	√	√
dcbtstls	—	√	—	√
dcbz	√	√	√	√
divwu[o][.]	√	√	√	√
divw[o][.]	√	√	√	√
dss	—	√	√	—
dssall	—	√	√	—
dst	—	√	√	—
dstst	—	√	√	—
dststt	—	√	√	—
dstt	—	√	√	—
eciwx	√	—	—	—
ecowx	√	—	—	—
efdabs	—	√	—	v2
efdadd	—	√	—	v2
efdcfs	—	√	—	v2
efdcfsf	—	√	—	v2
efdcfsi	—	√	—	v2
efdcfuf	—	√	—	v2
efdcfui	—	√	—	v2
efdcmpeq	—	√	—	v2
efdcmpgt	—	√	—	v2
efdcmplt	—	√	—	v2
efdctsf	—	√	—	v2
efdctsi	—	√	—	v2
efdctsiz	—	√	—	v2
efdctuf	—	√	—	v2
efdctui	—	√	—	v2
efdctuiz	—	√	—	v2
efddiv	—	√	—	v2
efdmul	—	√	—	v2
efdnabs	—	√	—	v2
efdneg	—	√	—	v2

Table 10. List of Instructions (continued)

Mnemonic	PowerPC	Power ISA	e600	e500
efdsub	—	√	—	v2
efdtsteq	—	√	—	v2
efdtstgt	—	√	—	v2
efdtstlt	—	√	—	v2
efsabs	—	√	—	v1/v2
efsadd	—	√	—	v1/v2
efscfsf	—	√	—	v1/v2
efscfsi	—	√	—	v1/v2
efscfuf	—	√	—	v1/v2
efscfui	—	√	—	v1/v2
efscmpeq	—	√	—	v1/v2
efscmpgt	—	√	—	v1/v2
efscmplt	—	√	—	v1/v2
efsctsf	—	√	—	v1/v2
efsctsi	—	√	—	v1/v2
efsctsiz	—	√	—	v1/v2
efsctuf	—	√	—	v1/v2
efsctui	—	√	—	v1/v2
efsctuiz	—	√	—	v1/v2
efsdiv	—	√	—	v1/v2
efsmul	—	√	—	v1/v2
efsnabs	—	√	—	v1/v2
efsneg	—	√	—	v1/v2
efssub	—	√	—	v1/v2
efststeq	—	√	—	v1/v2
efststgt	—	√	—	v1/v2
efststlt	—	√	—	v1/v2
eieio	√	Replaced with **mbar**	√	**mbar**
eqv[.]	√	√	√	√
evabs	—	√	—	v1/v2
evaddiw	—	√	—	v1/v2
evaddsmiaaw	—	√	—	v1/v2
evaddssiaaw	—	√	—	v1/v2
evaddumiaaw	—	√	—	v1/v2
evaddusiaaw	—	√	—	v1/v2
evaddw	—	√	—	v1/v2
evand	—	√	—	v1/v2
evandc	—	√	—	v1/v2
evcmpeq	—	√	—	v1/v2
evcmpgts	—	√	—	v1/v2
evcmpgtu	—	√	—	v1/v2
evcmplts	—	√	—	v1/v2

Table 10. List of Instructions (continued)

Mnemonic	PowerPC	Power ISA	e600	e500
evcmpltu	—	√	—	v1/v2
evcntlsw	—	√	—	v1/v2
evcntlzw	—	√	—	v1/v2
evdivws	—	√	—	v1/v2
evdivwu	—	√	—	v1/v2
eveqv	—	√	—	v1/v2
evextsb	—	√	—	v1/v2
evextsh	—	√	—	v1/v2
evfsabs	—	√	—	v1/v2
evfsadd	—	√	—	v1/v2
evfscfsf	—	√	—	v1/v2
evfscfsi	—	√	—	v1/v2
evfscfuf	—	√	—	v1/v2
evfscfui	—	√	—	v1/v2
evfscmpeq	—	√	—	v1/v2
evfscmpgt	—	√	—	v1/v2
evfscmplt	—	√	—	v1/v2
evfsctsf	—	√	—	v1/v2
evfsctsi	—	√	—	v1/v2
evfsctsiz	—	√	—	v1/v2
evfsctuf	—	√	—	v1/v2
evfsctui	—	√	—	v1/v2
evfsctuiz	—	√	—	v1/v2
evfsdiv	—	√	—	v1/v2
evfsmul	—	√	—	v1/v2
evfsnabs	—	√	—	v1/v2
evfsneg	—	√	—	v1/v2
evfssub	—	√	—	v1/v2
evfststeq	—	√	—	v1/v2
evfststgt	—	√	—	v1/v2
evfststlt	—	√	—	v1/v2
evldd	—	√	—	v1/v2
evlddx	—	√	—	v1/v2
evldh	—	√	—	v1/v2
evldhx	—	√	—	v1/v2
evldw	—	√	—	v1/v2
evldwx	—	√	—	v1/v2
evlhhesplat	—	√	—	v1/v2
evlhhesplatx	—	√	—	v1/v2
evlhhossplat	—	√	—	v1/v2
evlhhossplatx	—	√	—	v1/v2
evlhhousplat	—	√	—	v1/v2

Table 10. List of Instructions (continued)

Mnemonic	PowerPC	Power ISA	e600	e500
evlhhousplatx	—	√	—	v1/v2
evlwhe	—	√	—	v1/v2
evlwhex	—	√	—	v1/v2
evlwhos	—	√	—	v1/v2
evlwhosx	—	√	—	v1/v2
evlwhou	—	√	—	v1/v2
evlwhoux	—	√	—	v1/v2
evlwhsplat	—	√	—	v1/v2
evlwhsplatx	—	√	—	v1/v2
evlwwsplat	—	√	—	v1/v2
evlwwsplatx	—	√	—	v1/v2
evmergehi	—	√	—	v1/v2
evmergehilo	—	√	—	v1/v2
evmergelo	—	√	—	v1/v2
evmergelohi	—	√	—	v1/v2
evmhegsmfaa	—	√	—	v1/v2
evmhegsmfan	—	√	—	v1/v2
evmhegsmiaa	—	√	—	v1/v2
evmhegsmian	—	√	—	v1/v2
evmhegumiaa	—	√	—	v1/v2
evmhegumian	—	√	—	v1/v2
evmhesmf	—	√	—	v1/v2
evmhesmfa	—	√	—	v1/v2
evmhesmfaaw	—	√	—	v1/v2
evmhesmfanw	—	√	—	v1/v2
evmhesmi	—	√	—	v1/v2
evmhesmia	—	√	—	v1/v2
evmhesmiaaw	—	√	—	v1/v2
evmhesmianw	—	√	—	v1/v2
evmhessf	—	√	—	v1/v2
evmhessfa	—	√	—	v1/v2
evmhessfaaw	—	√	—	v1/v2
evmhessfanw	—	√	—	v1/v2
evmhessiaaw	—	√	—	v1/v2
evmhessianw	—	√	—	v1/v2
evmheumi	—	√	—	v1/v2
evmheumia	—	√	—	v1/v2
evmheumiaaw	—	√	—	v1/v2
evmheumianw	—	√	—	v1/v2
evmheusiaaw	—	√	—	v1/v2
evmheusianw	—	√	—	v1/v2
evmhogsmfaa	—	√	—	v1/v2

Table 10. List of Instructions (continued)

Mnemonic	PowerPC	Power ISA	e600	e500
evmhogsmfan	—	√	—	v1/v2
evmhogsmiaa	—	√	—	v1/v2
evmhogsmian	—	√	—	v1/v2
evmhogumiaa	—	√	—	v1/v2
evmhogumian	—	√	—	v1/v2
evmhosmf	—	√	—	v1/v2
evmhosmfa	—	√	—	v1/v2
evmhosmfaaw	—	√	—	v1/v2
evmhosmfanw	—	√	—	v1/v2
evmhosmi	—	√	—	v1/v2
evmhosmia	—	√	—	v1/v2
evmhosmiaaw	—	√	—	v1/v2
evmhosmianw	—	√	—	v1/v2
evmhossf	—	√	—	v1/v2
evmhossfa	—	√	—	v1/v2
evmhossfaaw	—	√	—	v1/v2
evmhossfanw	—	√	—	v1/v2
evmhossiaaw	—	√	—	v1/v2
evmhossianw	—	√	—	v1/v2
evmhoumi	—	√	—	v1/v2
evmhoumia	—	√	—	v1/v2
evmhoumiaaw	—	√	—	v1/v2
evmhoumianw	—	√	—	v1/v2
evmhousiaaw	—	√	—	v1/v2
evmhousianw	—	√	—	v1/v2
evmra	—	√	—	v1/v2
evmwhsmf	—	√	—	v1/v2
evmwhsmfa	—	√	—	v1/v2
evmwhsmi	—	√	—	v1/v2
evmwhsmia	—	√	—	v1/v2
evmwhssf	—	√	—	v1/v2
evmwhssfa	—	√	—	v1/v2
evmwhumi	—	√	—	v1/v2
evmwhumia	—	√	—	v1/v2
evmwlsmiaaw	—	√	—	v1/v2
evmwlsmianw	—	√	—	v1/v2
evmwlssiaaw	—	√	—	v1/v2
evmwlssianw	—	√	—	v1/v2
evmwlumi	—	√	—	v1/v2
evmwlumia	—	√	—	v1/v2
evmwlumiaaw	—	√	—	v1/v2
evmwlumianw	—	√	—	v1/v2

Table 10. List of Instructions (continued)

Mnemonic	PowerPC	Power ISA	e600	e500
evmwlusiaaw	—	√	—	v1/v2
evmwlusianw	—	√	—	v1/v2
evmwsmf	—	√	—	v1/v2
evmwsmfa	—	√	—	v1/v2
evmwsmfaa	—	√	—	v1/v2
evmwsmfan	—	√	—	v1/v2
evmwsmi	—	√	—	v1/v2
evmwsmia	—	√	—	v1/v2
evmwsmiaa	—	√	—	v1/v2
evmwsmian	—	√	—	v1/v2
evmwssf	—	√	—	v1/v2
evmwssfa	—	√	—	v1/v2
evmwssfaa	—	√	—	v1/v2
evmwssfan	—	√	—	v1/v2
evmwumi	—	√	—	v1/v2
evmwumia	—	√	—	v1/v2
evmwumiaa	—	√	—	v1/v2
evmwumian	—	√	—	v1/v2
evnand	—	√	—	v1/v2
evneg	—	√	—	v1/v2
evnor	—	√	—	v1/v2
evor	—	√	—	v1/v2
evorc	—	√	—	v1/v2
evrlw	—	√	—	v1/v2
evrlwi	—	√	—	v1/v2
evrndw	—	√	—	v1/v2
evsel	—	√	—	v1/v2
evslw	—	√	—	v1/v2
evslwi	—	√	—	v1/v2
evsplatfi	—	√	—	v1/v2
evsplati	—	√	—	v1/v2
evsrwis	—	√	—	v1/v2
evsrwiu	—	√	—	v1/v2
evsrws	—	√	—	v1/v2
evsrwu	—	√	—	v1/v2
evstdd	—	√	—	v1/v2
evstddx	—	√	—	v1/v2
evstdh	—	√	—	v1/v2
evstdhx	—	√	—	v1/v2
evstdw	—	√	—	v1/v2
evstdwx	—	√	—	v1/v2
evstwhe	—	√	—	v1/v2

Table 10. List of Instructions (continued)

Mnemonic	PowerPC	Power ISA	e600	e500
evstwhex	—	√	—	v1/v2
evstwho	—	√	—	v1/v2
evstwhox	—	√	—	v1/v2
evstwwex	—	√	—	v1/v2
evstwwex	—	√	—	v1/v2
evstwwo	—	√	—	v1/v2
evstwwox	—	√	—	v1/v2
evsubfsmiaaw	—	√	—	v1/v2
evsubfssiaaw	—	√	—	v1/v2
evsubfumiaaw	—	√	—	v1/v2
evsubfusiaaw	—	√	—	v1/v2
evsubfw	—	√	—	v1/v2
evsubifw	—	√	—	v1/v2
evxor	—	√	—	v1/v2
extsb[.]	√	√	√	√
extsh[.]	√	√	√	√
fabs[.]	√	√	√	—
fadds[.]	√	√	√	—
fadd[.]	√	√	√	—
fcfid[.]	√	√	√	—
fcmpo	√	√	√	—
fcmpu	√	√	√	—
fctidz[.]	√	√	√	—
fctid[.]	√	√	√	—
fctiwz[.]	√	√	√	—
fctiw[.]	√	√	√	—
fdivs[.]	√	√	√	—
fdiv[.]	√	√	√	—
fmadds[.]	√	√	√	—
fmadd[.]	√	√	√	—
fmr[.]	√	√	√	—
fmsubs[.]	√	√	√	—
fmsub[.]	√	√	√	—
fmuls[.]	√	√	√	—
fmul[.]	√	√	√	—
fnabs[.]	√	√	√	—
fneg[.]	√	√	√	—
fnmadds[.]	√	√	√	—
fnmadd[.]	√	√	√	—
fnmsubs[.]	√	√	√	—
fnmsub[.]	√	√	√	—
fres[.]	√	√	√	—

Table 10. List of Instructions (continued)

Mnemonic	PowerPC	Power ISA	e600	e500
frsp[.]	√	√	√	—
frsqrte[.]	√	√	√	—
fsel[.]	√	√	√	—
fsqrts[.]	√	√	—	—
fsqrt[.]	√	√	—	—
fsubs[.]	√	√	√	—
fsub[.]	√	√	√	—
icbi	√	√	√	√
icblc	—	√	—	√
icbt	—	√	√	√
icbtls	—	√	—	√
isel	—	√	—	√
isync	√	√	√	√
lbz	√	√	√	√
lbzu	√	√	√	√
lbzux	√	√	√	√
lbzx	√	√	√	√
lfd	√	√	√	—
lfdepx	—	√	—	—
lfdu	√	√	√	—
lfdux	√	√	√	—
lfdx	√	√	√	—
lfs	√	√	√	—
lfsu	√	√	√	—
lfsux	√	√	√	—
lfsx	√	√	√	—
lha	√	√	√	√
lhau	√	√	√	√
lhaux	√	√	√	√
lhax	√	√	√	√
lhbrx	√	√	√	√
lhz	√	√	√	√
lhzu	√	√	√	√
lhzux	√	√	√	√
lhzx	√	√	√	√
lmw	√	√	√	√
lswi	√	√	√	—
lswx	√	√	√	—
lvebx	—	√	√	—
lvehx	—	√	√	—
lvewx	—	√	√	—
lvsl	—	√	√	—

Table 10. List of Instructions (continued)

Mnemonic	PowerPC	Power ISA	e600	e500
lvsr	—	√	√	—
lvx	—	√	√	—
lvxl	—	√	√	—
lwarx	√	√	√	√
lwbrx	√	√	√	√
lwz	√	√	√	√
lwzu	√	√	√	√
lwzux	√	√	√	√
lwzx	√	√	√	√
mbar	—	√	—	√
mcrf	√	√	√	√
mcrfs	√	√	√	—
mcrxr	√	√	√	√
mfcr	√	√	√	√
mffs[.]	√	√	√	
mfmsr	√	√	√	√
mfpmr	—	√	√	√
mfspr	√	√	√	√
mfsr	√			
mfsrin	√			
mftb	√			
mfvscr	—	√	√	—
msync	see **sync**	√	—	√
mtcrf	√	√	√	√
mtfsb0[.]	√	√	√	—
mtfsb1[.]	√	√	√	—
mtfsfi[.]	√	√	√	—
mtfsf[.]	√	√	√	—
mtmsr	√	√	√	√
mtocrf	—	√	—	
mtpmr	—	√	√	√
mtspr	√	√	√	√
mtsr	√	—	√	—
mtsrin	√	—	√	—
mtvscr	—	√	√	—
mulhwu[.]	√	√	√	√
mulhw[.]	√	√	√	√
mulli	√	√	√	√
mullw[**o**][.]	√	√	√	√
nand[.]	√	√	√	√
neg[**o**][.]	√	√	√	√
nor[.]	√	√	√	√

Table 10. List of Instructions (continued)

Mnemonic	PowerPC	Power ISA	e600	e500
orc[.]	√	√	√	√
ori	√	√	√	√
oris	√	√	√	√
or[.]	√	√	√	√
rfci	—	√	√	√
rfi	√	√	√	√
rfmci	—	√	—	e500
rlwimi[.]	√	√	√	√
rlwinm[.]	√	√	√	√
rlwnm[.]	√	√	√	√
sc	√	√	√	√
slw[.]	√	√	√	√
srawi[.]	√	√	√	√
sraw[.]	√	√	√	√
srw[.]	√	√	√	√
stb	√	√	√	√
stbu	√	√	√	√
stbux	√	√	√	√
stbx	√	√	√	√
stfd	√	√	√	
stfdu	√	√	√	—
stfdux	√	√	√	—
stfdx	√	√	√	—
stfiwx	√	√	√	—
stfs	√	√	√	—
stfsu	√	√	√	—
stfsux	√	√	√	—
stfsx	√	√	√	—
sth	√	√	√	√
sthbrx	√	√	√	√
sthu	√	√	√	√
sthux	√	√	√	√
sthx	√	√	√	√
stmw	√	√	√	√
stswi	√	√	√	—
stswx	√	√	√	—
stvebx	—	√	√	—
stvehx	—	√	√	—
stvewx	—	√	√	—
stvx	—	√	√	—
stvxl	—	√	√	—
stw	√	√	√	√

Table 10. List of Instructions (continued)

Mnemonic	PowerPC	Power ISA	e600	e500
stwbrx	√	√	√	√
stwcx.	√	√	√	√
stwu	√	√	√	√
stwux	√	√	√	√
stwx	√	√	√	√
subfc[o][.]	√	√	√	√
subfe[o][.]	√	√	√	√
subfic	√	√	√	√
subfme[o][.]	√	√	√	√
subfze[o][.]	√	√	√	√
subf[o][.]	√	√	√	√
sync	√	msync	√	See Section 4.3.1, on page 13
tlbia	√	—	—	—
tlbie	√	—	√	—
tlbivax	—	√	—	√
tlbld	—	—	√	—
tlbli	—	—	√	—
tlbre	—	√	—	√
tlbsx		√	√	√
tlbsync	√	√	√	√
tlbwe	—	√	—	√
tw	√	√	√	√
twi	√	√	√	√
vaddcuw	—	√	√	—
vaddfp	—	√	√	—
vaddsbs	—	√	√	—
vaddshs	—	√	√	—
vaddsws	—	√	√	—
vaddubm	—	√	√	—
vaddubs	—	√	√	—
vadduhm	—	√	√	—
vadduhs	—	√	√	—
vadduwm	—	√	√	—
vadduws	—	√	√	—
vand	—	√	√	—
vandc	—	√	√	—
vavgsb	—	√	√	—
vavgsh	—	√	√	—
vavgsw	—	√	√	—
vavgub	—	√	√	—
vavguh	—	√	√	—
vavguw	—	√	√	—

Table 10. List of Instructions (continued)

Mnemonic	PowerPC	Power ISA	e600	e500
vcfsx	—	√	√	—
vcfux	—	√	√	—
vcmpbfp*x*	—	√	√	—
vcmpeqfp*x*	—	√	√	—
vcmpequb*x*	—	√	√	—
vcmpequh*x*	—	√	√	—
vcmpequw*x*	—	√	√	—
vcmpgefp*x*	—	√	√	—
vcmpgtfp*x*	—	√	√	—
vcmpgtsb*x*	—	√	√	—
vcmpgtsh*x*	—	√	√	—
vcmpgtsw*x*	—	√	√	—
vcmpgtub*x*	—	√	√	—
vcmpgtuh*x*	—	√	√	—
vcmpgtuw*x*	—	√	√	—
vctsxs	—	√	√	—
vctuxs	—	√	√	—
vexptefp	—	√	√	—
vlogefp	—	√	√	—
vmaddfp	—	√	√	—
vmaxfp	—	√	√	—
vmaxsb	—	√	√	—
vmaxsh	—	√	√	—
vmaxsw	—	√	√	—
vmaxub	—	√	√	—
vmaxuh	—	√	√	—
vmaxuw	—	√	√	—
vmhaddshs	—	√	√	—
vmhraddshs	—	√	√	—
vminfp	—	√	√	—
vminsb	—	√	√	—
vminsh	—	√	√	—
vminsw	—	√	√	—
vminub	—	√	√	—
vminuh	—	√	√	—
vminuw	—	√	√	—
vmladduhm	—	√	√	—
vmrghb	—	√	√	—
vmrghh	—	√	√	—
vmrghw	—	√	√	—
vmrglb	—	√	√	—
vmrglh	—	√	√	—

Table 10. List of Instructions (continued)

Mnemonic	PowerPC	Power ISA	e600	e500
vmrglw	—	√	√	—
vmsummbm	—	√	√	—
vmsumshm	—	√	√	—
vmsumshs	—	√	√	—
vmsumubm	—	√	√	—
vmsumuhm	—	√	√	—
vmsumuhs	—	√	√	—
vmulesb	—	√	√	—
vmulesh	—	√	√	—
vmuleub	—	√	√	—
vmuleuh	—	√	√	—
vmulosb	—	√	√	—
vmulosh	—	√	√	—
vmuloub	—	√	√	—
vmulouh	—	√	√	—
vnmsubfp	—	√	√	—
vnor	—	√	√	—
vor	—	√	√	—
vperm	—	√	√	—
vpkpx	—	√	√	—
vpkshss	—	√	√	—
vpkshus	—	√	√	—
vpkswss	—	√	√	—
vpkswus	—	√	√	—
vpkuhum	—	√	√	—
vpkuhus	—	√	√	—
vpkuwum	—	√	√	—
vpkuwus	—	√	√	—
vrefp	—	√	√	—
vrfim	—	√	√	—
vrfin	—	√	√	—
vrfip	—	√	√	—
vrfiz	—	√	√	—
vrlb	—	√	√	—
vrlh	—	√	√	—
vrlw	—	√	√	—
vrsqrtefp	—	√	√	—
vsel	—	√	√	—
vsl	—	√	√	—
vslb	—	√	√	—
vsldoi	—	√	√	—
vslh	—	√	√	—

Table 10. List of Instructions (continued)

Mnemonic	PowerPC	Power ISA	e600	e500
vslo	—	√	√	—
vslw	—	√	√	—
vspltb	—	√	√	—
vsplth	—	√	√	—
vspltisb	—	√	√	—
vspltish	—	√	√	—
vspltisw	—	√	√	—
vspltw	—	√	√	—
vsr	—	√	√	—
vsrab	—	√	√	—
vsrah	—	√	√	—
vsraw	—	√	√	—
vsrb	—	√	√	—
vsrh	—	√	√	—
vsro	—	√	√	—
vsrw	—	√	√	—
vsubcuw	—	√	√	—
vsubfp	—	√	√	—
vsubsbs	—	√	√	—
vsubshs	—	√	√	—
vsubsws	—	√	√	—
vsububm	—	√	√	—
vsububs	—	√	√	—
vsubuhm	—	√	√	—
vsubuhs	—	√	√	—
vsubuwm	—	√	√	—
vsubuws	—	√	√	—
vsum2sws	—	√	√	—
vsum4sbs	—	√	√	—
vsum4shs	—	√	√	—
vsum4ubs	—	√	√	—
vsumsws	—	√	√	—
vupkhpx	—	√	√	—
vupkhsb	—	√	√	—
vupkhsh	—	√	√	—
vupklpx	—	√	√	—
vupklsb	—	√	√	—
vupklsh	—	√	√	—
vxor	—	√	√	—
wrtee	—	√	—	√
wrteei	—	√	—	√

Table 10. List of Instructions (continued)

Mnemonic	PowerPC	Power ISA	e600	e500
xori[.]	√	√	√	√
xor[.]	√	√	√	√

5 Register Model

Most registers defined in the PowerPC architecture are unchanged in the e500 implementation of the Power Architecture model. A few have been replaced by other registers, and in some cases new fields are added, primarily to support functionality defined by categories that have been added to the architecture. Differences include the following:

- Bit numbering. 32-bit registers in the PowerPC architecture (e600) are numbered 0–31; the same registers in Power ISA (e500) are numbered 32–63. Any 32-bit registers that are defined as 64-bit registers in the Power ISA are treated as the lower word of the 64-bit versions. These include the GPRs, save/restore registers, and all registers that can hold addresses (such as the count and link registers).
- Register files. These sets of registers hold operands for computational, load, and store instructions. The architecture defines the following register files:
  - General-purpose registers (GPRs). All cores implement GPRs.

    The SPE uses the 32-bit GPRs extended to 64-bits. GPRs are often used to generate the effective address for instructions that access memory (because GPRs are used to hold addresses, 64-bit implementations require 64-bit GPRs). The e600 cores implement 32-bit GPRs. On the e600, these bits are numbered 0–31.

    See Section 5.1, "Register File Comparison."
  - Floating-point registers (FPRs). All cores that support the Power Architecture model base category floating-point instructions implement the FPRs, but the e500v1 and e500v2 do not.
  - Vector registers (VRs). The AltiVec instruction set implemented on the e600 uses VRs which are not supported on e500 cores.
- Instruction-accessible registers—Registers such as the condition register (CR), the floating-point status and control register (FPSCR), and some SPRs are accessed as side effects of executing certain instructions. All processors implement CRs, but processors that do not support FPRs also do not support the FPSCRs or floating-point functionality defined in the CR. Likewise, e500 cores do not support VRs.
- Special-purpose registers (SPRs)—On-chip registers that are part of the processor core. Although the basic set of SPRs is implemented across all cores, some SPRs may not be implemented on all cores, or may have different meanings relative to the core or to the device into which the core is integrated. Always check the register summary chapter in the SoC reference manual for the most specific information.

  Cores also include both the architecture-defined and implementation-specific SPRs required for the functionality provided. These differences are summarized at the register level in the comparison tables in the subsequent sections. Specific details are provided in the reference manuals for the cores and the integrated devices.

NOTE

- Performance monitor registers, or PMRs, offer an extensive set of on-chip registers similar to SPRs. These are defined by the Power ISA and implemented on the e500. This functionality is defined in SPRs on the e600 cores.

To optimize instruction execution, implementations typically employ duplicate space for certain heavily used registers, such as rename and shadow registers. Such microarchitectural resources vary from device to device and are not addressed here.

5.1 Register File Comparison

Figure 3 compares register files. Note that, as the GPRs in Figure 3 illustrate, bit numbering for 32-bit registers differs between the PowerPC architecture and the Power Architecture model.

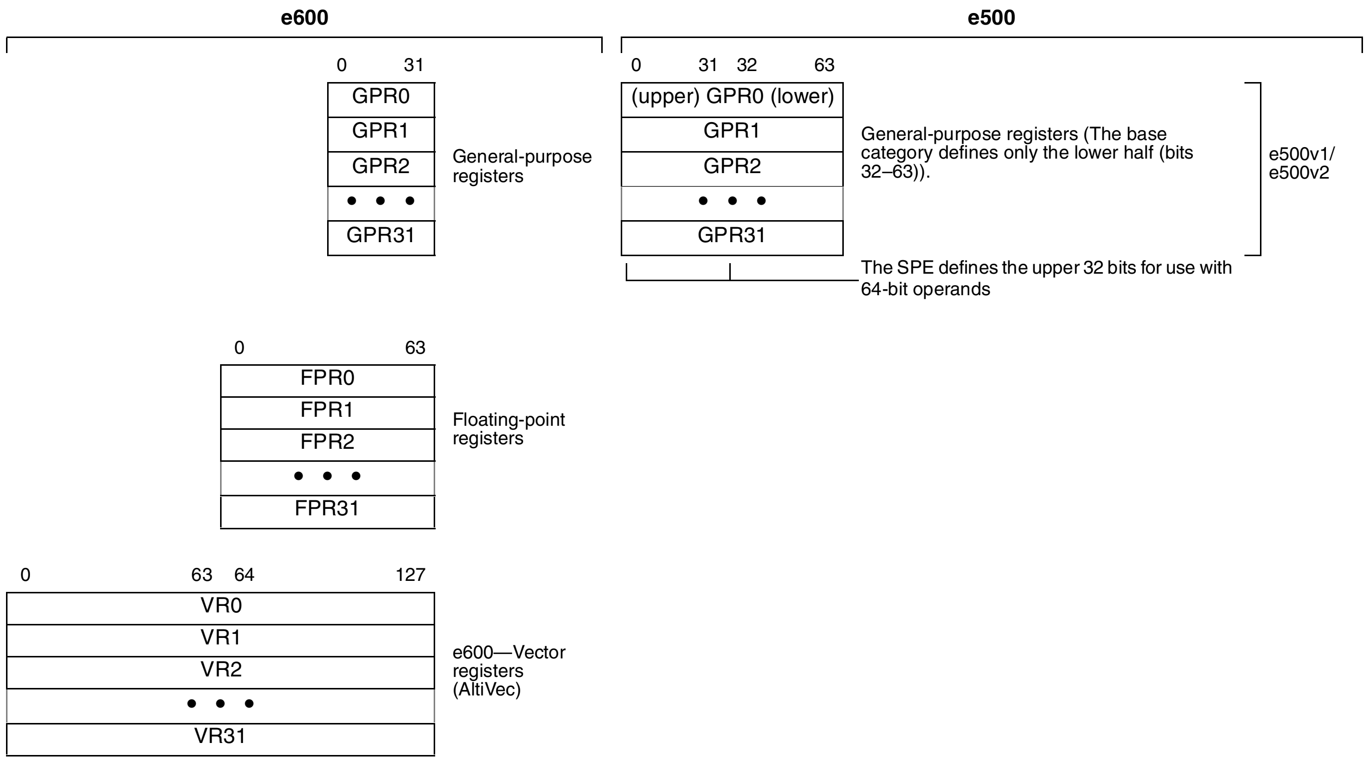

e600
e500
0 31
GPR0
GPR1
GPR2
• • •
GPR31
General-purpose registers
0 31 32 63
(upper) GPR0 (lower)
GPR1
GPR2
• • •
GPR31
General-purpose registers (The base category defines only the lower half (bits 32–63)).
e500v1/ e500v2
The SPE defines the upper 32 bits for use with 64-bit operands
0 63
FPR0
FPR1
FPR2
• • •
FPR31
Floating-point registers
0 63 64 127
VR0
VR1
VR2
• • •
VR31
e600—Vector registers (AltiVec)

Figure 3. Register File Comparison

Architecture-defined register files shown in Figure 3 are defined as follows:

- 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 GPR file consists of 32 GPRs designated as GPR0–GPR31.

  The e600 implements 32-bit GPRs with bits numbered 0–31.

  The SPE, implemented on e500v1 and e500v2 cores, extends the GPRs to accommodate 64-bit operands; scalar double-precision embedded floating-point instructions treat the 64 bits as a single operand; SPE vector instructions break the registers into two 32-bit elements, which for some instructions are broken into half-word elements.

- Floating-point registers (FPRs)—The floating-point model 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 Section 4.2.2, “Floating-Point Instructions (e600).”
- AltiVec vector registers (VRs)—AltiVec, now part of the Power ISA, defines a VR file that consists of thirty-two 128-bit VRs. The e500 does not implement these registers.

5.2 Instruction-Accessible Registers

Figure 4 shows a comparison of registers that may be updated as the by-product of instruction execution. For example, an overflow may update the condition register (CR), the floating-point status and control register (FPSCR), or the SPE/embedded floating-point status and control register (SPEFSCR). For some of these registers, such as the FPSCR and CR, explicit move to/move from instructions are defined to explicitly access these registers.

The differences in these register sets depend on whether SPE, AltiVec, and floating-point instructions are supported.

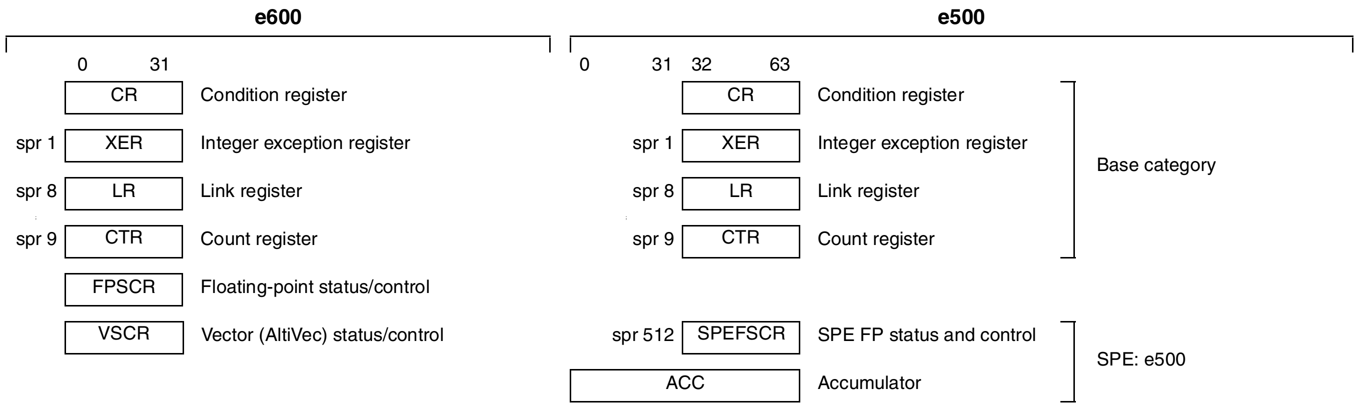


Figure 4. Instruction-Accessible Registers Comparison

The following e500 registers support SPE and embedded floating-point instructions:

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

5.3 Timer Register Comparison

Figure 5 shows a comparison of timer-related registers.

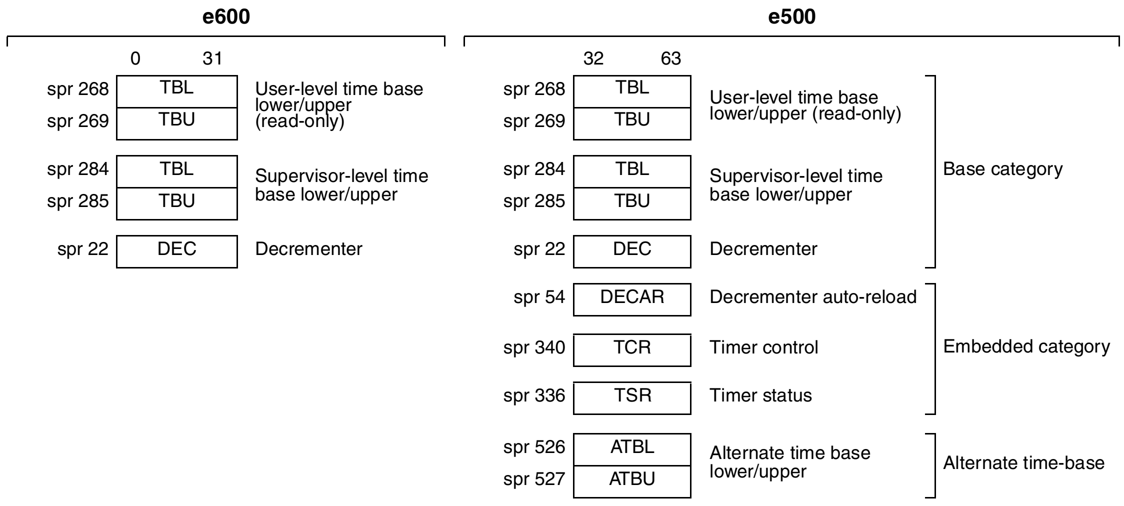


Figure 5. Time/Decrementer Registers Comparison

Both families implement the following registers:

- Time base (TBU and TBL). Provides timing functions for the system.
- 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, fixed-interval timer, or watchdog timer interrupt after a specified period.

The e500 implements Power ISA–defined registers that incorporate timing mechanisms for the fixed-interval and watchdog timer interrupts:

- Decrementer auto-reload register (DECAR). Can be used to automatically reload a programmed value into DEC. If DECAR is not used, a value has to be explicitly programmed into the DEC, as in the PowerPC architecture.
- 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.
- The alternate time base registers duplicate much of the functionality of the time base, but do not support the DECAR. The alternate time base is typically clocked at a higher frequency than the standard time base to offer a finer granularity.

5.4 MMU Control and Status Register Comparison

Because the original PowerPC architecture MMU specification was cumbersome for embedded applications, the e500 defined alternate features, especially to support software-managed page tables. See Section 7, “Memory Management Unit (MMU) Model.”

Figure 6 compares the MMU registers.

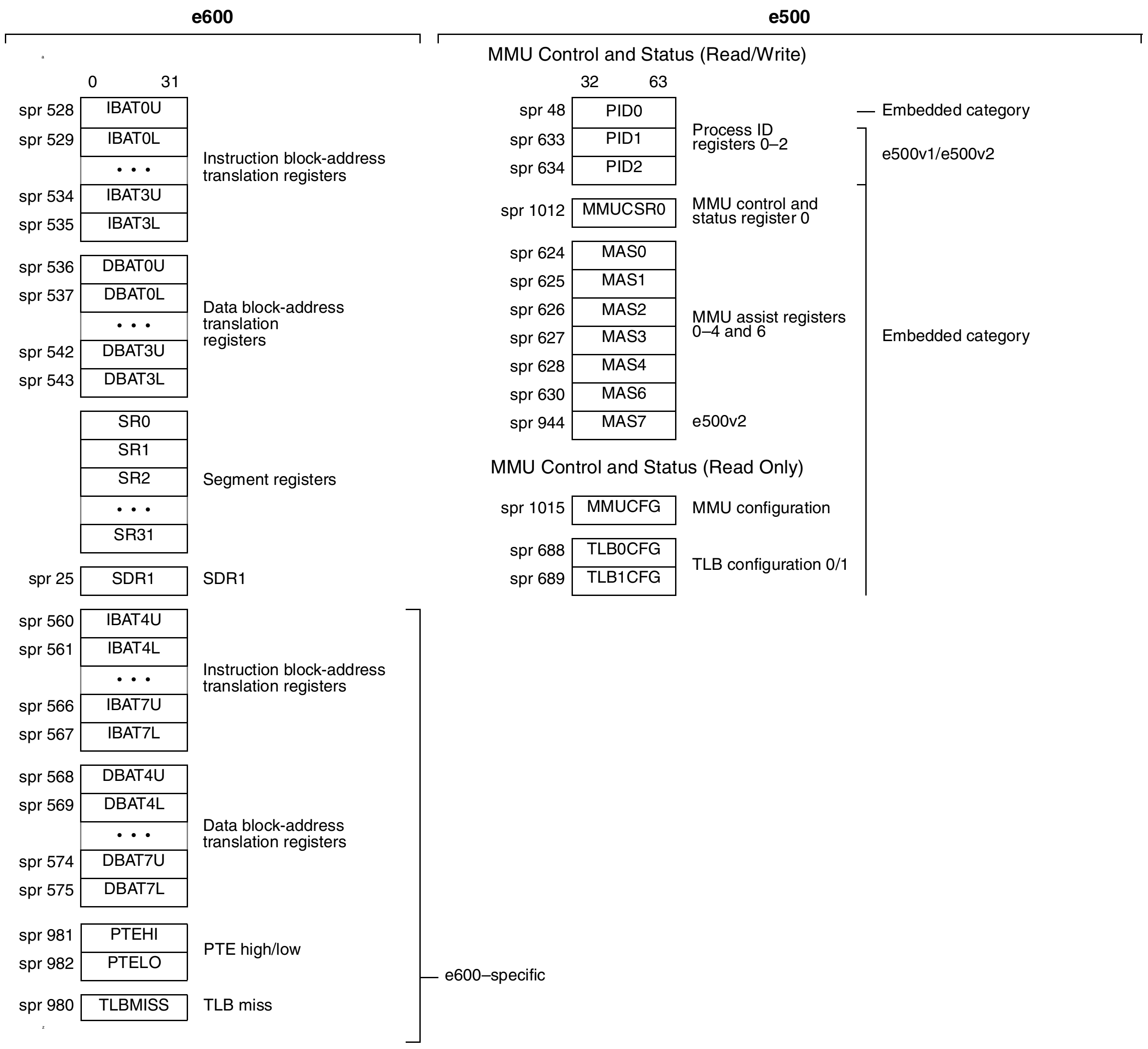


Figure 6. MMU Register Comparison

The e600 cores implement the following MMU registers defined by the PowerPC architecture, but not supported by the Power ISA embedded cores:

- Block address translation registers (BATs).
- SDR1

The e500 implements the following Power ISA–defined SPRs to support address translation:

- Process ID registers (PID0–PID2). Provides an identifier value associated with each effective address (instruction or data) generated by the processor. The Power ISA supports as many as 16 PIDs.
- MMU control and status register 0 (MMUCSR0). Used for general MMU control, for example, to invalidate TLBs.
- MMU assist (MAS) registers. Used with the **tlbwe** and **tlbre** instructions to configure and manage 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.

5.5 Cache Register Comparison

The Freescale EIS defines the e500 L1 cache configuration and status registers, shown in Figure 7. Neither version of the architecture defines cache registers.

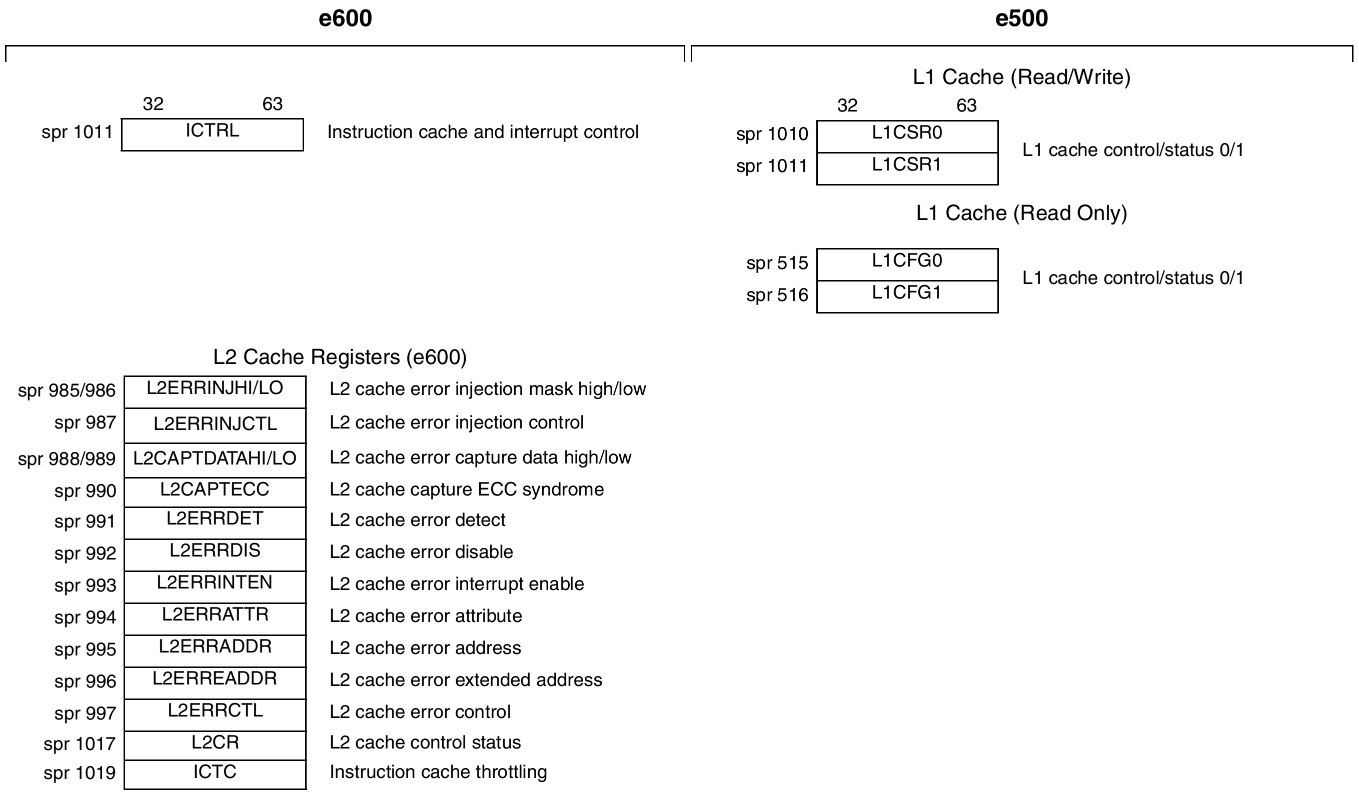


Figure 7. Cache Registers Comparison

The e500 registers in Figure 7 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.

The e600 implements the following L1 cache control bits in HID0:

- Instruction/data cache enable (ICE/DCE). Clearing the bit disables the cache; it can be neither accessed nor updated. Potential cache accesses from the bus (snoop and cache operations) are ignored and both caches are disabled at reset.
- Data/instruction cache lock (DLOCK/ILOCK). If this bit is set, all ways of the respective cache are locked. A locked cache supplies data normally on a hit, but is treated as a cache-inhibited

transaction on a miss. The e500 implements the cache locking instructions listed in Section 4.3.2, "Memory Control Instructions."

- Instruction/data cache flash invalidate (ICFI/DCFI). Setting this bit generates an invalidate operation that marks the state of each instruction cache block as invalid. Setting ICFI clears all the valid bits of the blocks and the PLRU bits to point to way L0 of each set.

The e600 includes cache way locking fields, ICTRL[ICWL] and LDSTRCR[DCWL])

See the core and SoC reference manuals for details about fields within these registers.

5.6 Interrupt Register Comparison

The Power ISA embedded category optimizes the architected resources to improve responsiveness to interrupts, especially for asynchronous interrupts signaled to the core from peripheral logic within the SoC. As Figure 8 shows, these differences include the following:

- In the PowerPC architecture which is implemented on the e600, an interrupt vector consists of a fixed offset prepended with a value as determined by MSR[IP], which is not part of the Power ISA. On e500 cores, these offsets are programmed through the interrupt vector prefix register (IVPR), which places the interrupt table in memory, and the interrupt vector offset registers (IVORs), which contain the offset for individual interrupts.

  IVORs hold the index from the base address provided by the IVPR for its respective interrupt type. IVORs provide storage for specific interrupts. The Power ISA definition allows implementations to define IVORs to support category- and implementation-specific interrupts. For example, the SPE defines IVOR32–IVOR35. Such IVORs are listed at the bottom of Table 11.
- To manage the increased traffic from peripheral devices, the Power ISA provides analogous resources for critical input interrupt with its own set of save and restore registers. This functionality also exists as implementation-specific functionality in some cores. The Power ISA defines similar resources for machine check interrupts implemented on e500 cores.
- Support for data related interrupts has changed, as follows:
  - The e500 implements the exception syndrome register (ESR) instead of the DSI syndrome register (DSISR). The DSISR is used for data storage and alignment interrupts. The ESR is used to track exceptions for a variety of interrupts.
  - The e500 implements the data exception address register (DEAR). DEAR is 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. The e600 implements the data address register (DAR) for this purpose.

Figure 8 compares the interrupt register models.

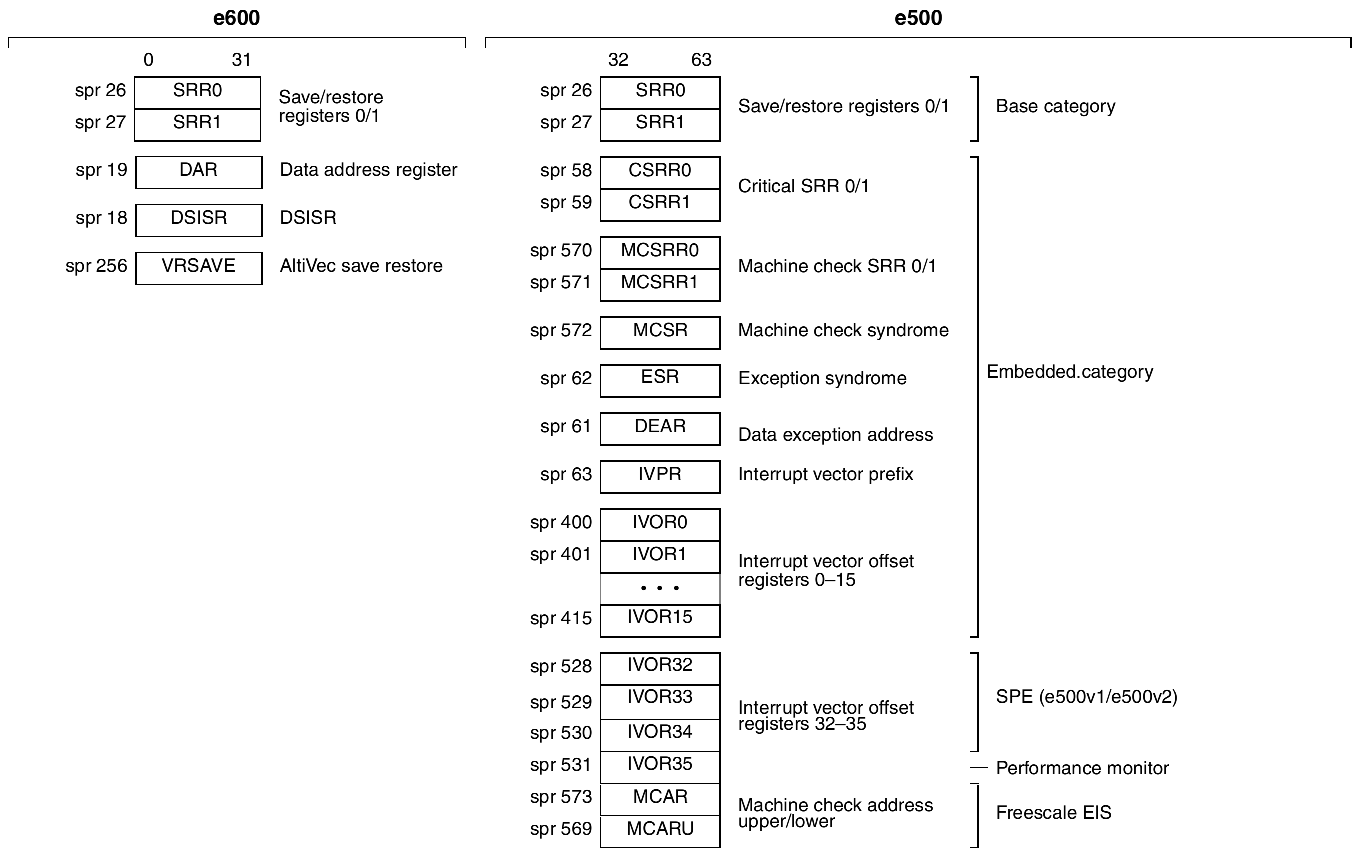


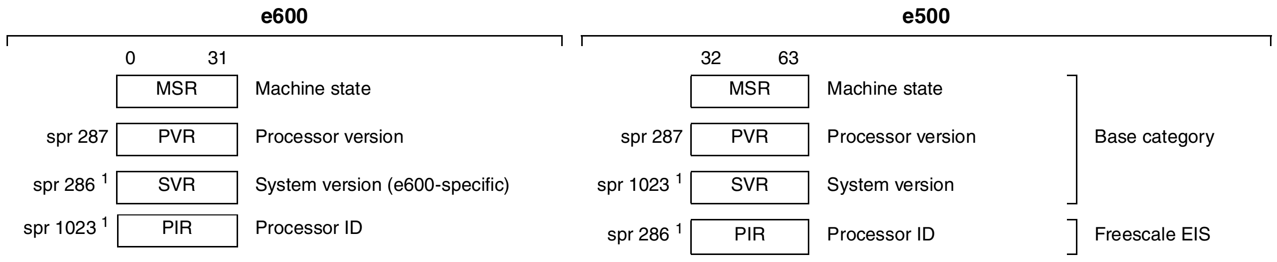
Figure 8. Interrupt Register Comparison

The e500 implements the following registers, defined by the Power ISA:

- The machine check interrupt model 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 e500 takes a machine check interrupt, it updates MCAR to indicate the address of the data associated with the machine check.

5.7 Configuration/Processor Control Register Comparison

The architecture defines registers that provide control, configuration, and status information of the machine state and process IDs. Figure 9 compares configuration registers. Note that this document does not address in detail all differences in the implementation of each register, particularly regarding MSR fields.



[1] Note that the SVR is SPR 1023 on the e500 and SPR 286 on the e600. The PIR is SPR 286 on the e500 and 1023 on the PIR.

Figure 9. Configuration Registers Comparison

- 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 MSR (e600) defines bits that enable data address translation (IR and DR) and modal big/little endian byte ordering (LE and ILE). On the e500 byte ordering is a page attribute configured through the MAS registers.

  The MSR includes bits for enabling and disabling asynchronous interrupts: EE for external interrupts, CE for critical interrupts, and ME for machine check interrupts. The core user documentation describes the behavior of these bits when the respective interrupt is taken and how they should be treated by the interrupt handler. Note also that the Power ISA implements the Write MSR External Enable instructions (**wrtee**[**i**]), which can be used to update only MSR[EE].

  MSR[LE] and MSR[ILE] on the e600 are used to set configure the big- and little-endian byte ordering; these are not implemented on the e500 and other Power ISA devices, which handle endianness on a per-page basis through the MAS registers.

  The MSR[IP] value places the interrupt table in either high or low memory. This is not implemented on the e500, which uses IVPR and the IVORs to define the interrupt table and place it in memory.
- 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 and Power ISA 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 system version register (SVR) identifies the integrated device that implements the core.

5.8 Performance Monitor Register Comparison

The e600 and e500 cores’ performance monitor utility uses the set of registers shown in Figure 10. The e600 uses SPRs to implement this functionality; the e500 processors implement this functionality in performance monitor registers (PMRs), which are part of the Power ISA. PMRs are similar to the SPRs and are accessed by **mtpmr** and **mfpmr** instructions.

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

It is important to note that the events counted can differ greatly among processors; consult the user documentation. Also note that most integrated devices implement a similar performance monitor that tracks events mostly involving peripheral device activities. These events may trigger an asynchronous interrupt, typically configured as an external interrupt.

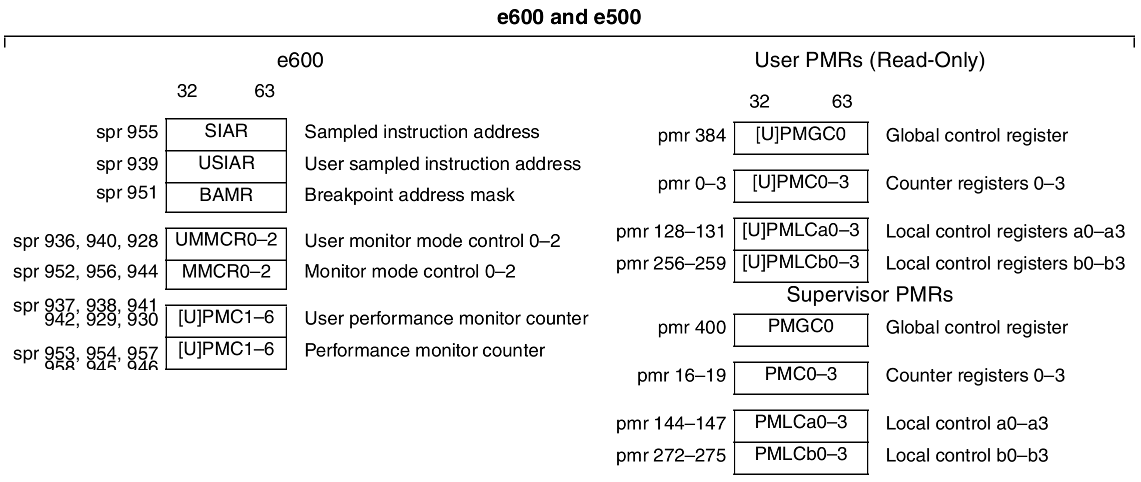


Figure 10. Performance Monitor Registers Comparison

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 can be 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 can be 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.

5.9 Debug Register Comparison

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 11 compares debug registers.

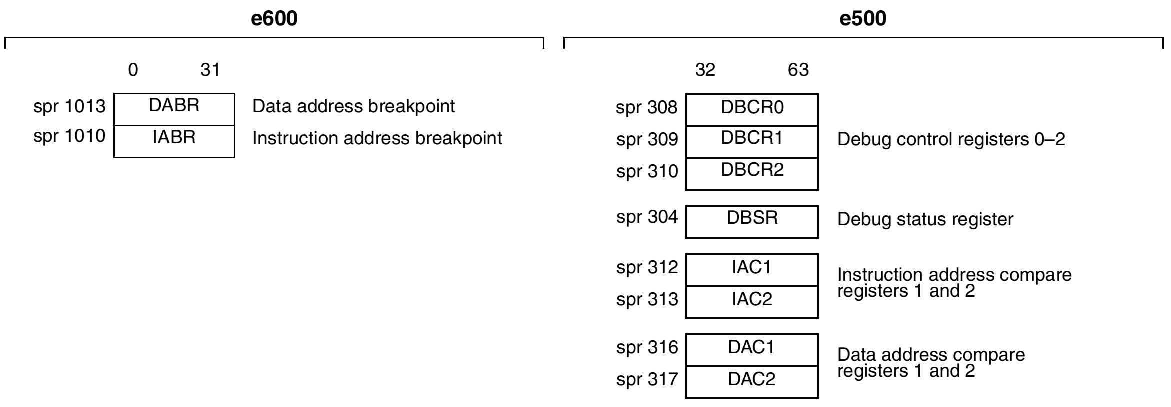


Figure 11. Debug Registers Comparison

The Power ISA does not implement data address breakpoint registers (DABRs) nor instruction address breakpoint registers (IABRs), but instead architects debugging support with the following registers:

- 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 (IACs and DACs). A debug event may be enabled to occur on an attempt to execute an instruction or access a data location from an address specified in an IAC or DAC, inside or outside a range specified by the IAC or DAC, or to blocks of addresses specified by the combination of the IACs and DACs.
- Note that additional enhanced embedded debug interrupt resources are described in Section 5.6, “Interrupt Register Comparison.”

5.10 Software-Use SPR Comparison

Software-use SPRs (SPRGs), shown in Figure 12, have no defined functionality, although many are added to the register set in conjunction with other functionality, for example the Freescale MMU architecture.

SPRGs consist of the following supervisor-level read/write registers:

- SPRG0–SPRG3—defined in both versions of the architecture.
- SPRG4–SPRG7—implementation-specific registers implemented on e600 cores, and also defined by the Power ISA.

On the e600, SPRGs are all supervisor-only, read/write registers. e500 SPRGs are defined by the Power ISA as follows:

- SPRG0–SPRG2—accessible only in supervisor mode.

- SPRG3—write-only in supervisor mode. It is readable in supervisor mode, but whether it can be read in user mode is implementation-dependent. Note that, as Figure 12 shows, the SPR numbers differ for user and supervisor accesses.
- SPRG4–SPRG7—write-only in supervisor mode but readable in supervisor or user mode. Note that, as Figure 12 shows, the SPR numbers differ for user and supervisor accesses.
- USPRG0—can be accessed in supervisor or user mode. Note that USPRG0 is a separate physical register from SPRG0.

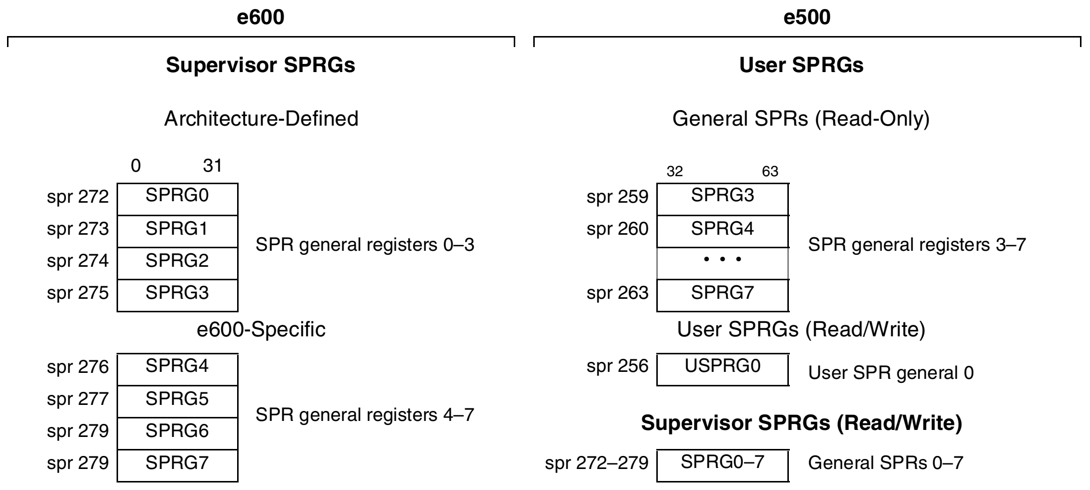


Figure 12. General SPRs (SPRGs)

5.11 Miscellaneous Implementation-Specific Register Comparison

To handle special functions, implementations typically have SPRs not defined by the architecture, some of which may appear on multiple implementations with similar functionality. In particular, implementations define hardware implementation-dependent registers (HIDs) that typically control hardware-related functionality as shown in Figure 13.

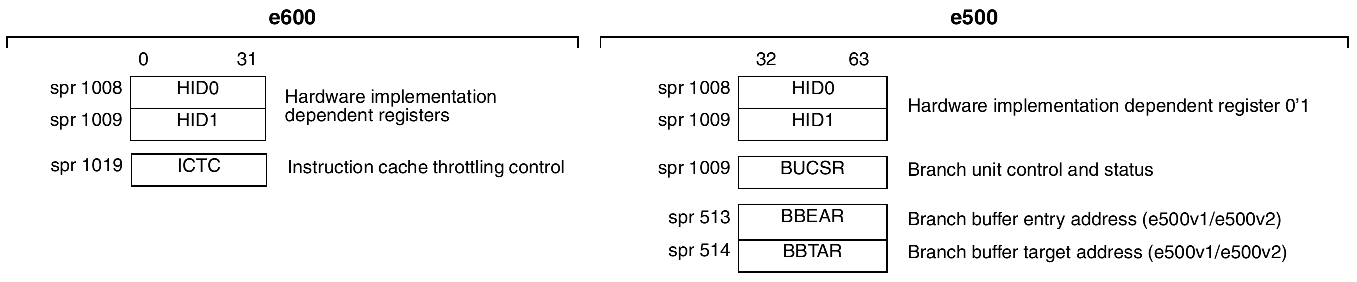

e600
e500
0
31
spr 1008
HID0
spr 1009
HID1
Hardware implementation dependent registers
spr 1019
ICTC
Instruction cache throttling control
32
63
spr 1008
HID0
spr 1009
HID1
Hardware implementation dependent register 0'1
spr 1009
BUCSR
Branch unit control and status
spr 513
BBEAR
Branch buffer entry address (e500v1/e500v2)
spr 514
BBTAR
Branch buffer target address (e500v1/e500v2)

Figure 13. Implementation-Specific Registers Comparison

6 Interrupt Model

Both architecture versions of the interrupt model are similar with respect to the interrupts that are defined and the kind of exceptions that can cause them. This is especially true for those interrupts that are closely related to program execution. The Power ISA extends the interrupt model somewhat both to provide greater responsiveness and lower interrupt latency critical to an embedded environment and to accommodate changes in the MMU model.

NOTE

Note that e600 documentation uses the terms ‘exception’ and ‘interrupt’ differently than the Power ISA and Freescale’s e500 documentation. This document uses the terms as follows:

- An exception is the event that, if enabled, causes the processor to take an interrupt. Exceptions are generated by signals from internal and external peripherals, instructions, the internal timer facility, debug events, or error conditions.
- An interrupt is the action in which the processor saves its context (typically the machine state register (MSR) and next instruction address) and begins execution at a predetermined interrupt handler address with a modified MSR.

Most of the general characteristics of the interrupt model are common across all architecture versions; the 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.

The conditions that cause exceptions can vary from processor to processor and some may be mode dependent. Consult the user documentation.

General differences between the PowerPC architecture and the Power ISA are as follows:

- The Power ISA embedded environment does not define, and e500 processors do not implement, a system reset interrupt. On Power ISA embedded cores, a system reset is typically initiated in one of the following ways:
  - Assertion of a signal that resets the internal state of the core complex
  - By writing a 1 to DBCR0[34], if MSR[DE] = 1
- 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 0x000*n_nnnn*. If IP is set, vector offsets are added to the physical address 0xFFF*n_nnnn*. Table 11 shows the vector offsets associated with each interrupt type. Finally, the PowerPC architecture includes the system reset, trace, and floating-point assist interrupts which are not part of the Power ISA.
- MSR[IR,DR] are cleared when the e600 takes an interrupt, putting it in real mode. Because Power ISA devices do not implement real mode, the e500 core is always translating effective addresses.
- Interrupts in the Power ISA embedded category. Defines interrupt vector offset registers (IVORs), interrupt vector prefix registers (IVPRs), and critical interrupts. An IVOR is assigned to each interrupt type. The IVPR provides the base address location to which the offset in the IVORs is added. Table 11 shows the IVORs associated with each interrupt type.

  The save and restore resources are part of the are largely identical to those defined by the OEA. Save and restore registers (shown in Figure 8) save the return address and machine state when they are taken. A return from interrupt instruction (**rfi**, **rfci**, or **rfmci**) restores state at the end of the interrupt routine.

The Power ISA resources are defined as follows:

— Critical interrupts—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 Power ISA version defines the critical input, watchdog timer, and debug interrupts as critical interrupts (although debug interrupts may be implemented as separate interrupt types).

— Machine check interrupt—Analogous to critical interrupt with separate save and restore registers (MCSRR0/MCSRR1) and **rfmci** instruction.

Table 11 lists other differences.

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

Table 11 shows a comparison of the interrupt models.

Table 11. Interrupts and Conditions—Overview

Interrupt Type	**Vector Offset** ('—' Indicates not implemented)		**Cause/Description**
	e600	**e500**	
System reset	0x100	—	Not implemented on e500
Critical input	—	IVOR0	Assertion of *cint* typically managed by a programmable interrupt controller integrated into the SoC and enabled through MSR[CE]. Similar to external interrupt.
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, the processor goes into checkstop. e500 provides separate resources MCSRR0, MCSRR1, and **rfmci**. An address related to the machine check may be stored in MCAR. MCSR reports the cause of the machine check.
Data storage interrupt	0x300	IVOR2	A data memory access cannot be performed. On the e500, the ESR reports the cause and DEAR holds the EA of the data access. e600: DSISR reports the cause; DAR is set based on DSISR settings.

Table 11. Interrupts and Conditions—Overview (continued)

Interrupt Type	Vector Offset ('—' Indicates not implemented)		Cause/Description
	e600	e500	
Instruction storage interrupt	0x400	IVOR3	Instruction fetch cannot be performed. Causes include the following: • The EA cannot be translated. For example, when there is a page fault for this portion of the translation, an ISI must be taken to retrieve the page (and possibly the translation), typically from a storage device. • An attempt is made to fetch an instruction from a no-execute memory region or from guarded memory when MSR[IR] = 1. • The fetch access violates memory protection. e500: ISI assists implementations that: • cannot dynamically switch byte ordering between consecutive accesses • do not support the byte order for a class of accesses • 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).
Alignment	0x600	IVOR5	The processor cannot perform a memory access because of one of the following: • The operand of a load or store is not aligned. • a **dcbz** referenced storage that is write-through required or cannot be established in the data cache. e500: ESR reports the interrupt cause; DEAR holds the EA of the data access. e600: DSISR reports the cause; DAR is set based on DSISR. Implementations may vary with respect to taking interrupts for certain exception conditions. Consult the user documentation.
Program	0x700	IVOR6	One of the following conditions occurs during instruction execution: • Floating-point enabled exception—Generated when MSR[FE0,FE1] $\neq$ 00 and FPSCR[FEX] is set. Not implemented on the e500v1 or e500v2. 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. • Illegal or unimplemented 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. e500: an unimplemented operation exception may occur if an unimplemented, defined instruction is encountered. Otherwise, an illegal instruction interrupt occurs. ESR reports the cause.
Floating-point unavailable	0x800	IVOR7	e600: 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.

Table 11. Interrupts and Conditions—Overview (continued)

Interrupt Type	Vector Offset ('—' Indicates not implemented)		Cause/Description
	e600	**e500**	
Decrementer	0x900	IVOR10	As defined by the PowerPC architecture: occurs when the msb of the DEC changes from 0 to 1 and MSR[EE] = 1. e500: implements the additional Power ISA–defined resources: TSR records status on timer events. An auto-reload value in the DECAR is written to DEC when it decrements from 0x0000_0001 to 0x0000_0000.
System call	0xC00	IVOR8	Occurs when a System Call (**sc**) instruction is executed.
Trace	0xD00	—	e600: MSR[SE] =1 or a branch instruction is completing and MSR[BE] =1. The e600 core operates as specified in the OEA by taking this exception on an **isync**. e500: not implemented
Instruction translation miss	0x1000	—	e600: The EA for an instruction fetch cannot be translated by the ITLB. e500: Not implemented. See instruction TLB error (IVOR14)
Data load translation miss	0x1100	—	e600: The EA for a data load operation cannot be translated by the DTLB. e500: Not implemented. See data TLB error (IVOR13)
Data store translation miss	0x1200	—	e600: The effective address for a data store operation cannot be translated by the DTLB, or when a DTLB hit occurs and the change bit in the PTE must be set due to a data store operation. e500: Not implemented. See data TLB error (IVOR13)
Instruction address breakpoint	0x1300	—	The address (bits 0–29) in the IABR matches the next instruction to complete in the completion unit, and IABR[30] is set.
System management interrupt	0x1400	—	MSR[EE] = 1 and the *smi* input is asserted. e500: Not implemented.
AltiVec assist	0x1600	—	AltiVec assist. May occur if an AltiVec floating-point instruction detects denormalized data as an input or output in Java mode. After this exception occurs, execution resumes at offset 0x01600 from the physical base address indicated by MSR[IP].
Performance monitor	0x0F00	IVOR35	An interrupt-enabled event defined by the performance monitor occurred. The e600 implements performance monitor registers using SPRs; the e300 uses PMRs.
Fixed interval timer	—	IVOR11	A fixed-interval timer exception exists (TSR[FIS] = 1), and the interrupt is enabled (TCR[FIE] = 1 and MSR[EE] = 1).
Watchdog timer	—	IVOR12	Critical interrupt. Occurs when a watchdog timer exception exists (TSR[WIS] = 1), and the interrupt is enabled (TCR[WIE] = 1 and MSR[CE] = 1).
Data TLB error	—	IVOR13	A virtual address associated with an instruction fetch does not match any valid TLB entry.
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).

Table 11. Interrupts and Conditions—Overview (continued)

Interrupt Type	Vector Offset ('—' Indicates not implemented)		Cause/Description
	e600	e500	
Vector (SPE/AltiVec) unavailable	0x0F20	IVOR32	MSR[SPE] is cleared and an SPE or embedded floating-point instruction is executed.
			MSR[SPV] is cleared and an SPE/embedded floating-point category instruction is executed. The Power ISA defines this interrupt for use with future implementations that support AltiVec. On the e600, this occurs due to an attempt to execute any non-streaming AltiVec instruction when MSR[VEC] = 0. This exception is not taken for data streaming instructions (**dst*x***, **dss,** or **dssall**).
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

7 Memory Management Unit (MMU) Model

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

The flow diagram in Figure 14 gives a high-level comparison of the address translation mechanisms.

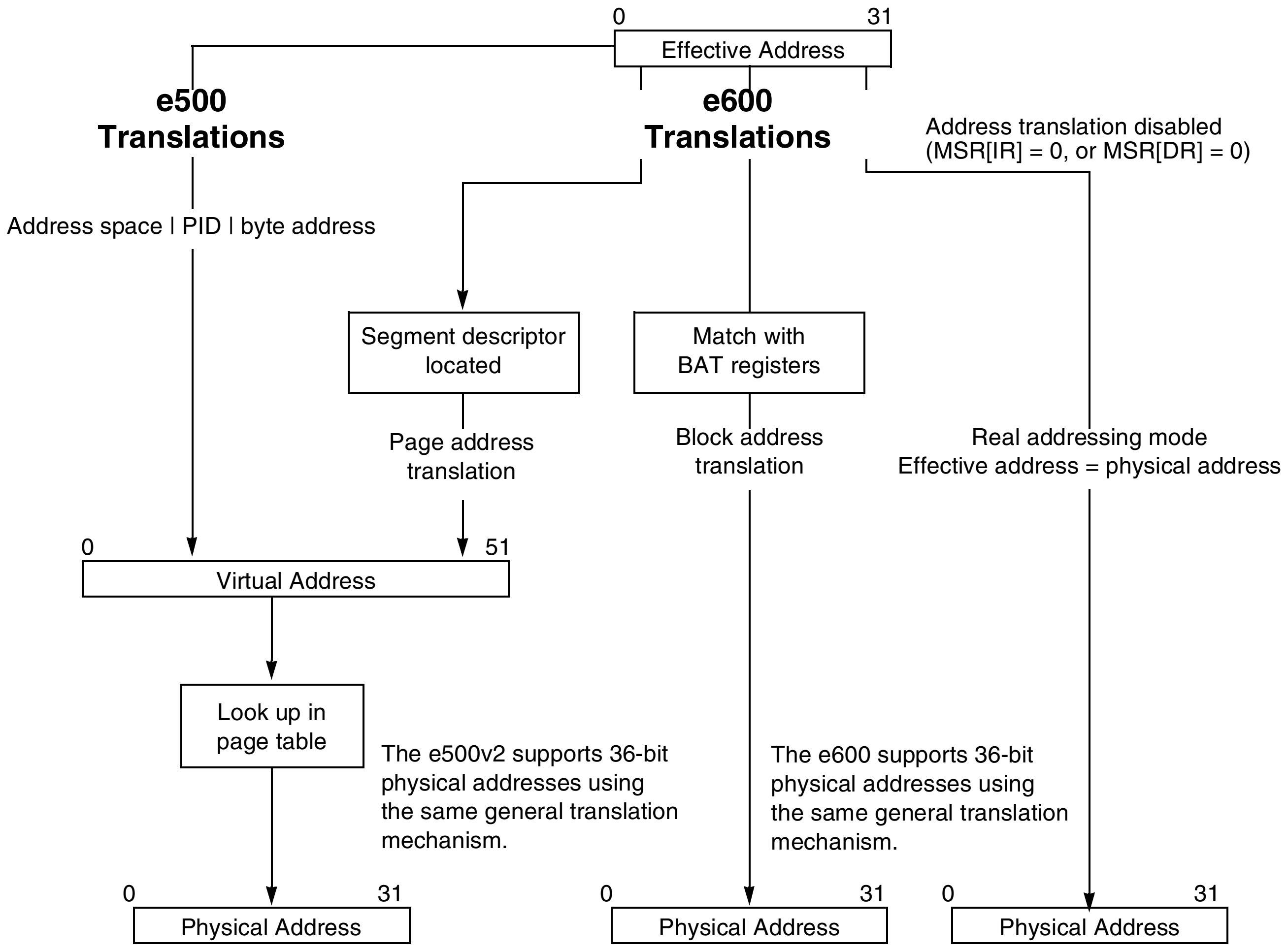

0
31
Effective Address
e500 Translations
e600 Translations
Address translation disabled (MSR[IR] = 0, or MSR[DR] = 0)
Address space | PID | byte address
Segment descriptor located
Match with BAT registers
Page address translation
Block address translation
Real addressing mode Effective address = physical address
0
51
Virtual Address
Look up in page table
The e500v2 supports 36-bit physical addresses using the same general translation mechanism.
The e600 supports 36-bit physical addresses using the same general translation mechanism.
0
31
Physical Address
0
31
Physical Address
0
31
Physical Address

Figure 14. Address Translation Types

Figure 12 outlines general differences between the PowerPC architecture 1.10 and the Power Architecture model embedded category MMU models.

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

In addition to instruction accesses and data accesses generated by load and store instructions, addresses specified by cache instructions also require address translation.

Translation lookaside buffers (TLBs) are commonly implemented to keep recently used page address translations on-chip.

The MMU models shares many general characteristics, particularly those related to memory protection and cache coherency and the general concepts of pages and TLBs. Differences are described in Section 7.1, “MMU Features in the PowerPC Architecture Definition.”

7.1 MMU Features in the PowerPC Architecture Definition

The e600 supports three types of address translation: page-address translation, block address translation and real mode (where the hardware translation mechanism is turned off and the effective address is used as the physical address). Power ISA devices, including the e500, do not support real mode.

Page address translation is defined in terms of segment descriptors implemented as a set of 16 segment registers (SRs). The segment information translates the effective address to an interim virtual address, and the page-table information translates the virtual address to a physical address. Effective address spaces are divided into 256-Mbyte segments. Segments that correspond to memory-mapped areas are divided into 4-Kbyte pages. As shown in Section 5.4, "MMU Control and Status Register Comparison," Power ISA devices do not support SRs.

The definition of the segment and page-table data structures provides significant flexibility for a range of computing environments. Therefore, the methods for storing segment or page-table information on-chip vary from implementation to implementation. For example, the e600 provides the implement-specific load TLB entry instructions (**tlbld** and **tlbli**) to directly access TLBs. The Power ISA defines **tlbwe** and **tlbre**, which the e500 uses to directly configure TLBs with translation and memory protection information by loading and storing values defined in the memory assist (MAS) registers. The e500 implements additional instructions for searching and invalidating entries and for synchronizing TLB accesses.

The PowerPC architecture describes a hardware model for providing page address configuration, protection, and translation, but the flexibility of the architecture also allows implementation-specific, software-managed MMUs, such as that implemented on e300 devices. The Power ISA defines an architecture for software MMU management, which is likewise flexible and may vary somewhat among implementations, described in Section 7.2, "MMU Features in the Embedded Category Definition."

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

The PowerPC architecture block address translation (BAT) mechanism allows the operating system to configure attributes for blocks of memory through a set of paired SPRs, described in Section 5.4, "MMU Control and Status Register Comparison." The BATs also contain protection and memory coherency information. As Figure 6 shows, separate BATs are defined for instruction memory (IBATs) and the data memory (DBATs). Also as Figure 6 shows, BATs and block address translation are not defined by the Power ISA and not implemented on the e500.

7.2 MMU Features in the Embedded Category Definition

Note that the Power ISA does not support the Power Architecture translation enable bits, MSR[IR,DR]; thus there is no default real mode in which the effective address (EA) is the same as the physical address. Translation is always enabled.

The embedded MMU model supports demand-paged virtual memory as well as a variety of management methods that depend on precise control of effective-to-real address translation and configurable memory protection. Address translation misses and protection faults cause precise exceptions. Sufficient information is available for system software to correct the fault and restart the faulting instruction.

Each program on a 32-bit implementation can access 2^{32} bytes of effective address space, subject to limitations imposed by the operating system. In a typical system, each program's EA space is a subset of a larger virtual address (VA) space managed by the operating system. Note that the e500v1 supports 32-bit effective addresses; the e500v2 supports 36-bit effective addresses.

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

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

The MMU model defines a set of MMU assist (MAS) registers that can be programmed via the **mtspr** instructions to update the TLBs directly with translation and configuration information. The configuration data in the MAS registers is written to the TLBs on the execution of a TLB Write Entry (**tlbwe**) instruction. Likewise, TLB contents can be saved back to the MAS registers by executing a TLB Read Entry (**tlbre**) instruction. The TLB Search Indexed instruction (**tlbsx**) searches valid TLB arrays for an entry corresponding to the virtual address and reads appropriate values into the MAS registers.

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

Table 12. PowerPC Architecture and Power ISA Embedded MMU Models

e600	e500: Power ISA Embedded Environment
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	Supports both fixed- and variable-sized page address translation mechanisms
Segmented memory model	Segments not defined
Hardware page address translation definition with little architected support for software management. The e600 supports software table searches.	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 little-endian support provided through MSR[LE] and MSR[ILE].	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.

The e500 processor executes the TLB Read Entry and TLB Write Entry instructions (**tlbre** and **tlbwe**) by reading or writing the contents of a set of MMU assist (MAS) SPRs into the TLBs. The MAS registers provide the translation, protection, byte-ordering, and cache characteristics for the relevant pages.

8 Revision History

Table 13 provides a revision history for this application note.

Table 13. Document Revision History

Rev. Number	Date	Substantive Change(s)
0	10/31/2007	Initial release.

Document Number: AN3531
Rev. 0
10/2007