Using the system address map described in this chapter as a base for discussion, the environment foreseen by the PC Emulation option has all of the following attributes:

- From the points of view of the processor and I/O bus masters (including third party DMA controllers), an image in memory, from 0 through a "top of emulated memory," which exactly matches that of a PC.
- The io-hole is enabled, to allow I/O bus master and third party DMA access to legacy devices such as VGA.
- From the point of view of the processor, memory above the "top of emulated memory" in which an emulator may reside, including areas for unique interrupt and exception handling code.
- From the points of view of I/O bus masters (including ISA bus masters, if below 16 MB), the address space above the "top of emulated memory" is prevented from mapping to System Memory, and may instead reference PCI/ISA memory or be intercepted by the host bridge as an error.
- From the points of view of I/O bus masters, disabling of the alias for the low 16 MB of System Memory (the system-memory-alias space) so that address range would address PCI memory or be intercepted by the host bridge as an error.

If a platform implements the PC Emulation option, this will be indicated by the existence of the *pc-emulation* property in the root node of the OF device tree, and the option is enabled by the *set-pc-emulation* method in the system node. In addition, the *set-pc-emulation* method initializes the Top of Emulated Memory Register (TEMR) and the Exception Relocation Register (ERR), and assures the requirement 3–34 is met. Figure 9 on page 46 shows an example of an address map with the PC Emulation option enabled.

Personal Use Copy - Not for Reproduction

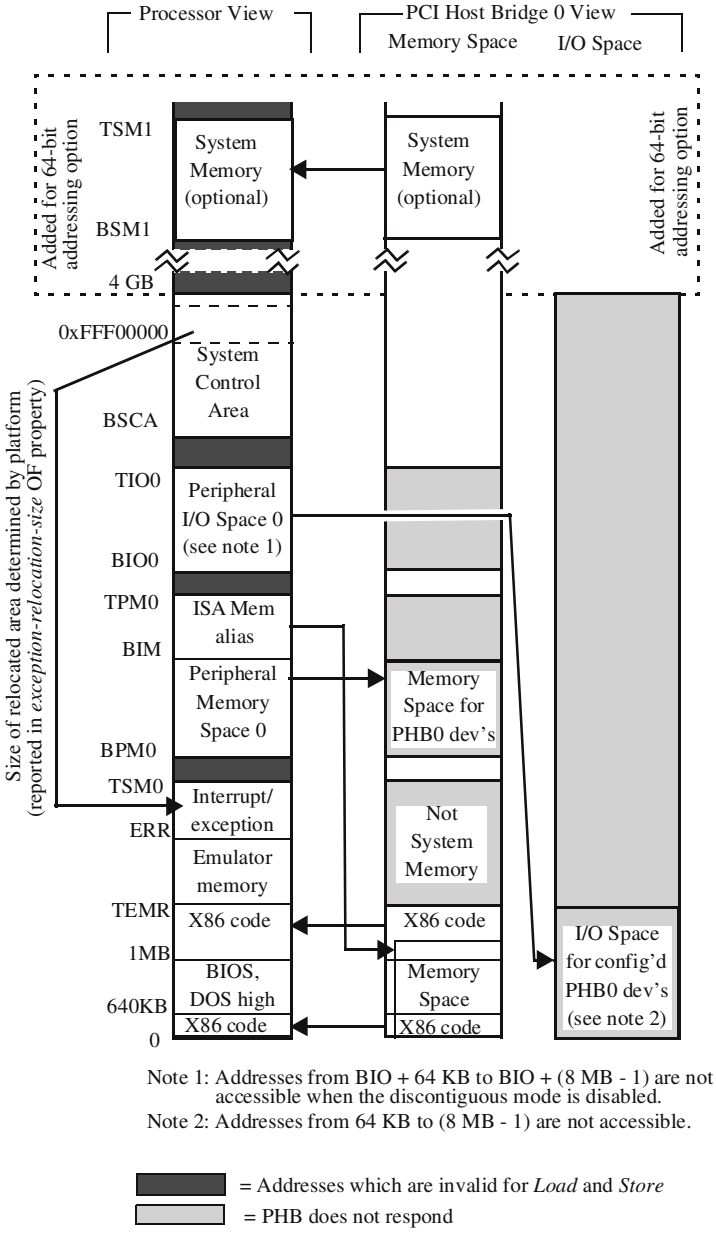

Figure 9. Example Address Map with the PC Emulation Option Enabled

Personal Use Copy - Not for Reproduction

To create the environment described above, one must define two pointer registers and associate some new behaviors with various regions of the system address map. The following requirements describe the functionality that is needed to provide the appropriate system address map.

Requirements:

3–31. For PC Emulation option: The platform must implement the TEMR, and all of the following must be true:

a. The TEMR must contain the largest address in System Memory which is set aside for the image of PC memory.

b. The granularity of the contents of TEMR must be 1 MB.

c. The HB must not respond to I/O bus Memory Space DMA operations in the address range of (TEMR + 1) to the top of System Memory, or must respond and signal an invalid address error.

3–32. For PC Emulation option: PCI devices must not be configured between (TEMR + 1) and the top of System Memory which is below BSCA.

3–33. For PC Emulation option: The platform must implement the ERR, and all of the following must be true:

a. The value of *exception-relocation-size* OF property must be the amount of address space above 0xFFF00000 that is relocated down into System Memory and is a fixed value, as determined by the platform.

b. The granularity of *exception-relocation-size* must be 4 KB and the minimum size must be 12 KB.

c. The contents of the ERR must be the address of the base of the region in System Memory to which references to the interrupt/exception handling area (normally at the top of the 32-bit address space) are relocated.

d. The granularity of the ERR must be 1 MB.

3–34. For PC Emulation option: When the PC Emulation option is enabled, the peripheral-memory-alias space must be enabled, the system-memory-alias space must be disabled, and the io-hole must be enabled.

Hardware Implementation Note: Since the software only has access to the ERR and the TEMR through the *set-pc-emulation* OF method, the

Personal Use Copy - Not for Reproduction

actual implementation of these registers can be abstracted from the software, and the value that gets put into the actual hardware registers can be transformed by *set-pc-emulation* to be something more directly usable by the hardware.

Software Implementation Notes:

- It is assumed that the processor is programmed with Machine State Register (MSR) Interrupt Prefix (IP) bit set to a 1 (MSR_{IP}=1; that is, exception/interrupt handlers at the top of the 32-bit address space).
- Software must set up the interrupt/exception area at the new location before relocating that area via the ERR (that is, before calling the *set-pc-emulation* method which sets the ERR).
- The s*et-pc-emulation* method, when used to disable the PC Emulation option, does not have to put the system-memory-alias space back into the state that it was in before the enabling of the PC Emulation option.
- The s*et-pc-emulation* method may use the *set-initial-aliases* OF method of the PHB0 node to assure that the peripheral-memory-alias is enabled and may use the *set-io-hole* OF method of the PHB- to assure that the io-hole is enabled.
- The *set-initial-aliases* method should not be called while the PC Emulation option is enabled, or else the system-memory-alias may get re-enabled.

Personal Use Copy - Not for Reproduction

Processor and Memory

4

The purpose of this chapter is to specify the processor and memory related requirements of the CHRP architecture. The processor architecture section addresses differences between the processors in the PowerPC family as well as their interface variations and features of note. The memory architecture section addresses coherency, minimum system memory requirements, memory controller requirements, and cache requirements.

4.1 Processor Architecture

The PowerPC architecture governs software compatibility at an instruction set and environment level. However, each processor implementation has unique characteristics which are described in its user's manual. To facilitate shrink-wrapped software, the CHRP architecture places some limitations on the variability in processor implementations. Nonetheless, it is possible for these differences to affect platform-aware software. Further, it is possible for system support chips (for example memory controller, L2 cache, PCI bridge) to affect platform-aware software. Platform-aware software is defined as any software which attempts to manipulate the platform external to the processor itself, including optimization of external data transfers and synchronization, for example. Lastly, evolution of the PowerPC architecture and implementations creates a need for both software and hardware developers to stay current with its progress. The following material highlights areas deserving special attention and provides pointers to the latest information.

4.1.1 Processor Architecture Compliance

The PowerPC architecture is defined in *The PowerPC Architecture* [1]. This architecture description is broken into three parts as defined below:

- Book I, *PowerPC User Instruction Set Architecture*
- Book II, *PowerPC Virtual Environment Architecture*
- Book III, *PowerPC Operating Environment Architecture*

The latest updates to the PowerPC architecture can be obtained by accessing the following URL: *http://www.austin.ibm.com/tech/ppc-chg.html*

Requirements:

4–1. Platforms must incorporate only processors which comply fully with the PowerPC architecture.

4–2. **For the Symmetric Multiprocessor option:** Multiprocessing platforms must use only processors which implement the processor identification register. See *PowerPC 604 RISC Microprocessor User's Manual* [6] for a definition of this register.

4–3. Platforms must incorporate only processors which implement *tlbie* and *tlbsync*, and *slbie* and *slbia* for 64-bit implementations.

4–4. Except where specifically noted otherwise in Section 4.1.4, "PowerPC Architecture Features Deserving Comment," on page 53, platforms must support all functions specified by the PowerPC architecture.

Hardware Implementation Note: Requirements 4–1, 4–2, and 4–3 may restrict the platform developer's choice of processor. For example, an embedded processor might have a different operating environment architecture. All announced PowerPC microprocessors in the 600-series (for example, 603, 603e, and 604) conform to these requirements, with the exception that the 603 family does not support requirement 4–2.

Hardware and Software Implementation Note: The PowerPC architecture and the CHRP architecture view *tlbia* as an optional performance enhancement. Processors need not implement *tlbia*. Software that needs to purge the TLB should provide a sequence of instructions that is functionally equivalent to *tlbia* and use the content of the OF device tree to choose the software implementation or the hardware

Personal Use Copy - Not for Reproduction

instruction. See Section 4.1.2, “PowerPC Microprocessor Differences,” on page 51 for details.

Hardware Implementation Note: The timebase freeze/thaw functionality described in requirement 12–8 may also be useful to a uniprocessor system in achieving B2 and better security ratings.

4.1.2 PowerPC Microprocessor Differences

There are a few significant differences among the programming models provided by PowerPC microprocessors in the 600-series. The 603 family implements a set of power management modes which are not described in the PowerPC architecture, and which are different from the minimal support provided by the 604 family. The 64-bit implementations use a different page table format which enables a 64-bit address space, and adds a 64-bit execution mode. A more complete understanding of these differences may be obtained by studying *The PowerPC Architecture* [1] and the user’s manuals for the various processors. The 601 is not supported by the CHRP architecture.

The CHRP architecture creators cooperate with processor designers to maintain a list of supported differences, to be used by operating systems instead of the processor version number (PVN), enabling execution on future processors. Open Firmware communicates these differences via properties of the *cpu* node of the OF device tree. The currently supported differences and their associated properties in the OF device tree include the following: *64-bit*, *32-64-bridge*, *603-translation*, *603-power-management, general-purpose, graphics, tlbia, performance-monitor, emulation-assists,* and *external-control*. See *PowerPC processor binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [11] for more details.

In addition to the differences described above, the PowerPC architecture allows for implementation-specific, software-transparent differences. The second and third pages above *Base* (the address specified for interrupt vector location by MSR_{IP}) are reserved for implementation-specific routines provided by the processor designer which make details of the processor implementation transparent to operating system and application software; see Table 8 on page 52 and also the description of the ERR in requirement 3–33. The software-assisted translation mechanism implemented by the 603 is an example of how the second and third pages can be used.

Software Implementation Note: A 64-bit operating system need not require 64-bit client interface services in order to boot. Because of the problems that might be introduced by dynamically switching between 32-bit and 64-bit modes in Open Firmware, the configuration variable

Personal Use Copy - Not for Reproduction

64-bit-mode? is provided so that Open Firmware can statically configure itself to the needs of the operating system.

Software Implementation Note: Only those portions of the range from 0xFFF00000 to 0xFFF02FFF which are actually used prior to setting MSR_{IP}=0 must be set aside in ROM as described in Table 8 on page 52.

.

Table 8. Fixed Real Storage Locations Having Defined Uses

Real Address (hex) when MSR_{IP}=0	Real Address (hex) when MSR_{IP}=1	Use
0000 - 00FF	FFF00000-FFF000FF	Reserved for OS use.
0100 - 0FFF	FFF00100-FFF00FFF	These locations are for the interrupt vectors.
1000 - 2FFF	FFF01000-FFF02FFF	Reserved for microprocessor-dependent firmware use. OF must initialize this area and OS software must preserve this area.

Requirements:

4–5. Operating systems must use the properties of the *cpu* node of the OF device tree to determine the programming model of the processor implementation.

4–6. Operating systems must provide an execution path which uses the properties of the *cpu* node of the OF device. The PVN is available to the platform aware operating system for exceptional cases such as performance optimization and errata handling.

4–7. Operating systems must support both of the page table formats (32-bit and 64-bit) defined by the PowerPC architecture.

4–8. Processors which exhibit the *64-bit* property of the *cpu* node of the OF device tree must also implement the “bridge architecture,” an option in the PowerPC architecture. See the updates on the Internet associated with *The PowerPC Architecture* [1].

4–9. Platforms must restrict their choice of processors to those whose programming models may be described by modifications to that of the 604 by the properties defined for the *cpu* node of the OF device tree in *PowerPC processor binding to: IEEE Std 1275-1994 Standard for Boot*

Personal Use Copy - Not for Reproduction

(Initialization, Configuration) Firmware [11] (for example, *64-bit* and *603-translation).*

4–10. Platform firmware must initialize the second and third pages above *Base* correctly for the processor in the platform, and in the correct endian mode, prior to giving control to the operating system.

4–11. Operating system and application software must not alter the state of the second and third pages above *Base*.

4.1.3 Processor Interface Variations

The processor interface for the 32-bit processors (for example, 603 and 604) is described by the *PowerPC 60x Microprocessor Interface Specification* [18]. The processor interface for the 64-bit processors is described by the *PowerPC 6xx Bus Definition* [19]. Individual implementations may subset these specifications, provided these variations are described in their respective user's manuals. The processor interface for the 603 family has been optimized for portable applications by keeping address-only cycles internal to the processor. This optimization leads to functional restrictions which must be understood by the platform and system software providers.

Requirements:

4–12. **For the Symmetric Multiprocessor option:** The 603 family of processors must not be used in a symmetric multiprocessing complex.

4–13. The 603 family of processors must be used only with system logic that does not reorder data transfers.

4.1.4 PowerPC Architecture Features Deserving Comment

Some PowerPC architecture features are optional, and so need not be implemented in a platform. Usage of others may be discouraged due to their potential for poor performance. The following sections elaborate on the disposition of these features in regard to compliance with the PowerPC architecture.

4.1.4.1 Unaligned Little-Endian Scalar Operations

The initial set of PowerPC processor designs did not support unaligned Little-Endian scalar operations in hardware, but interrupted to the system alignment

Personal Use Copy - Not for Reproduction

handler to provide the opportunity for the operating system to emulate these operations. Later implementations of the architecture have added hardware support for these operations.

Requirements:

4–14. Operating systems must provide an alignment interrupt handler which correctly emulates the execution of any unaligned LE accesses that are not implemented in hardware, except those which may be generated by *lwarx, ldarx, stwcx., stdcx., eciwx, ecowx,* and the multiple scalar operations (see 4.1.4.2, "Little-Endian Multiple Scalar Operations," on page 54).

Software Implementation Note: Because of performance impacts, software is strongly discouraged from using unaligned Little-Endian scalar operations.

Hardware Implementation Note: Because legacy software generates unaligned Little-Endian scalar operations, hardware is not exempted from supporting these operations.

4.1.4.2 Little-Endian Multiple Scalar Operations

The PowerPC architecture does not support multiple scalar operations in Little-Endian mode. The multiple scalar operations are *Load and Store String* and *Load and Store Multiple*.

Requirements:

4–15. Software must not use multiple scalar operations in LE mode. Results of multiple scalar operations in LE mode are undefined.

Architecture Note: Multiple scalar operations in LE mode actually generate an alignment interrupt instead of a program interrupt. They are, nonetheless, intended to be unsupported.

4.1.4.3 Direct-Store Segment Support

The CHRP architecture does not require platforms to support direct-store segments. However, direct-store segments may be required for some I/O bus extensions to the CHRP architecture.

Personal Use Copy - Not for Reproduction

Requirements:

4–16. Platforms must not use direct-store segments to implement interfaces defined by the CHRP architecture.

4–17. Operating systems must not depend on direct-store segment support when using interfaces specified by the CHRP architecture.

4.1.4.4 Big-Endian Multiple Scalar Operations

The PowerPC architecture supports multiple scalar operations in Big-Endian mode. The multiple scalar operations are *Load and Store String* and *Load and Store Multiple*. In future PowerPC processor implementations, these instructions are likely to have greater latency and take longer to execute, perhaps much longer, than a sequence of individual *Load* or *Store* instructions.

Software Implementation Note: Because of the long-term performance disadvantage associated with multiple scalar operations, their use by software is not recommended. However, there is a window of system implementations in which store multiples will provide the highest bandwidth from the processor to I/O. Therefore, with the knowledge of imminent need to rewrite some code, certain developers (especially graphics device driver developers) may choose to use store multiples. Note that the PowerPC architecture does not guarantee order among data stored by a *Store Multiple* instruction.

4.1.4.5 External Control Instructions (Optional)

The external control instructions (*eciwx* and *ecowx*) provide a mechanism for non-privileged code to interact with external devices which require a knowledge of real, rather than effective, addresses. Together with monitoring of translation activity by the external hardware, they provide a means for external access to system memory without pinning that memory.

Software Implementation Note: Operating systems should isolate applications from the external control instructions by providing them in service libraries, and may also use them in device drivers.

Hardware Implementation Note: Platforms are not required to support the external control instructions.

Personal Use Copy - Not for Reproduction

Requirements:

4–18. If the external control facility defined by the PowerPC architecture is supported, that support will be described as properties of the appropriate nodes (for example, memory controller and host bridge) of the OF device tree. See *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10] for more details.

4–19. Platforms must not use external control instructions as the sole interface to functions specified by the CHRP architecture.

4–20. Operating systems must not require the external control instructions when using interfaces specified by the CHRP architecture.

4.2 Memory Architecture

The Memory Architecture of a Common Hardware Reference Platform implementation is defined by *The PowerPC Architecture* [1] and the System Memory, Storage Ordering Models, Memory Controller, and Cache Memory sections which follow, as well as Chapter 3, "System Address Map," on page 23, which defines what platform elements are accessed by each real (physical) system address.

4.2.1 System Memory

System Memory normally consists of dynamic read/write random access memory which is used for the temporary storage of programs and data being operated on by the processor(s). A platform usually provides for the expansion of System Memory via plug-in memory modules and/or memory boards.

Requirements:

4–21. Platforms must provide at least 8 MB of System Memory. (Also see Chapter 3, "System Address Map," on page 23 for other requirements which apply to memory within the first 16 MB of System Memory.)

4–22. Platforms must support the expansion of System Memory to 32 MB or more.

Personal Use Copy - Not for Reproduction

Hardware Implementation Note: These requirements are minimum requirements. Each operating system has its own recommended configuration which may be greater.

Software Implementation Note: System Memory will be described by the properties of the *memory* node(s) of the Open Firmware (OF) device tree.

4.2.2 Storage Ordering Models

For "correct" execution of a program, correct ordering of its storage accesses is essential. Most programs are written with a "sequential" ordering (also called "serial," or "strong" ordering) assumption or model. In this type of ordering, *Load*s and *Store*s are performed in the order specified by the program. The satisfaction of this assumption is guaranteed by the hardware.

The sequential execution model in the PowerPC architecture requires that *Load*s and *Store*s issued by a program appear to the program to be strongly ordered. (See requirement 4–27.) However, it is not necessary that those accesses complete in storage, or are viewed by other processors or mechanisms, in the same order. This is called a *weakly ordered* storage model. Weak ordering can lead to performance improvements in complex systems by allowing a processor to consider its *Load* and *Store* instructions complete before the corresponding storage operations are performed with respect to other processors and mechanisms.

4.2.2.1 Memory Coherence

Coherence refers to the ordering of writes to a single location. Specifically, a location is said to be coherent if no two processors or I/O observe any subset of a series of *Store*s to that location to be occurring in mutually conflicting orders. The *PowerPC Architecture* [1] provides specifications for implementing coherence among caches and physical memory for System Memory locations within a symmetric multiprocessor.

The PowerPC architecture specifies accesses (*Load*s and *Store*s) to bytes, half-words aligned on half-word boundaries, words aligned on word boundaries, and, for 64-bit implementations of the architecture, also, double-words aligned on a double-word boundaries, as being "single-copy atomic." A single-copy atomic operation is performed without permitting other processors or mechanisms to access the target locations from the beginning to the end of the operation. Accesses to data of all other sizes can be non-single copy atomic.

Single copy atomic *Store*s to a given location are coherent if they are serialized in some order, and no processor is able to observe any subset of those

Personal Use Copy - Not for Reproduction

*Store*s in a conflicting order. This serialization results in a sequence of values assigned to the location. For a location that is cachable, the unique sequence is defined over values assumed not only by the physical memory location, but also by copies of the location in caches. It is not necessary that the physical memory location assume each value in the sequence. For example, if a processor has a write-back cache, it may update a location several times before the value is written to the physical memory. Not every value written is visible to another processor. However, if the location is coherent, the order of values seen by another processor is compatible with the order in which *Store*s are made.[1] Thus, for example, a processor can never *Load* a "newer" value first and then, later, *Load* an "older" value.

A platform may maintain coherence on blocks of data, referred to as *coherency blocks*, that are larger than data-sizes prescribed for single-copy atomic accesses. Typically, the coherency block is of the size of a cache line, but it can be smaller. The coherency block is treated as a whole, or as a single location, with respect to values it assumes, and these values form a unique sequence of which all the processors and I/O observe a subset.

The PowerPC architecture allows assignment of the hardware coherence property on a per-page (4 KB) basis. If a given page of physical memory is declared as Memory Coherence Required[2] (M-bit equals 1), all coherency blocks within that page are kept coherent. It is permitted by the PowerPC architecture, however, that a page in physical memory may be accessed via different virtual addresses that have different coherence property specifications. This is called "M-bit aliasing." Similar mismatches are also permitted by the PowerPC architecture among Caching Inhibited (I-bit), Write Through (W-bit), and Guarded Storage (G-bit) properties. Under certain circumstances, the PowerPC architecture disallows the assumption of hardware-maintained coherence when a location is accessed with aliased W, I, or M bits.[3]

Requirements:

4–23. Platforms must provide the ability to maintain System Memory Coherence.

4–24. I/O transactions to System Memory through a Host Bridge must be made with coherence required.

---

[1] See the section entitled "Memory Coherence" in Book II, *The PowerPC Architecture* [1].

[2] See the section entitled "Storage Access Modes" in Book III, *The PowerPC Architecture* [1].

[3] *ibid.*

Personal Use Copy - Not for Reproduction

Hardware Implementation Note: For some system buses, enabling a transaction for snooping is a method for invoking the coherence mechanism. For these buses, requirement 4–24 translates into a requirement that all I/O transactions to system memory be enabled for snooping.

Hardware Implementation Note: Some processors may ignore the M=1 setting for a line when it is accessed as a result of an instruction-fetch miss. However, if the line contains both instructions and data, there may be subsequent data accesses to that line which are required to be coherent. In this case, the non-coherent copy of the line must be discarded and the line must be refetched coherently. I/O writes to the line must also be performed coherently with respect to copies of the line contained in data caches or in combined caches.

Memory, other than System Memory, is not required to be coherent. Such memory may include memory in I/O devices.

4.2.2.2 Consistency Model

Consistency refers to the ordering of accesses to more than one location. In systems which support a *sequentially consistent* memory model, *Load*s and *Store*s performed by a program not only appear to be strongly ordered to the program executing them, but are also observed by concurrent programs on other processors as being performed in the same order. This is not true of a system that employs a *weakly consistent* memory model. In a weakly consistent memory model, weak ordering is the default for storage accesses.

Weakly ordered storage accesses can lead to a temporarily inconsistent view of memory among processors. For example, suppose a processor performs a *Store* to location A, followed by a *Store* to location B, then for some finite period of time, it is possible that if another processor or mechanism reads the new value of B, it can subsequently perform a read on location A and yet receive the old value of A — a view of memory that is inconsistent with that of the program which performed the two store operations.

However, with mutually inconsistent views of memory, cooperating programs cannot communicate correctly and, therefore, cannot compute correctly. Such programs must ensure that a consistent view of memory is attained before they start to communicate. An ability to enforce a sequential consistency on demand is therefore essential.[1]

[1] Note that cooperating concurrent programs running on a system that enforces sequential consistency for all storage accesses always execute correctly.

Personal Use Copy - Not for Reproduction

The PowerPC architecture supports a weakly consistent memory model. To enforce sequential consistency on demand, it provides ordering instructions: *eieio, sync,* and *isync*. *Eieio* guarantees that qualifying storage accesses separated by it get performed in the order they are issued by the program. *Sync* and *isync* instructions guarantee that when they complete, all coherent storage accesses issued prior to them have been performed with respect to all processors and mechanisms that accesses those locations coherently. Thus, using these instructions, it is possible for communicating programs to guarantee that accesses before and after these instructions are observed to have performed in a mutually consistent order.

Most platforms based on the CHRP architecture are expected to incorporate the weakly consistent memory model as specified by the PowerPC architecture.

Requirements:

4–25. Software must assume only the Weakly Consistent storage model.

4–26. Platforms must guarantee that a processor's accesses to the same location are kept strongly ordered, unless the location is accessed by the processor with WIM-bit aliasing which prohibits the assumption of hardware-maintained coherence.

As specified in the PowerPC architecture, software must use ordering instructions to impose strong ordering between accesses to different physical memory locations, coherent or otherwise, when it has a dependency on the order of accesses being strong. Self modifying code, if used, must be written in such a fashion as to ensure that the instruction fetch pipeline and the cache do not contain the original code once the modification has been made.

Software Implementation Note: Explicit ordering must be employed if self modifying code is used. The following sequence of instructions is one way to accomplish this: *dcbst, sync, icbi, sync, isync*.

4.2.2.3 Storage Ordering and I/O

Looking out from a processor, there are interfaces and bridge devices beyond which the ordering operations defined by the PowerPC architecture cannot be propagated. Such interfaces or bridges form the boundary of the system which encloses the PowerPC *coherency domain* which satisfies the PowerPC coherency architecture.

Figure 10 on page 61 shows an example system. The shaded portion is the PowerPC coherency domain. Buses 1 through 3 lie outside this domain. The figure shows two I/O subsystems, each interfacing with the host system via a

Personal Use Copy - Not for Reproduction

Host Bridge. Notice that the boundary is shown to traverse through the Host Bridges. This marks the usually identifiable boundary inside the Host Bridge hardware. On one side (the shaded portion) of this boundary PowerPC architectural semantics are in force, and on the other side lies the logic that interfaces with protocols of the I/O interconnect.

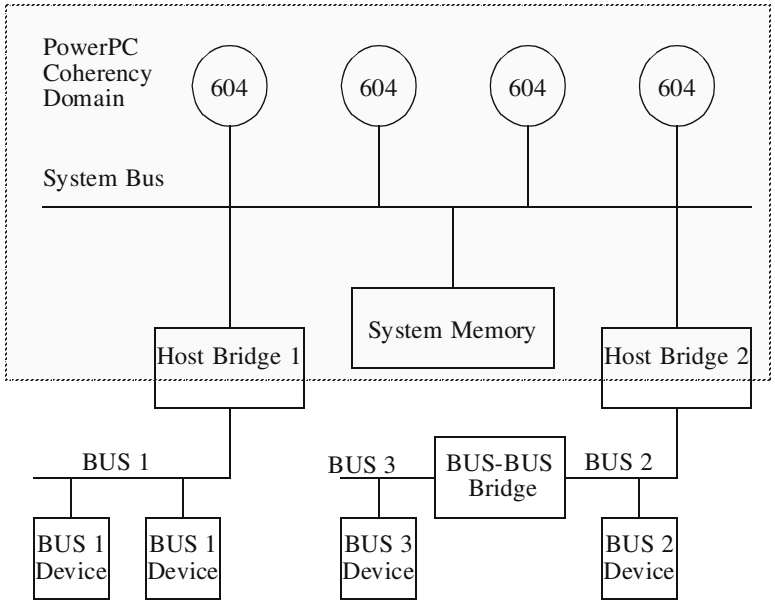

Figure 10. Example System Diagram Showing the PowerPC Ordering Domain

The fundamental ordering rule for the PowerPC sequential execution model is that if a processor performs a *Store* operation to a location followed by a *Load* to the same location, and if it is the only processor storing to that location, then the *Load* must return the value deposited by the *Store* operation. This ordering rule might be violated outside the coherency domain if I/O devices within the same I/O subsystem are allowed to make concurrent accesses to the location. The burden of guaranteeing satisfaction of this sequential ordering

Personal Use Copy - Not for Reproduction

rule, therefore, is on both system hardware and software. For a more detailed discussion see Section 5.1.2.3, "PCI Delayed Read Interaction," on page 74.

Accesses outside the coherency domain are assumed to be made to I/O devices. These accesses are considered performed (or complete) when they complete at the device's I/O bus interface.

It is not sufficient to ensure strong ordering on a processor's accesses to a single location outside the coherency domain. It is also important to able to ensure that *Store*s to a number of different locations outside the coherency domain are completed in a strongly ordered manner. For example, suppose some data has to be deposited in an I/O device before the device's control register is written to start its operation. It is essential, in this case, that the *Store*s that deposit the data complete before the *Store* that writes into the control register. For this reason, semantics of *eieio* must be extended beyond the coherency domain. Because the PowerPC ordering operations do not travel beyond the PowerPC coherency domain, system hardware must take appropriate action to enforce the ordering semantics of *eieio* on accesses leaving the domain. See Chapter 5, "I/O Bridges," on page 69.

Requirements:

4–27. Platforms must guarantee that, for a particular processor, accesses to the same location beyond the PowerPC coherency domain are performed in a strongly ordered manner, given that no I/O device is allowed to make concurrent accesses to the location. Note that it is not sufficient to merely produce an appearance of strong ordering with respect to the processor performing the accesses.

4–28. Platforms must guarantee that the storage ordering semantics of *eieio* are preserved for accesses leaving the PowerPC coherency domain at any given host bridge, all the way to destination I/O devices. That is, *Load* and *Store* accesses (in any combination) by a processor to an I/O device which are separated by an *eieio*, must complete in the same order that the processor issued those accesses.

Apart from the ordering disciplines stated in requirements 4–27 and 4–28, no other ordering discipline is guaranteed by the system hardware for *Load*s and *Store*s performed by a processor to locations outside the PowerPC coherency domain. Any other ordering discipline, if necessary, must be enforced by software via programming means.

Software Implementation Note: For example, if software wants to initiate a peer-to-peer operation only after a number of *Store* operations it issued to the device have completed, it can issue a sequence of

Personal Use Copy - Not for Reproduction

instructions such as follows: *Store* <Location A>, *Store* <Location B>, *Store* <Location C>, *eieio*, *Load* <Location C>, *eieio*, where locations A, B, and C are in the device and are assumed to have a "cache-inhibited and guarded" mapping. After issuing this sequence the software can start the peer-to-peer operation. The first *eieio* guarantees that all the *Store*s are sent before the *Load* is sent. The second *eieio* guarantees that all subsequent storage operations are not sent until the *Load* completes at the processor, at which point the program can be certain that all the previous *Store*s to device are also complete.

The elements of a system outside its coherency domain are not expected to issue explicit PowerPC ordering operations. System hardware must therefore take appropriate action to impose ordering disciplines on storage accesses entering the coherency domain. A strong-ordering rule is enforced on an I/O device's accesses to the same location. Write operations from the same source are completed in a sequentially consistent manner.

Requirements:

4–29. Platforms must guarantee that accesses entering the PowerPC coherency domain that are from the same I/O device and to the same location are completed in a sequentially consistent manner.

4–30. Platforms must guarantee that multiple write operations entering the PowerPC coherency domain that are issued by the same I/O device are completed in a sequentially consistent manner.

It is the responsibility of the host bridges to guarantee that requirements 4–27 through 4–30 are satisfied. A host bridge must take into account the particular I/O interconnection network that it interfaces to and the protocol characteristics and interfaces of the I/O buses involved, which will influence the actual set of ordering rules that the above requirements translate into and that the host bridge implements. For example, Section 5.1.2, "Data Buffering and Instruction Queuing," on page 70 specifies ordering rules for a PCI host bridge that interfaces to a PCI-based I/O tree.

4.2.2.3.1 Storage Ordering and I/O Interrupts

The conclusion of I/O operations is often communicated to processors via interrupts. For example, at the end of a DMA operation that deposits data in the system memory, the device performing the operation might send an interrupt to the processor. Arrival of the interrupt, however, is no guarantee that all the data has actually been deposited; some might be on its way. The receiving program

Personal Use Copy - Not for Reproduction

must not attempt to read the data from the memory before ensuring that all the data has indeed been deposited. There may be system and I/O subsystem specific method for guaranteeing this. See Section 5.1.2.2, “DMA Ordering,” on page 73.

4.2.2.4 Atomic Update Model

An update of a memory location by a processor, involving a *Load* followed by a *Store*, can be considered “atomic” if there are no intervening *Store*s to that location from another processor or mechanism. The PowerPC architecture provides primitives in the form of *Load And Reserve* and *Store Conditional* instructions which can be used to determine if the update was indeed atomic. These primitives can be used to emulate operations such as “atomic read-modify-write” and “atomic fetch-and-add.” Operation of the atomic update primitives is based on the concept of “Reservation,”[1] which is supported in a CHRP system via the coherence mechanism.

Requirements:

4–31. The *Load And Reserve* and *Store Conditional* instructions must not be assumed to be supported for Write-Through storage.

Software Implementation Note: To emulate an atomic read-modify-write operation, the instruction pair must access the same storage location, and the location must have the Memory Coherence Required attribute.

Hardware Implementation Note: The reservation protocol is defined in Book II of the *PowerPC Architecture* [1] for atomic updates to locations in the same coherency domain.

4.2.3 Memory Controllers

A *Memory Controller* responds to the real (physical) addresses produced by a processor or a host bridge for accesses to System Memory. It is responsible for handling the translation from these addresses to the physical memory modules within its configured domain of control.

[1] See the section entitled “Synchronization” in Appendix E of Book I, *The PowerPC Architecture* [1], and the section entitled “Atomic Update Primitives” in Book II, *The PowerPC Architecture* [1]

Personal Use Copy - Not for Reproduction

Requirements:

4–32. Memory controller(s) must support the accessing of System Memory as defined in Chapter 3, "System Address Map," on page 23.

4–33. Memory controller(s) must be fully initialized and set to full power mode prior to the transfer of control to the operating system. This requirement applies to normal boot and wakeup. (See Chapter 11, "Power Management," on page 185 for an explanation of these terms.)

4–34. All allocations of System Memory space among memory controllers must have been done prior to the transfer of control to the operating system.

4–35. Memory controller(s) must maintain System Memory in Big-Endian byte order when the system is operating in Big-Endian mode (MSR_{LE}=0) and in Little-Endian byte order when the system is operating in Little-Endian mode (MSR_{LE}=1). Whether System Memory is maintained in "True Little-Endian" or "PowerPC Little-Endian"[1] form while in Little-Endian mode is platform dependent, but must be transparent to software. See Section 2.3, "Bi-Endian Support," on page 14 and Appendix C, "Bi-Endian Designs," on page 265 for more information on Bi-Endian designs.

Software Implementation Note: Memory controller(s) are described by properties of the *memory-controller* node(s) of the OF device tree.

4.2.4 Cache Memory

All of the PowerPC microprocessors include some amount of on-chip or *internal cache* memory. The CHRP architecture allows for cache memory which is external to the processor chip, and this *external cache* memory forms an extension to internal cache memory.

Requirements:

4–36. All caches must meet the requirements of the CHRP architecture coherency model, as stated in "Storage Ordering Models" on page 57.

4–37. **For the Symmetric Multiprocessor or Power Management option:** Each cache must be able to be completely flushed and invalidated via

[1] See the section entitled "PowerPC Little-Endian Byte Ordering" in Appendix D of Book I of *The PowerPC Architecture* [1], for an explanation of PowerPC Little-Endian byte ordering.

Personal Use Copy - Not for Reproduction

the RTAS *cache-control* function. (See Section 7.3.10.1, “Cache-control,” on page 136 for more information.)

4–38. For the Power Management option: External caches must be able to be placed in low power or disabled states via the RTAS *cache-control* function.

4–39. For the Symmetric Multiprocessor or Power Management option: To ensure compatibility with all CHRP implementations, operating systems must use the RTAS *cache-control* function to flush an entire cache rather than code directly to any specific system or processor implementation.

4–40. If a platform implementation elects not to cache portions of the address map in all external levels of the cache hierarchy, the result of not doing so must be transparent to the operation of the software, other than a difference in performance.

4–41. All caches must be fully initialized and enabled, and they must have accurate state bits prior to the transfer of control to the operating system.

4–42. If an in-line external cache is used, it must support one reservation as defined for the *Load And Reserve* and *Store Conditional* instructions.

4–43. For the Symmetric Multiprocessor or Power Management option: Platforms must implement their cache hierarchy such that all caches at a given level in the cache hierarchy can be flushed and disabled before any caches at the next level which may cache the same data are flushed and disabled (that is, L1 first, then L2, and so on).

4–44. For the Symmetric Multiprocessor or Power Management option: If a cache implements snarfing, then the cache must be capable of disabling the snarfing during flushing in order to implement the RTAS *cache-control* function in an atomic way.

4–45. Software must not depend on being able to change a cache from copy-back to write-through.

Software Implementation Note: Each first level cache will be defined via properties of the *cpu* node(s) of the OF device tree. Each higher level cache will be defined via properties of the *l2-cache* node(s) of the OF device tree. See the *PowerPC processor binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [11] for more details.

Personal Use Copy - Not for Reproduction

Software Implementation Note: To ensure proper operation, cache(s) at the same level in the cache hierarchy should be flushed and disabled before cache(s) at the next level (that is, L1 first, then L2, and so on).

Hardware Implementation Note: The PowerPC architecture would seem to imply that all caches can be controlled via the cache control and prefetching instructions and the page table attribute bits. In fact, the vast majority of external caches are not affected by these attributes of the PowerPC architecture and, in general, depend solely on a fixed address boundary check to determine cachability. Internal caches are expected to comply with the PowerPC processor architecture. It is the responsibility of the platform designer to verify that the allowed flexibility in external cache function affects only the performance and not the programming model (coherency, for example) of the platform.

Hardware Implementation Note: There may be a significant performance impact to some operating systems if parts of an OS's System ROM and/or a local bus video frame buffer are not actually cached outside the processor, for example, in an external Level 2 cache. The CHRP architecture permits this cost vs. performance trade-off to be made by hardware vendors, since software compatibility is, by definition, not affected.

Personal Use Copy - Not for Reproduction

Blank page when cut - 68

I/O Bridges

5

There is at least one I/O bridge in a Common Hardware Reference Platform. That is, there will be at least one bridge which interfaces to the system bus on the processor side, and interfaces to the Peripheral Component Interface (PCI) bus on the other. This bridge is called the PCI Host Bridge (PHB). The architecture for PHBs is defined in this chapter. In addition, there may be other bridge components in the platform to bridge from one I/O bus to some other or to the same I/O bus. For example, to bridge from the PCI bus to an ISA bus. Requirements for these other bridges will be discussed, as appropriate, in this chapter.

5.1 PCI Host Bridge (PHB) Architecture

The PHB architecture places certain requirements on PHBs. There should be no conflict between this document and the *PCI Local Bus Specification,* revision level 2.1 [14] document, but if there is, the PCI documentation takes precedence. The intent of this architecture is to provide a base architectural level which supports the PCI architecture and to provide optional constructs which allow for use of 32-bit PCI devices in platforms with greater than 4 GB of system addressability.

Requirements:

5–1. All PHB implementations must be compliant with the *PCI Local Bus Specification,* revision level 2.1 [14].

5–2. All requirements defined in Chapter 3, “System Address Map,” on page 23 for HBs must be implemented by all PHBs in the platform.

5–3. HB0 must be a PHB (PHB0).

5.1.1 PHB Implementation Options

There are a few implementation options when it comes to implementing a PHB. Some of these become requirements, depending on the characteristics of the system for which the PHB is being designed. Requirement 5–2, which states that all requirements defined in Chapter 3, "System Address Map," on page 23 for HBs must be implemented by all PHBs, defines the requirements for the following:

- The 64-bit addressing option requires that PHBs designed for use in platforms which are designed to support System Memory configured at an address of 4 GB or above must provide the TCE address translation mechanism in order to give 32-bit PCI devices in such platforms the capability to address all of System Memory directly. The existence of this 64-bit addressing option support is reported by the existence of the *64-bit-addressing* property in the PHB node(s) of the OF device tree, and the 64-bit addressing option can be enabled by the operating system by the *set-64-bit-addressing* OF method.
- Platforms which implement the 64-bit addressing option may optionally support the PCI Dual Address Cycle (DAC) capability during DMA operations. The existence of this DAC option support is reported by the existence of the *64-bit-dma* property in the PHB node(s) of the OF device tree, and this option is enabled along with the 64-bit addressing option by the operating system by the *set-64-bit-addressing* OF method.
- PHB implementations (for 32-bit or 64-bit addressing systems) may choose to implement one or both of the compatibility holes, as defined in Chapter 3, "System Address Map," on page 23.

5.1.2 Data Buffering and Instruction Queuing

Some PHB implementations may include buffers or queues for DMA, *Load*, and *Store* operations. The PowerPC architecture provides several instructions, including *eieio* and *sync,* which allow programs to order and synchronize operations to System Memory or I/O. These instructions may or may not propagate (depending on the processor and platform implementation) down to the PHB(s), but will definitely not propagate beyond a PHB. As far as *eieio* is concerned, where the propagation stops is immaterial, and the platform will keep the programming model the same (see requirement 5–4, below). The *sync*

Personal Use Copy - Not for Reproduction

instruction, however, cannot be propagated out to the end I/O device because the PCI bus does not lend itself to this capability. What this means is that for the cases where the programming model is dependent on completion of a write to an I/O device before continuing, software must take care of the synchronization. For example, the PCI architecture specifies that a read operation issued to a PCI device will flush the PCI and PHB data buffers. This can be used, then, to assure completion of a write to a device by doing a read of something on the device after writing to the device. This will force the write data to arrive at the device before the read data is returned (within the restrictions detailed in section 5.1.2.3, "PCI Delayed Read Interaction," on page 74). In addition, interrupts from a PCI device can bypass data written by the PCI device, and it may be necessary to use a similar technique to guarantee that data written by a device is forced out of data buffers on the I/O side of the PHB prior to using the data.

Most processor accesses to System Memory go through the processor data cache. When sharing System Memory with I/O devices, hardware must maintain consistency with the processor data cache and the System Memory, as defined by the requirements in Section 4.2.2.1, "Memory Coherence," on page 57.

Requirements:

5–4. PHB implementations which include buffers or queues for DMA, *Load*, and *Store* operations must make sure that these are transparent to the software, with a few exceptions which are allowed by the PCI architecture, by the PowerPC architecture, and in Section 4.2.2.1, "Memory Coherence," on page 57.

Since the implications of combining the PowerPC ordering rules and the PCI ordering rules and the exceptions that they allow are not obvious, the following sections will describe the combination of these rules and the resulting requirements on the PHB.

5.1.2.1 *Load* and *Store* Ordering

Although it is possible to create a PHB which would emulate an *eieio* instruction, such a PHB would be unnecessarily complex. By combining the PowerPC and PCI ordering rules and implementing those rules, a PHB can maintain the *eieio* programming model without having to put an *eieio* instruction in its instruction queue. The following are the requirements on the PHB in order to meet these other architectures' ordering rules.

Personal Use Copy - Not for Reproduction

Requirements:

5–5. A *Load* or *Store* to either the Peripheral Memory Space or the Peripheral I/O Space of a PHB must never be passed to the I/O bus before a previous *Store* to either the Peripheral Memory Space or the Peripheral I/O Space of that same PHB (that is, multiple *Stores* to the I/O bus generated by one PHB must be kept in order and a *Load* must not pass a *Store*).

5–6. A *Load* to either the Peripheral Memory Space or the Peripheral I/O Space of a PHB must never be passed to the I/O bus before a previous *Load* to either the Peripheral Memory Space or the Peripheral I/O Space of that same PHB when both of those *Loads* go to the exact same address.

With the exception of the one case stated in requirement 5–6 for two *Load* operations to the same address, a *Store* or *Load* to either the Peripheral Memory or Peripheral I/O Spaces is allowed to pass a previous *Load*, but is not required to do so. This works because when the software is concerned about the ordering of these, it will place an *eieio* after the *Load*, and the processor will not issue the following *Load* or *Store* until the data is returned from the previous *Load* (that is, the processor handles the ordering in this case when the software cares about the ordering). The reason for requirement 5–6 is that the read operations on the PCI bus are not tagged as to which device requested the read, and if one *Load* (read) is issued to the PCI bus and retried and a *Load* to the same address issued by another processor is allowed to pass this retried *Load*, then the second processor could get the data that was requested by the first processor. Even this would not be a problem, providing the second processor did not first do a *Store* to that address followed by a *Load*. In this case, the *Load* would pick up the data from the first processor's *Load*, and therefore would get back something different than what it thought it had previously sent with its *Store* (that is, without requirement 5–6 the sequence of *Load* from processor 1, followed by a *Store* by processor 2, followed by a *Load* by processor 2, for example, could return the wrong data to processor 2). Of course, since it is not a requirement for a *Store* or *Load* to either the Peripheral Memory or Peripheral I/O Spaces to be allowed to pass a previous *Load*, one solution (and legitimate implementation as an implementation cost versus performance trade-off) is to never allow any *Load* or *Store* to pass any other *Load* or *Store* in a PHB's instruction queue (that is, order all *Load* and *Store* operations from the processors).

Personal Use Copy - Not for Reproduction

5.1.2.2 DMA Ordering

There are certain ordering rules that DMA operations must follow. In general, the ordering for the DMA path operations from the I/O bus to the processor side of the PHB is independent from the *Load* and *Store* path, with one exception as stated in requirement 5–11 below, and, indeed must be independent in the cases which are stated in requirements 5–7 and 5–8, below. Note that in the requirements, below, a *read request* is the initial request to the PHB and the *read completion* is the data phase of the transaction (that is, the data is returned). The PHB may signal a "retry" on the initial read request and later provide the data when the requester comes back to get it (that is, when the device requests it again). Thus, the read request and read completion may occur during different PCI bus transactions.

Requirements:

5–7. Data from a DMA read completion must be allowed to complete prior to *Load* or *Store* operation which was previously queued in the PHB, in order to prevent a possible deadlock.

5–8. *Load* data buffered in a PHB must not prevent a subsequent DMA write request from being posted into the PHB or from making progress through the PHB, in order to prevent a possible deadlock on the PCI bus.

5–9. A DMA write or read request from an I/O device to the processor side of the PHB must never be passed to the processor side of the PHB before the data from a previous I/O DMA write operation has been flushed to the processor side.

5–10. A previous DMA read request accepted by a PHB but not yet completed must not prevent a subsequent DMA write request from being posted into the PHB or from making progress through the PHB, in order to prevent a possible deadlock on the PCI bus.

5–11. All DMA write data (destined for the processor side of the PHB) in the PHB buffers must be flushed out of the PHB prior to delivering data from a *Load* operation which has come after the DMA write operations.

Requirement 5–11 is the one case where the *Load* and *Store* path is coupled to the DMA path. This requirement guarantees that the software has a method for forcing DMA write data out of any buffers in the path prior to servicing a completion interrupt from the device. Note that the device can do that, also, via

Personal Use Copy - Not for Reproduction

requirement 5–9, by following any write by a read through the same path prior to issuing a completion interrupt.

A DMA read operation is allowed to be processed prior to the completion of a previous DMA read operation, but is not required to be.

5.1.2.3 PCI Delayed Read Interaction

As mentioned previously, read operations on the PCI bus are not tagged as to which device requested the read. For PCI devices designed prior to version 2.1 of the *PCI Local Bus Specification* [14], this did not matter, since the PCI architecture did not allow delayed read operations. Version 2.1 does allow delayed read operations (a delayed read operation is one where the target device retries the PCI master, but goes off and does the read operation anyway, so that it can have the data ready when the master comes back and tries the operation again). This means that if two devices on the PCI bus (one of which could be the PHB) are reading from the exact same address (that is, the exact same byte address), and if one or both of those devices can write the data at that address, then the device(s) doing the writing can get stale data upon a read. This has implications in programming certain I/O operations. Several scenarios can occur:

1. Two (or more) processors reading the same address on a PCI device, with one of those device drivers also writing that address.
2. Two peer PCI devices reading the same address on another PCI device, with one of those devices also writing that address.
3. Two peer PCI devices reading the same address in System Memory, with one of those devices also writing that address.
4. The device driver (where the PHB acts as a PCI master on behalf of the device driver) and a PCI I/O device reading the same address on another PCI device, with one of those also writing that address.

Except for scenario 1, software must create the appropriate protocols to assure that the problem does not occur. Scenario 1 is prevented from happening by requirement 5–6.

5.1.3 Byte Ordering Conventions

A system can run in Little-Endian (LE) or Big-Endian (BE) mode. Depending on which mode the system is operating, the paths for data from the processor to the I/O or the System Memory to the I/O may need to do something special based on the mode of operation. Some of the LE and BE aspects that are unique

Personal Use Copy - Not for Reproduction

to the PHB will be discussed below. For more information about system requirements, see Section 2.3, “Bi-Endian Support,” on page 14 and Appendix C, “Bi-Endian Designs,” on page 265.

The PCI bus itself can be thought of as not inherently having an endianess associated with it (although its numbering convention indicates LE). It is the devices on the PCI bus that can be thought of as having endianess associated with them. Some PCI devices will contain a mode bit to allow them to appear as either a BE or LE device. Some devices will even have multiple mode bits; one for each data path (*Load* and *Store* versus DMA). In addition, some devices may have multiple concurrent apertures, or address ranges, where the device can be accessed as a LE device in one aperture and as a BE device in another.

Requirements:

5–12. The hardware platform must be designed such that software must follow Table 9 on page 76 while running in the various processor modes (LE = 0 and LE = 1) and issuing *Load* and *Store* operations to various entities with various endianess, including doing any necessary address un-modification when running with LE = 1 (for platforms using processors implementing LE mode via address modification).

5–13. When performing DMA operations through a PHB while running with the processor mode of LE = 1, if the platform is implemented with the true LE format in System Memory, or while running with the processor mode of LE = 0 with BE format in System Memory, then the platform must not modify the data during the transfer process; the lowest addressed byte in System Memory being transferred to the lowest addressed byte on the PCI bus, the second byte in System Memory being transferred as the second byte on the PCI bus, and so on.

5–14. When performing DMA operations through a PHB while running with the processor mode of LE = 1, if the platform is implemented with the PowerPC LE format in System Memory, then the platform must transform the data during the transfer process to true LE format by reflecting the bytes within a doubleword (that is, the DMA is done as though the data is accessed a byte at a time, with the address modified by the same modification as used by the processor for 1-byte *Loads* and *Stores*, namely, exclusive-or the address with 0b111).

Personal Use Copy - Not for Reproduction

Table 9. *Load* and *Store* Programming Considerations

Destination	PowerPC Processor Mode: LE=0 (processor operating in BE mode)	PowerPC Processor Mode: LE=1 (processor operating in LE mode)
BE scalar entity: For example, TCE or BE register in a PCI device	*Load* or *Store*	*Load* or *Store Reverse*
LE scalar entity: For example, LE register in a PCI device or PCI Configuration Registers	*Load* or *Store Reverse*	*Load* or *Store*

Hardware Implementation Note: Requirement 5–14, above, may have implications on the design of a PHB, depending on the hardware platform implementation and component partitioning (see Appendix C, “Bi-Endian Designs,” on page 265 for more information).

Software Implementation Note: When dealing with non-scalar (string) data, BE and true LE data is in System Memory in the same order (MSB at the lowest address). The PCI device will typically treat the data register as a non-scalar register, and thus, as the equivalent of a BE scalar register. If the processor is running in the BE mode, then essentially the BE to BE case in Table 9 on page 76 holds true, and in such cases, programmed transfers of data is accomplished by doing (depending on the direction of data transfer) either a *Load* from the System Memory buffer followed by a *Store* to the I/O device or a *Load* from the I/O device and a *Store* to System Memory. If the processor is running in the LE mode and is implemented using the address modification technique, then the string is stored basically reflected within a doubleword in System Memory. In such cases, a *Load* of something greater than 1-byte will bring the string into the processor register in reverse order (most significant character of the string in the LSB of the register), and when the *Store* is issued to the device, the platform will reverse the order of the bytes within the operation size (as required for scalar data to get true LE data; that is, in order to make the lower right hand corner operation of Table 9 on page 76 come out correctly), and the data will appear with the most significant character of the string on the lowest address byte of the PCI bus. Moves from the I/O to System Memory work in the reverse. In any case, the *Load* or *Store Reverse* instructions are not used for these string move operations.

Personal Use Copy - Not for Reproduction

5.1.4 PCI Bus Protocols

This section details the items from the "PCI Local Bus Specification" and "PCI System Design Guide" documents where there is variability allowed, and therefore further specifications, requirements, or explanations are needed.

5.1.4.1 Peripheral I/O Space

Although an HB in general does not have to implement a Peripheral I/O Space, the PCI bus contains an I/O address space, and therefore requires a Peripheral I/O Space.

Requirements:

5–15. There must be exactly one Peripheral I/O Space per PHB.

5.1.4.2 Dual Address Cycle (DAC)

The DAC capability of the PCI architecture allows PCI devices which are attached to a 32-bit PCI bus to use a 64-bit address (the address phase of the transaction extending to two clocks). PHB support for DAC when the PHB is the target of a DMA operation and the 64-bit addressing option is enabled is optional, but is specified by this architecture (see Chapter 3, "System Address Map," on page 23 and Figure 8 on page 43). PHB support for DAC when the PHB is the master on the PCI bus (that is, support for *Load* or *Store* operations at or above 4 GB to Peripheral Memory Space) is not required and is not specified by this architecture. If DAC DMA capability is supported, this support is reported by the existence of the *64-bit-dma* property in the HB node(s) of the OF device tree, and is enabled along with the 64-bit addressing option by the operating system by the *set-64-bit-addressing* OF method.

5.1.4.3 PCI Interrupt Acknowledge Cycle

The PCI Interrupt Acknowledge Cycle was provided in the PCI architecture for use with bridges which require 8259-like interrupt controller support.

Requirements:

5–16. If a PHB has an interrupt controller on the PCI side of the bridge which requires the PCI Interrupt Acknowledge Cycle generation, then that PHB must provide a 1-byte register which, when read by a processor

Personal Use Copy - Not for Reproduction

using a 1-byte *Load* instruction, will generate a PCI Interrupt Acknowledge cycle on the PCI bus.

If this register exists in a PHB, then so will the *8259-interrupt-acknowledge* property of the root node of the OF device tree, and the address of this register will be passed via this *8259-interrupt-acknowledge* property in the root node.

5.1.5 Programming Model

Normal memory mapped *Load* and *Store* instructions are used to access a PHB's facilities or PCI devices on the I/O side of the PHB. Chapter 3, "System Address Map," on page 23 defines the addressing model. Addresses of I/O devices are passed by OF via the device tree.

Requirements:

5–17. If a PHB defines any registers that are outside of the PCI Configuration space, then the address of those registers must be in the Peripheral Memory Space or Peripheral I/O Space for that PHB, or must be in the System Control Area.

PCI master DMA transfers refer to data transfers between a PCI master device and another PCI device, an ISA device (when ISA is implemented), or System Memory, where the PCI master device supplies the addresses and controls all aspects of the data transfer. Transfers from a PCI master to the PCI I/O Space are essentially ignored by a PHB (except for address parity checking). Transfers from a PCI master to PCI Memory Space are either directed at PCI Memory Space (for peer to peer operations) or need to be directed to the host side of the PHB. DMA transfers directed to the host side of a PHB may be to System Memory or may be to another I/O device via the Peripheral Memory Space of another HB. Transfers that are directed to the Peripheral I/O Space of another HB are considered to be an addressing error (see Chapter 10, "Error and Event Notification," on page 157). For information about decoding these address spaces and the address transforms necessary, see Chapter 3, "System Address Map," on page 23.

5.2 I/O Bus to I/O Bus Bridges

The PCI bus architecture was designed to allow for bridging to other slower speed I/O buses or to another PCI bus. The requirements when bridging from one I/O bus to another I/O bus in the platform are defined below.

Personal Use Copy - Not for Reproduction

Requirements:

5–18. All bridges must comply with the bus specification(s) of the buses to which they are attached.

5.2.1 What Must Talk to What

Since there are a number of different address spaces and multiple buses in the platform, this section describes which devices must be able to access which address spaces.

An *ISA DMA* device is one which requires a DMA controller in the system to provide an address and control the transfer (also called a third party DMA device), whereas an *ISA DMA master* (or I/O bus master) is a device which generates the address and controls the operation.

Requirements:

5–19. Table 10 on page 79 details the minimum requirements that the platform must implement relative to I/O device access to the various address spaces.

Table 10. Which I/O Devices Must Be Able to Access Which Address Spaces

Source Device	Destination Device	Platform Support Required?	Other Requirements and Comments
ISA DMA master	ISA memory	No	
	PCI memory	Limited	Platforms must support this if the io-hole is implemented and the source and destination devices reside on the same side of the same PHB and the operation does not traverse a PCI to PCI bridge and the device is configured in the io-hole (640 KB to (1 MB - 1) address range) and the io-hole is enabled.
	System Memory	Limited	System Memory addresses in the 0 to (16 MB - 1) range only and with a possible hole due to the io-hole (if enabled) or due to ISA devices configured in this range.

Personal Use Copy - Not for Reproduction

Table 10. Which I/O Devices Must Be Able to Access Which Address Spaces ***(Continued)***

Source Device	Destination Device	Platform Support Required?	Other Requirements and Comments
ISA DMA	ISA memory	No	
	PCI memory	Yes	Platforms must support this when the source and destination devices reside on the same side of the same PHB and the operation does not traverse a PCI to PCI bridge. The destination device must be configured either in the BPM to (BIM - 1) range, or if io-hole is enabled, in the 640 KB to (1 MB - 1) address range.
	System Memory	Yes	System Memory addresses in the 0 to (16 MB - 1) range should be accessed via the system-memory-alias area to get around possible holes in the address space due to io-hole being enabled or due to ISA devices configured in this range.
PCI Master	ISA memory	Yes	The PHB as a PCI master must be able to access the full 16 MB ISA Memory Space. Other PCI masters must support this when the ISA memory is in the 640 KB to (1 MB - 1) range and the io-hole is enabled.
	PCI memory	Yes	Platforms must support this when the source and destination devices reside on the same side of the same PHB. In this case, the destination device must be configured either in the BPMn to TPMn (n not equal to 0; BPM0 to (BIM - 1) for PHB0) range, or for PHB0, if io-hole is enabled, may also reside in the 640 KB to (1 MB - 1) address range. Platforms with multiple PHBs may also support this when the destination device resides in the Peripheral Memory Space of a different PHB than the source device.
	System Memory	Yes	For PHB0, if the io-hole is enabled, then the system-memory-alias must be enabled and used to access System Memory addresses in the 640 KB to (1 MB - 1) range.
ISA DMA or DMA Master	ISA or PCI I/O	No	
PCI Master	I/O Space of ISA or PCI	Yes	This is only required when the destination device resides on the same side of the same PHB as the source device (see also requirement 3–6).

Some operating systems may not support all these modes (refer to information available from the operating systems). Examples of what an operating sys-

Personal Use Copy - Not for Reproduction

tem might not support include:

- A PCI device which requires configuration below 1 MB when the operating system does not support the io-hole.
- PCI devices which require a PCI address below 1 MB and which will not configure into the io-hole due to their size may not be configured by the OF or the operating system.
- ISA devices which would cause holes in the ISA DMA master buffer area in System Memory or the corresponding ISA DMA master(s). Therefore system board devices should not be configured outside the io-hole range, and firmware should configure I/O devices in the io-hole range if possible.

ISA to PCI operations cannot traverse a PCI to PCI bridge or else a deadlock condition can occur.

5.2.2 PCI to PCI Bridges

The CHRP architecture allows the use of PCI to PCI bridges in the platform. If the 64-bit addressing option is enabled, the TCEs can be used with the devices attached to the other side of the PCI to PCI bridge when those devices are accessing something on the processor side of the PHB. After configuration, PCI to PCI bridges are basically transparent to the software as far as addressing is concerned (the exception is error handling). For more information, see the *PCI to PCI Bridge Architecture Specification* [16].

Requirements:

5–20. PCI to PCI bridges used on the base platform must implement the architecture as specified in the *PCI to PCI Bridge Architecture Specification* [16].

Hardware Implementation Note: PCI to PCI bridges may limit operations to the PCI I/O space to 16 bits on the side opposite the PHB (that is, addresses of 0 to (64K - 1) are passed through the bridge, but not addresses of 64K and above). Therefore, I/O devices will be limited to the first 64 KB of Peripheral I/O Space (with discontiguous mode disabled) in such cases.

Personal Use Copy - Not for Reproduction

5.2.3 PCI to ISA Bridges

The CHRP architecture allows for at most one ISA bridge attached to one (and only one) PCI bus in a platform. If the 64-bit addressing option is enabled, the TCEs can be used with ISA devices when those devices are accessing something on the processor side of the PHB.

Requirements:

5–21. There must be at most one ISA bus in a platform.

5–22. If there is an ISA bridge on a platform, it must be attached to the I/O side of HB0.

5–23. If there is an ISA bridge on a platform, a DMA controller must be available for the ISA DMA operations, and the DMA controller must be compatible with the register set defined in Chapter 9, "I/O Devices," on page 151.

5–24. OF must program a PCI to ISA bridge such that all addresses for ISA DMA master operations get passed through the PCI to ISA bridge and do not get translated as they pass through the bridge.

5–25. If an ISA device is to participate in PCI to ISA peer to peer operations then the ISA device must be configured in the 640 KB to (1 MB - 1) address range (the io-hole) and the io-hole must be enabled.

5–26. PCI to ISA bridges must do a subtractive decode on the PCI side of the bridge in the PCI Memory Space from 0 to (16 MB - 1) and in the PCI I/O Space from 0 to (64 KB - 1) (that is, they must pass any PCI access in these address ranges to the ISA bus if the PCI cycles in these ranges are not first picked up by another PCI device).

Requirement 5–21 does not imply the number of ISA expansion slots and an ISA bus without slots counts as one ISA bus. It may be possible to meet requirement 5–21 with multiple physical ISA buses, but it must appear to software to be one logical ISA bus, including all programming models and the appearance in the OF device tree as one and only one PCI to ISA bridge.

Since the OF may not know which devices will participate in PCI to ISA peer to peer operations, requirement 5–25 says that OF should configure all ISA devices in the io-hole address range if possible. See also the requirements in Section 5.2.1, "What Must Talk to What," on page 79 for additional requirements.

Personal Use Copy - Not for Reproduction

Hardware Implementation Notes:

- Type F DMA is recommended for multimedia operations.
- The IEEE definition of the ISA bus is in draft form and is called IEEE 996, *A Standard for an Extended Personal Computer Back Plane Bus* [13].
- The *PCI to PCI Bridge Architecture Specification* [16] does not support the attachment of PCI to ISA bridges to the PHB through a PCI to PCI bridge, and does not support ISA DMA or ISA DMA master transactions through a PCI to PCI bridge to another PCI device.

5.2.4 16-Bit PC Card (PCMCIA) and Cardbus PC Card Bridges

The CHRP architecture allows for 16-bit PC Card and Cardbus PC Card bridges. If the 64-bit addressing option is enabled, the TCEs can be used with 16-bit PC Card or Cardbus PC Card devices. For more information on PC Cards, see the *PC Card Standard* specification [17].

If a platform supports 16-bit PC Card or Cardbus PC Card cards, then the following requirement applies.

Requirements:

5–27. A platform which supports Cardbus PC Card devices must also support 16-bit PC Card devices.

Personal Use Copy - Not for Reproduction

Blank page when cut - 84

Interrupt Controller 6

This chapter specifies the requirements for the CHRP interrupt controller. It also proposes a distributed implementation of the interrupt controller.

6.1 Interrupt Controller Architecture

The Open PIC Multiprocessor Interrupt Controller Register Interface Specification[1] is the basis of the CHRP interrupt controller architecture. The Open PIC Specification contains a register-level architectural definition of an interrupt controller that supports up to 32 processors. The Open PIC specification defines means for assigning properties such as priority, vector, destination, etc., to I/O and interprocessor interrupts, as well as an interface for presenting them to processors. It supports both specific and distributed methods for interrupt delivery. The interrupt controller also provides timer-interrupt facilities and means to facilitate assertion of "Initialize" signals to processors.

Requirements:

6–1. Platforms must implement interrupt controllers that are in register-level architectural compliance with *Open PIC Multiprocessor Interrupt Controller Register Interface Specification,* Revision 1.2 [8] including its PowerPC architecture appendix.

[1] The PowerPC architecture Appendix in the Open PIC Specification Revision 1.2 defines options for PowerPC-based systems. However, more options are currently in the process of being included in the said appendix and are expected to appear in future revisions of the specification.

6–2. The Interrupt Acknowledge register implemented in the CHRP interrupt controller.

6–3. Platforms must make per-processor registers available in the “publicly accessible area” of the Open PIC register map.

6–4. All interrupt controller registers must be accessed via Caching-Inhibited and Guarded mapping.

6–5. The INIT signals defined in Open PIC must be connected to Soft Reset pins on PowerPC processors.

6–6. Interrupts must be disabled at the CHRP interrupt controller at the point of transfer of control to the operating system.

Software Implementation Note: Private access interface to the per-processor registers is optional. Software should not depend on it being available.

Software Implementation Note: The Who Am I register might not be implemented in a CHRP interrupt controller. An alternative system-specific method may be defined, instead, for a processor to find out its own Open PIC identity for use in specifying interrupt destinations. For an example of such a method, see Section Section 12.2, “An SMP Boot Process,” on page 218.

Hardware Implementation Note: Allocation of PCI interrupts is discussed in Section 9.1.3, “Assignment of Interrupts to PCI Devices,” on page 152.

Hardware Implementation Note: The number of I/O interrupts supported are platform dependent.

6.2 Distributed Implementation — A Proposal

It is an intention of the CHRP architecture to allow a variety of system structures and sizes. It is in keeping with this intention to expect system structures that include multiple I/O busses which are physically distant from each other and from the processors themselves. It would be impractical in such systems to have an interrupt controller implemented as a single centrally placed integrated circuit. The following distributed implementation of the Open PIC architecture is therefore proposed. The CHRP architecture provides addressing infrastructure to support it.

Personal Use Copy - Not for Reproduction

The interrupt controller is logically divided into two partitions: The I/O interrupt source partition containing the Interrupt Source Registers that accepts I/O interrupt signals and translates them into the vector, priority, and destination information to be presented to processors; and another partition that contains the rest of the controller functionality. The I/O interrupt source partition may be further partitioned into multiple sub-partitions. Each sub-partition represents an I/O interrupt group. An I/O interrupt group is defined by its lowest interrupt number and the number of interrupts in the group. These are specified via Open Firmware Interrupt Controller properties. See section on Open PIC interrupt controller node properties in *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10].

The interrupt controller may be partitioned into one or more physical units: A base unit called the Interrupt Delivery Unit (IDU), which houses the global, timer-related, and public-access per-processor registers, contains logic to deliver interrupts to processors, and may optionally house a single I/O interrupt source sub-partition. Additionally, there may be zero or more external Interrupt Source Units (ISUs), each housing Interrupt Source Registers for an I/O interrupt group.

The IDU and the external ISUs may communicate via a system-dependent means. Figure 11 on page 88 shows a system map with the IDU containing an I/O interrupt group and one external ISU.

Personal Use Copy - Not for Reproduction

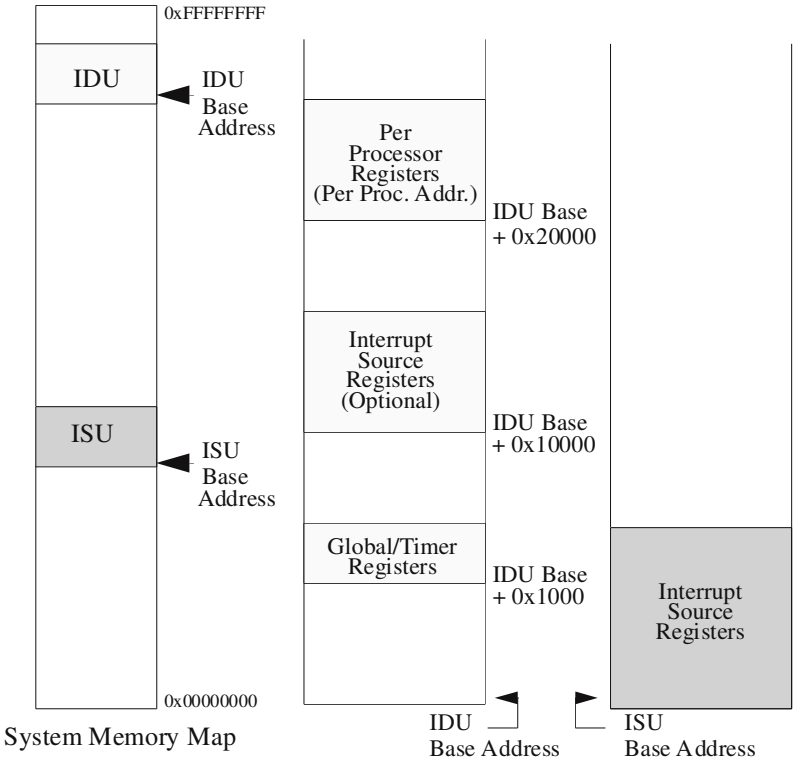

Figure 11. System Memory Map Showing Mapping of the IDU and an ISU

The IDU and the ISUs are individually mapped, in a non-overlapped manner, in the system memory. Typically, an external ISU will be mapped within the address range occupied by the bus, from whose devices it receives interrupts. Open Firmware properties for the interrupt controller specify the base addresses for the IDU and for the external ISUs, if any. See section on Open PIC interrupt controller node properties *PowerPC Microprocessor Common Hardware Reference Platform System binding to: IEEE Std 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [10].

The relative offsets of register addresses within the IDU with respect to its base address are identical to those specified in the Open PIC Specification. In the ISU, the ISRs for interrupt group are mapped contiguously starting at the

Personal Use Copy - Not for Reproduction